BLAISE PASCAL MAGAZINE 113

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

Blaise Pascal

Blaise Pascal Magazine 113 2023

Photo by Alexx Cooper on Unsplash

https://unsplash.com/@hialexxlarioss

BLAISE PASCAL MAGAZINE 113
Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web Apps Android / IOS / Mac / Windows & Linux

CONTENT

ARTICLES

Photo by Alexx Cooper on Unsplash
https://unsplash.com/@hialexxlarioss

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

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

CONTRIBUTORS

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

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

Subscriptions can be taken out online at www.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 (Stichting Ondersteuning Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Ondersteuning Programmeertaal Pascal)
Subscription department Edelstenenbaan 21 / 3402 XA IJsselstein, Netherlands Mobile: + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

SUBSCRIPTIONS (2023 prices)	Internat. excl. VAT	Internat. incl. 9% VAT	Shipment	TOTAL
Printed Issue (8 per year) **±60 pages :**	**€ 200**	**€ 218**	**€ 130**	**€ 348**
Electronic Download Issue (8 per year) **±60 pages :**	**€ 64,22**	**€ 70**		

From your editor

Hello to all of you,
to begin with: I have changed the layout to a much more workable and readable layout: make it flatter – less colorful – larger space between the lines and a sharper font.
The coding values have been changed and I hope that all of you are satisfied with this new layout.
This was the outcome of a number of discussions I had with readers.
Some loved the colorful, but I thought when you want to work with the text it should be as easy to read as can be – with little contrast, more clearly arranged.
Of course it now is a much duller in appearance.
But that's one or the other.
So now I ask you tell me if you like this new layout or what would be your opinion.
Let me know at: `editor@blaisepascal.eu.`
I hope you will agree that we always want better and therefore we have to change things.

We were forced to increase the price of the **printed version** due to the significant increase in shipping costs. The awful thing is that every issue I ship costs €10. The cost of each shipment would still exceed 50% of the overall cost even if I could reduce the cost of printing. So, it will be an extremely expensive endeavor. I have no idea how to overcome this.

To make our services better we have included the internet version of the "**Library of all issues**" of our Magazine – only for reading. If you want the complete library with all downloads and code you will have to buy the "**LIB-stick on USB**" – **credit card**.

Right at this moment we are building the new version of our website and we would like you to tell us what kind of service you would want to be added. I would like to persuade you to write an article. There are may of you with brilliant ideas. Let the world know. We will help you with writing your article.

In this issue "**Michalis Kamburelis**" wrote about his gaming engine and now how to program (*code*) the game. That is wonderful. It will now be possible to do this in an easy way.
Maybe you can even write your own game? Ever thought of combining a user interface with 3D rendered moving parts?
So that the user will be surprised with easy examples of the use of your application interface?
Playful learning and teaching? Take a look at the fantastic and very convincing example of 3D animals.They are stunningly beautiful and available for use.
`https://sketchfab.com/features/free-3d-models`
have a lot of examples for free and of course you can create your own, or look for specific designed models:
`https://youtu.be/NTaHgf7okwk` - a walking cheetah.
I placed a very short video of the bad chess game on our website. Just take look. Its very convincing and it all can be done in **Pascal**: in **Delphi** as well in **Lazarus**.

There has been a tremendous development: **Webassembly for FreePascal**.
We now have been able to **run FPC inside Webassembly** and that means that a whole lot of new techniques will become available as well for **Pas2js**.
We will write about this in the next issue 114/115
to explain all the coming techniques and show you samples of what we achieved.
Embarcadero has promised to come up with a new version 12. I think it will be presented next month, and so I hope to show you a lot of new stuff from Delphi.

Please let me know your ideas about this item, and thank you for reading.

Detlef

POCKET PACKAGE (2BOOKS) EXCLUDING VAT AND SHIPPING

LAZARUS HANDBOOK PRICE: € 25,00

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

MIGRATE TO 64-BIT MAX-BOX

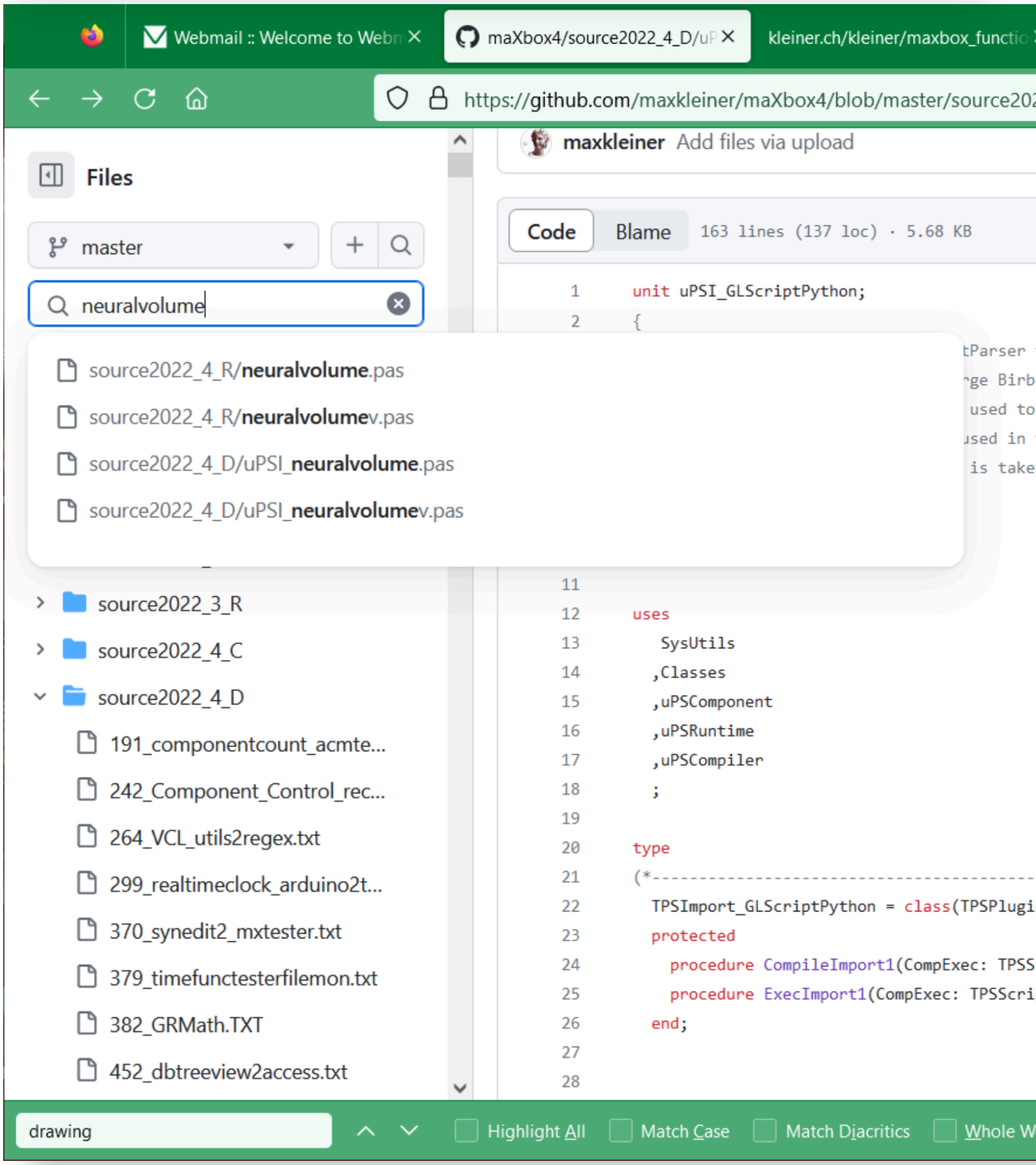
CONVERT A 32-BIT APPLICATION TO 64-BIT FROM DELPHI 2007 TO DELPHI 10.4 SYDNEY.

Starter Expert D11

maXbox Starter 113
"Lost in translation – ghost in application" - Hardcore code.
Source: `firstdemo_master11_cop21web.txt` *and* `my6 cannonball.txt` blog

INTRODUCTION

If you have a code base of 32-bit **Windows** Delphi applications that you want to convert to 64-bit Windows, you should first do a reorganisation of the sources to get an overview.
The Source is organised in **_C** for **Components** and **_R** for **Runtime** (*native Units*) and **_D** for **Design Units** (*script mapping imports*) like the following graph of Package **Neuralvolume** of **CAI NeuralNetwork** shows as 4 files:



https://github.com/maxkleiner/maXbox4/blob/master/source20

maxkleiner Add files via upload

Files

master

neuralvolume

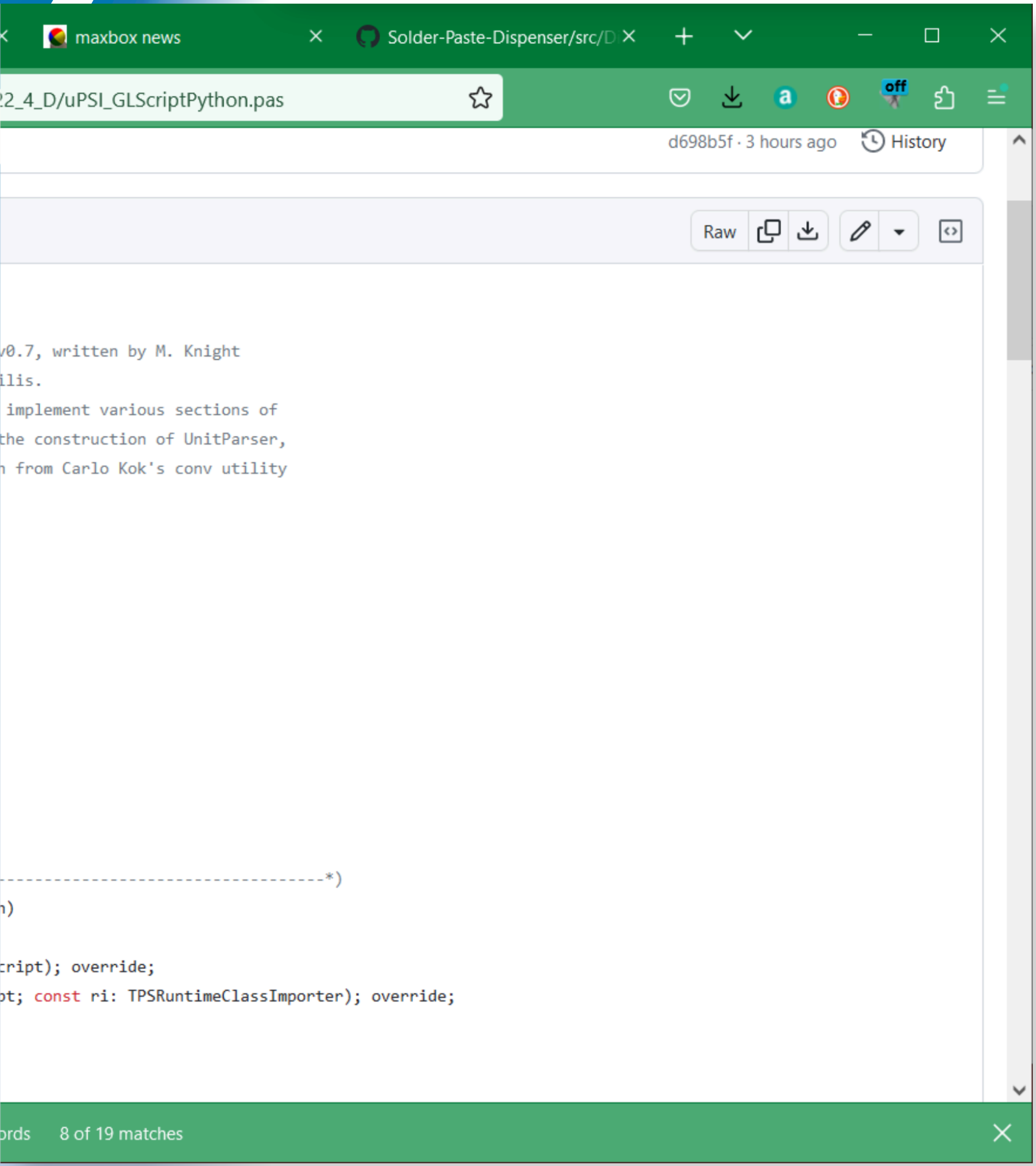
source2022_4_R/**neuralvolume**.pas
source2022_4_R/**neuralvolume**v.pas
source2022_4_D/uPSI_**neuralvolume**.pas
source2022_4_D/uPSI_**neuralvolume**v.pas

source2022_3_R
source2022_4_C
source2022_4_D
191_componentcount_acmte...
242_Component_Control_rec...
264_VCL_utils2regex.txt
299_realtimeclock_arduino2t...
370_synedit2_mxtester.txt
379_timefunctesterfilemon.txt
382_GRMath.TXT
452_dbtreeview2access.txt

Code Blame 163 lines (137 loc) · 5.68 KB

```
unit uPSI_GLScriptPython;
{
...
uses
   SysUtils
  ,Classes
  ,uPSComponent
  ,uPSRuntime
  ,uPSCompiler
  ;

type
(*-----------------------------------------
  TPSImport_GLScriptPython = class(TPSPlugi
  protected
    procedure CompileImport1(CompExec: TPSS
    procedure ExecImport1(CompExec: TPSScri
  end;
```

So its not that easy open your 32-bit application in the IDE, add and activate the 64-bit Windows target platform, and compile your application as a 64-bit Windows application.
While digging or diving through the source code of maXbox4 it seems to be impossible to migrate over 3300 units (exactly 3335) in a decent and proper way to maXbox5 aka 64.bit Version.

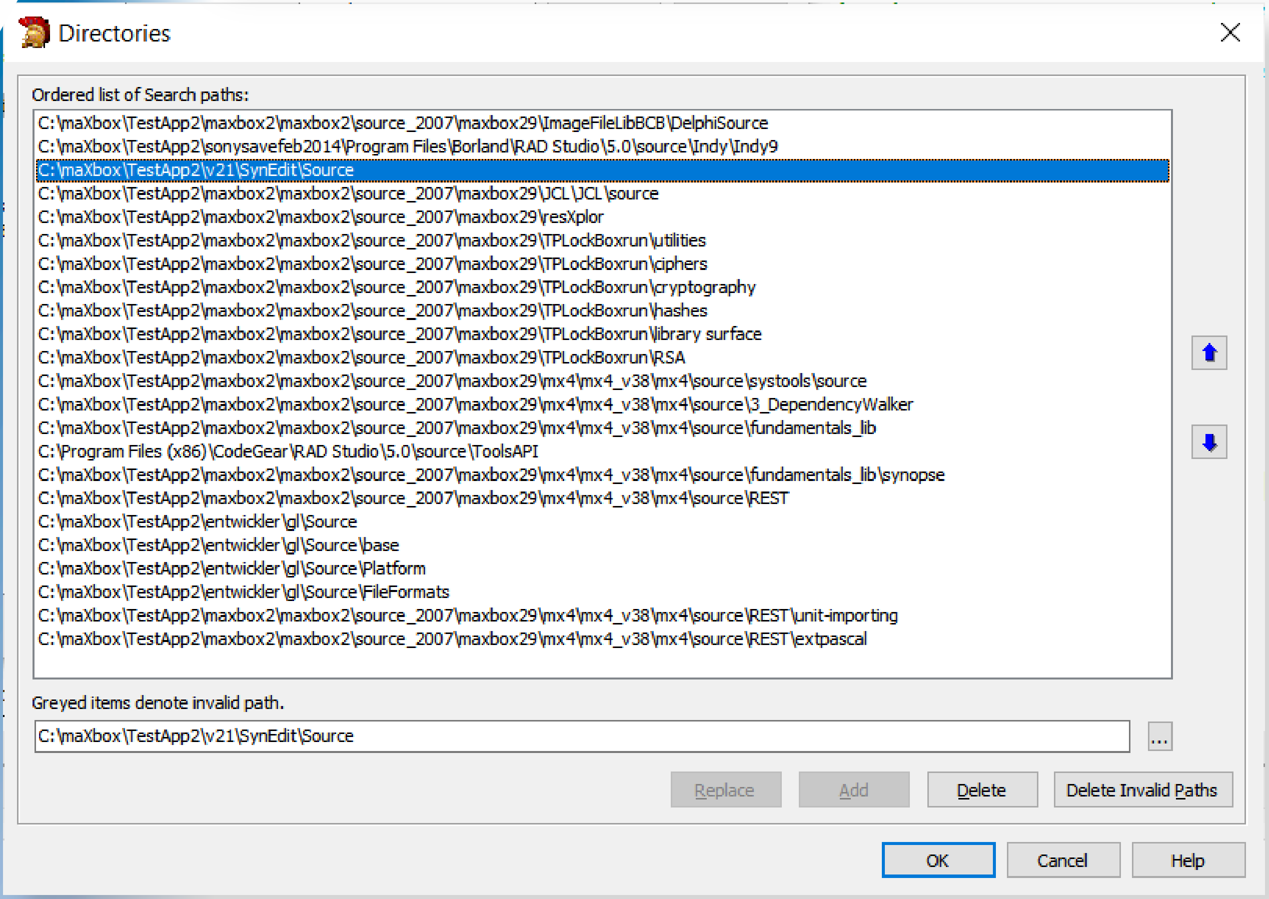


```
v0.7, written by M. Knight
ilis.
 implement various sections of
the construction of UnitParser,
n from Carlo Kok's conv utility

----------------------------------*)
n)

cript); override;
pt; const ri: TPSRuntimeClassImporter); override;
```

SOURCE ORGANISATION

First thing I must say is the missing support of any kind of the whole "**HiRes/4K or DPI Awareness Resolution and Delphi forms**" revolution;
There's no “Make my form look right on all resolutions" checkbox or a emulator which go through all forms. But you can drawback to the "don't support hi-DPI" setting.
I know this is not the improvement we want, but this causes the least headaches.
As in **mX4** and for the forthcoming **mX5** the App is “out of the box” (*self containment*) and needs no installation nor registration. It has a independent system architecture (**ISA**).

So for the reorganisation of the sources I have the latest revision with patches from **issue #202** (*commit 86a057c*) but I am unable to compile the files at first (*Core_D26*) that are part of the **`PascalScript_Core_D27.dpk`** for that platform for **Linux64, Win64 nor MacOS64**.



Ordered list of Search paths:

```
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\ImageFileLibBCB\DelphiSource
C:\maXbox\TestApp2\sonysavefeb2014\Program Files\Borland\RAD Studio\5.0\source\Indy\Indy9
C:\maXbox\TestApp2\v21\SynEdit\Source
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\JCL\JCL\source
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\resXplor
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\utilities
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\ciphers
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\cryptography
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\hashes
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\library surface
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\TPLockBoxrun\RSA
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\systools\source
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\3_DependencyWalker
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\fundamentals_lib
C:\Program Files (x86)\CodeGear\RAD Studio\5.0\source\ToolsAPI
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\fundamentals_lib\synopse
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\REST
C:\maXbox\TestApp2\entwickler\gl\Source
C:\maXbox\TestApp2\entwickler\gl\Source\base
C:\maXbox\TestApp2\entwickler\gl\Source\Platform
C:\maXbox\TestApp2\entwickler\gl\Source\FileFormats
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\REST\unit-importing
C:\maXbox\TestApp2\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\source\REST\extpascal
```

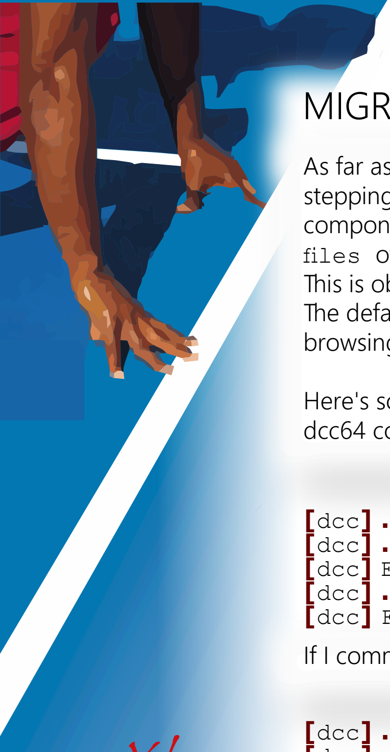
Greyed items denote invalid path.

```
C:\maXbox\TestApp2\v21\SynEdit\Source
```

In **Delphi**, I can include a folder's source code by adding it to the **project Search Path,** or adding it to the **Library Path**. The **Search Path** applies only to the current project, while the Library Path applies to any project opened with the **IDE**.

But other than that, is there no functional difference between the Search and Library paths?

The reason is I have a folder X with source used by project A. When I include that folder under Project A's search path, it says it cannot find a specific file in that folder. When I include it under the Library path, then test project A compiles fine.



As far as I know, browsing path is where the debugger should look for files when breaking/ stepping into source files that's not in the library path. Lets say that you have a third-party component that you use. You point the **library path** to the directory where the pre-compiled `dcu-files` of that component are placed. Your project will use these dcu-files when you compile.
This is obvious, because it wont be recompiled every time you do a build.
The default settings for the VCL show this. In **library path** they have put `$(BSD)\Lib`, and in the browsing path they have put `$(BDS)\SOURCE\WIN32`...

Here's some compiler output at first to compare using **Delphi 10.3.2 Rio** and then **10.4 dccosx64** or dcc64 compiler (*similar results exist for dcclinux64*):

```
[dcc] ./PascalScript_Core_D26.dpk
[dcc] ./uPSUtils.pas (730)
[dcc] Error: E2008 Incompatible types
[dcc] ./uPSCompiler.pas (1374)
[dcc] Fatal: F2063 Could not compile used unit 'uPSUtils.pas'
```

If I comment out that offending code then I get the following which is starting to look non-trivial...

```
[dcc] ./PascalScript_Core_D26.dpk
[dcc] ./uPSRuntime.pas (8923)
[dcc] Warning: W1057 Implicit string cast from 'AnsiString' to 'string'
[dcc] ./uPSRuntime.pas (11640)
[dcc] Error: E1025 Unsupported language feature: 'ASM'
[dcc] ./uPSRuntime.pas (11640)
[dcc] Error: E2029 ';' expected but 'ASM' found
[dcc] ./uPSRuntime.pas (11640)
[dcc] Warning: W1011 Text after final 'END.' - ignored by compiler
[dcc] ./uPSRuntime.pas (58)
[dcc] Error: E2065 Unsatisfied forward or external declaration: 'TPSProcRec.Create'
```

The reason of all problems in OSX64 (*and* **Linux64**, *i think also*) with PS is "
The `LongInt` and `LongWord` Data Type are different on 64-bit **POSIX*** platforms.
To keep interoperability between **Delphi** and **POSIX* API**, for 64-bit **POSIX*** platforms, the size of `LongInt` and `LongWord` types are changed to 64-bit. All 32-bit platforms and 64-bit **Windows** platforms keep 32-bit for the `LongInt` and `LongWord` types."

So, fixing can be very simple - `change ALL LongInt type to Integer. Files: uPSCompiler, uPSComponent, uPSDebugger, uPSRuntime, uPSUtils.`

But don't change it by auto-replace from `"Longint" to "Integer", because in this case declarations like AddTypeCopyN('Integer', 'LongInt');`
and others – as the **reference as string literal will be broken**.

I stubbed out 2 **assembler** routines which I hope could be translated to pure **pascal** by someone who understands their intent. I'm not really sure what the assembler is doing

```
procedure MyAllMethodsHandler;
procedure PutOnFPUStackExtended(ft: extended);
```

Perhaps as suggested we can just `{$DEFINE empty_methods_handler}` to avoid the **assembler**.
But I don't know what its trying to do, so is a stub acceptable or do we need it to do something?
As you can see the work for the 64bit box has begun but libs, maps, object-files, transpiler and registering is full of traps. If "**Use Debug DCUs**" option is not activated, and I debug our application, I can only single step through my own code. This is what we want in the most cases, because it is our code that is buggy, not **Delphi's** code normally. It will be quite annoying to keep stepping into **Delphi's** code.

The **Portable Operating System Interface (POSIX; IPA*)** *is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines both the system and user-level application programming interfaces (APIs), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems. POSIX is also a trademark of the IEEE. POSIX is intended to be used by both application and system developers.*
The **International Phonetic Alphabet (IPA)** *is an alphabetic system of phonetic notation based primarily on the Latin script.*

REORGANISATION

After restructured the source from **Delphi 2007** (*see pic 2 article page5*) to **Github** and configured in **Delphi 10.4 I** began to review and handle the following issues (*mostly related to pointed operations in* **WinAPI** *issues,* **NativeInt** *size, and* **Assembly** *code*):

If you pass pointers to `SendMessage/PostMessage/TControl.Perform`, the `wParam` and `lParam` parameters should be type-casted to the `WPARAM/LPARAM` type
and not to `Integer/Longint`.

Correct: `SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));`
Wrong: `SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));`
Replace `SetWindowLong/GetWindowLog` with `SetWindowLongPtr/GetWindowLongPtr`

for `GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDPARENT` and `GWLP_WNDPROC` as they return pointers and handles.
Pointers that are passed to `SetWindowLongPtr` should be type-casted to `LONG_PTR` and not to `Integer/Longint`.

Correct: `SetWindowLongPtr(hWnd, GWLP_WNDPROC,LONG_PTR(@MyWindowProc));`
Wrong: `SetWindowLong(hWnd, GWL_WNDPROC, Longint(@MyWindowProc));`

In the **runtime library** several issues had to be done:

```
 <?xml version="1.0" encoding="UTF-8"?>
{$IFNDEF WIN32}
  'This components are for 32bitDelphi only!'???›??
{$ENDIF}
```

Review the `IFNDEF WIN32` in the sense: try to convert or leave!
In **Delphi**, strings as result values are treated like var parameters.
In other words, a function like **`Foo`** is in fact compiled as:

```
function Foo(): String;
begin
  Result := 'foo';
  RaiseException('...');
end;
procedure Foo(var Result: string);
begin
  Result := 'Foo';
  RaiseException(...);
end;
```

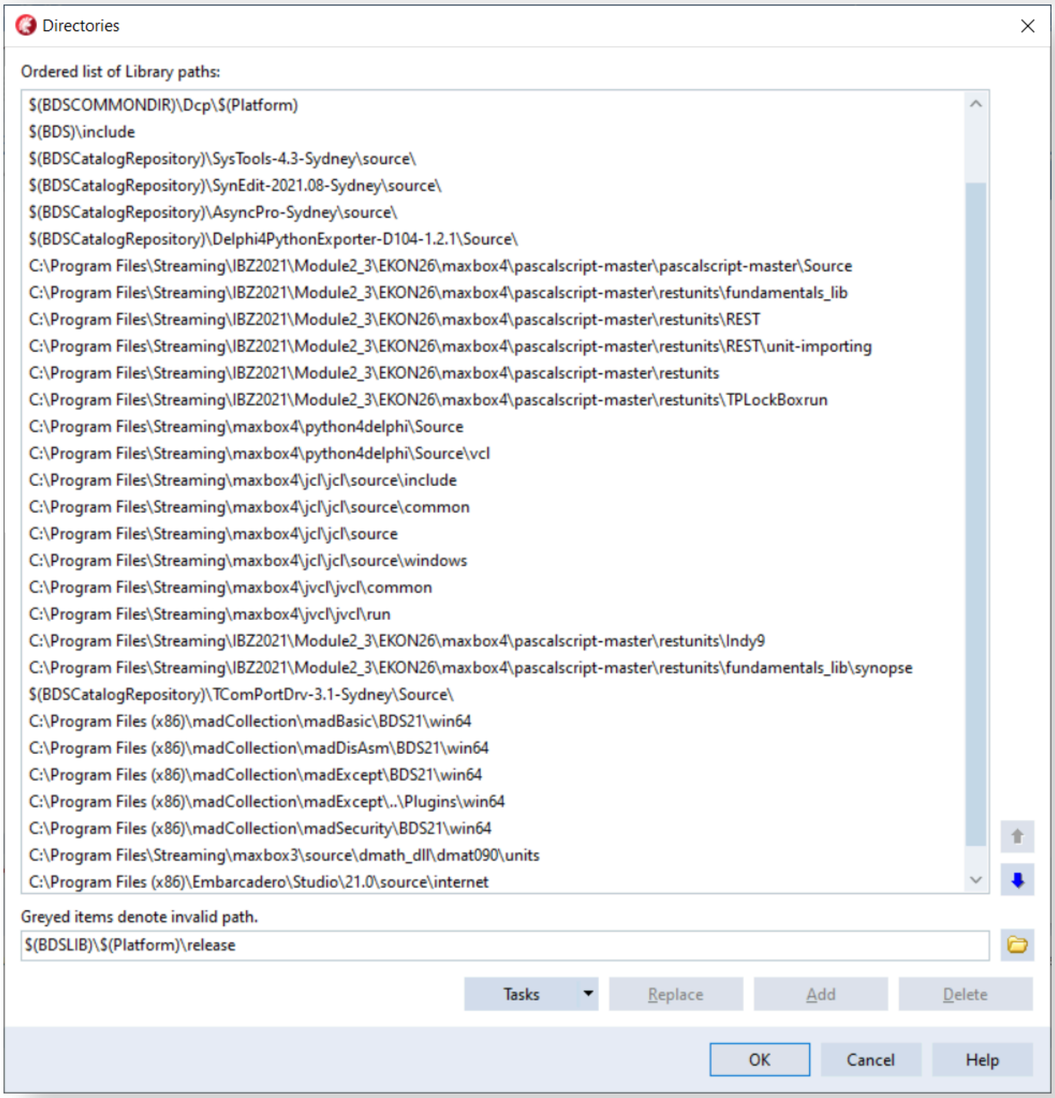
So the chance to convert lines of **WIN32** to **WIN64** is possible with references or type-less references like procedure `Foo(var Result;);`
`Overloads`: For functions that took `PChar`, there are now `PAnsiChar` and `PWideChar` versions so the appropriate function gets called. `AnsiXXX` functions are a consideration:

- **`SysUtils.AnsiXXXX`** functions, such as AnsiCompareStr:
- **They remain declared** with string and float to UnicodeString.
- **Offer better** backward compatibility (*no need to change code*).

The `AnsiStrings` unit's `AnsiXXXX` functions offer the same capabilities as the SysUtils. `AnsiXXXX` functions, but work only for `AnsiString`. Also, the `AnsiStrings.AnsiXXXX` functions provide better performance for an `AString` than `SysUtils. AnsiXXXX` functions, which work for both `AnsiString` and `UnicodeString`, because no implicit conversions are performed.String information functions:

- **`StringElementSize`** returns the actual data size.
- **`StringCodePage`** returns the code page of string data.
- **`System.StringRefCount`** returns the reference count.

The **RTL** provides many helper functions that enable users to do explicit conversions between code pages and element size conversions. If developers are using the `Move` function on a `character array`, they cannot make assumptions about the element size. Much of this problem can be mitigated by making sure all `RValue` references generate the proper calls to **RTL** to ensure proper element sizes. In the meantime I got the import and list in **D10.4**:



Ordered list of Library paths:

```
$(BDSCOMMONDIR)\Dcp\$(Platform)
$(BDS)\include
$(BDSCatalogRepository)\SysTools-4.3-Sydney\source\
$(BDSCatalogRepository)\SynEdit-2021.08-Sydney\source\
$(BDSCatalogRepository)\AsyncPro-Sydney\source\
$(BDSCatalogRepository)\Delphi4PythonExporter-D104-1.2.1\Source\
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\pascalscript-master\Source
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\fundamentals_lib
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\REST
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\REST\unit-importing
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\TPLockBoxrun
C:\Program Files\Streaming\maxbox4\python4delphi\Source
C:\Program Files\Streaming\maxbox4\python4delphi\Source\vcl
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\include
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\common
C:\Program Files\Streaming\maxbox4\jcl\jcl\source
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\windows
C:\Program Files\Streaming\maxbox4\jvcl\jvcl\common
C:\Program Files\Streaming\maxbox4\jvcl\jvcl\run
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\Indy9
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\fundamentals_lib\synopse
$(BDSCatalogRepository)\TComPortDrv-3.1-Sydney\Source\
C:\Program Files (x86)\madCollection\madBasic\BDS21\win64
C:\Program Files (x86)\madCollection\madDisAsm\BDS21\win64
C:\Program Files (x86)\madCollection\madExcept\BDS21\win64
C:\Program Files (x86)\madCollection\madExcept\..\Plugins\win64
C:\Program Files (x86)\madCollection\madSecurity\BDS21\win64
C:\Program Files\Streaming\maxbox3\source\dmath_dll\dmat090\units
C:\Program Files (x86)\Embarcadero\Studio\21.0\source\internet
```

Greyed items denote invalid path.

`$(BDSLIB)\$(Platform)\release`

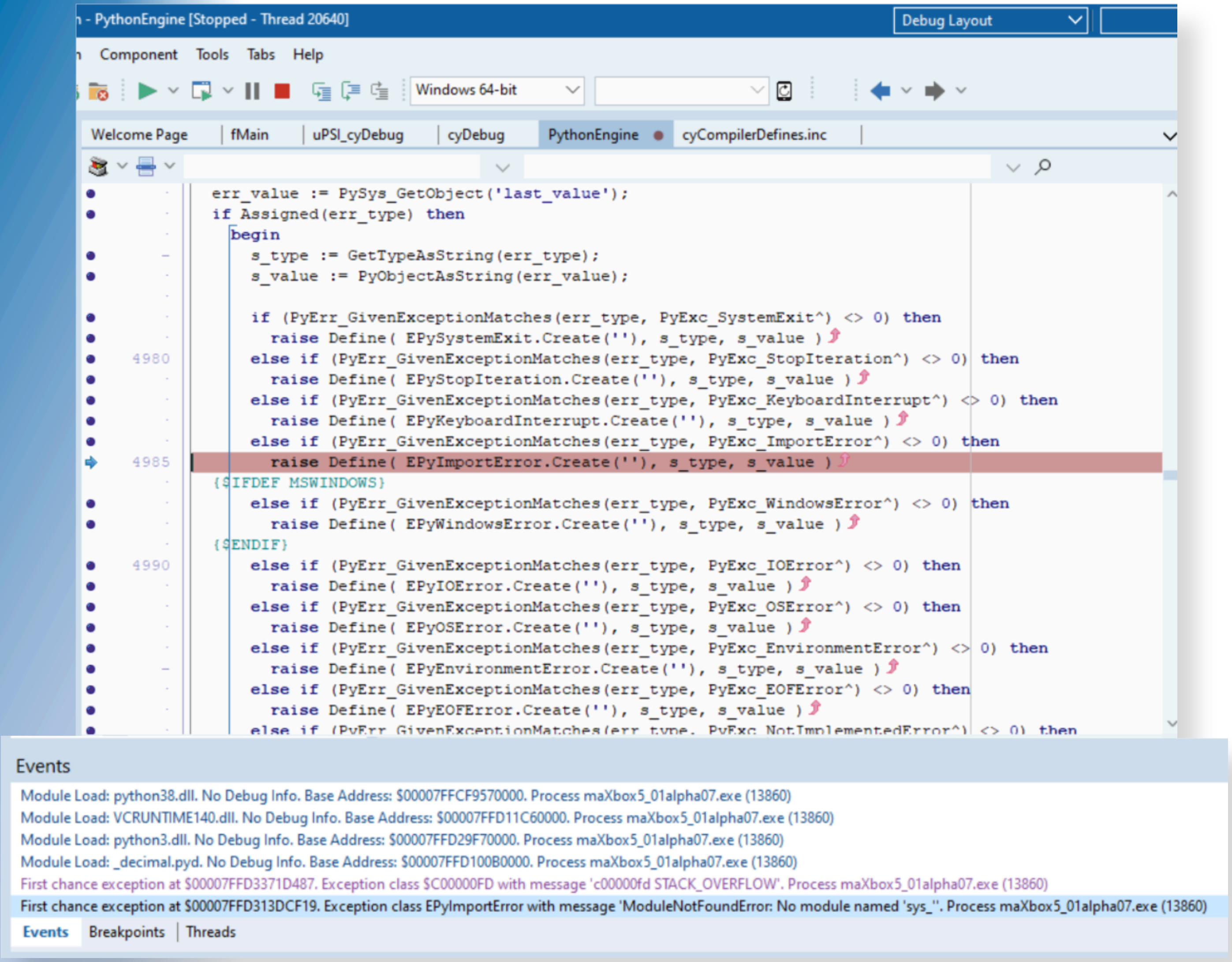
A big mess was or is the Tencoding (not finished yet) because Tencoding makes the real difference from ANSI to unicode:

- Defaults to users' active code page.
- Supports **UTF-8.**
- Supports **UTF-16**, big and little endian.
- **Byte Order Mark (BOM)** support.
- You can create descendent classes for user-specific encodings.

You need to perform these steps:

❶ Review char- and string-related functions.
❷ Rebuild the application.
❸ Review surrogate pairs.
❺ Review string payloads.

Night after night I got many **AV's (Access Violation)** of these type:
Exception code `0xc0000005` is an **Access Violation**. An **AV** at fault offset `0x00000000` means that something in your service's code is accessing a nil pointer. You will just have to debug the service while it is running to find out what it is accessing. If you cannot run it inside a debugger, then at least install a third-party exception logger framework, such as **EurekaLog** or **MadExcept**, to find out what your service was doing at the time of the **AV**. Most of the single time you get an **AV** in a unit with a `initialisation` section of e.g. a `Setmem` or some memory allocation stuff.


- PythonEngine [Stopped - Thread 20640]
Debug Layout
Component Tools Tabs Help
Windows 64-bit
Welcome Page | fMain | uPSI_cyDebug | cyDebug | PythonEngine | cyCompilerDefines.inc

```
  err_value := PySys_GetObject('last_value');
  if Assigned(err_type) then
    begin
      s_type := GetTypeAsString(err_type);
      s_value := PyObjectAsString(err_value);

      if (PyErr_GivenExceptionMatches(err_type, PyExc_SystemExit^) <> 0) then
        raise Define( EPySystemExit.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_StopIteration^) <> 0) then
        raise Define( EPyStopIteration.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_KeyboardInterrupt^) <> 0) then
        raise Define( EPyKeyboardInterrupt.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_ImportError^) <> 0) then
        raise Define( EPyImportError.Create(''), s_type, s_value )
{$IFDEF MSWINDOWS}
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_WindowsError^) <> 0) then
        raise Define( EPyWindowsError.Create(''), s_type, s_value )
{$ENDIF}
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_IOError^) <> 0) then
        raise Define( EPyIOError.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_OSError^) <> 0) then
        raise Define( EPyOSError.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_EnvironmentError^) <> 0) then
        raise Define( EPyEnvironmentError.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_EOFError^) <> 0) then
        raise Define( EPyEOFError.Create(''), s_type, s_value )
      else if (PyErr_GivenExceptionMatches(err_type, PyExc_NotImplementedError^) <> 0) then
```

Events

```
Module Load: python38.dll. No Debug Info. Base Address: $00007FFCF9570000. Process maXbox5_01alpha07.exe (13860)
Module Load: VCRUNTIME140.dll. No Debug Info. Base Address: $00007FFD11C60000. Process maXbox5_01alpha07.exe (13860)
Module Load: python3.dll. No Debug Info. Base Address: $00007FFD29F70000. Process maXbox5_01alpha07.exe (13860)
Module Load: _decimal.pyd. No Debug Info. Base Address: $00007FFD100B0000. Process maXbox5_01alpha07.exe (13860)
First chance exception at $00007FFD3371D487. Exception class $C00000FD with message 'c00000fd STACK_OVERFLOW'. Process maXbox5_01alpha07.exe (13860)
First chance exception at $00007FFD313DCF19. Exception class EPyImportError with message 'ModuleNotFoundError: No module named 'sys_''. Process maXbox5_01alpha07.exe (13860)
```

This means that an exception was thrown, but there's no catch handler for it anywhere.
This is most likely a programming error, probably on your part. It looks like you called an **OpenCV** or **API** function which failed by throwing a **CV::Exception**, but you're not catching it.
This would normally lead to a crash, but since you're running inside a debugger, you get the option to ignore this exception. That's what the **Continue** button will do on this dialogue. So instead of throwing an exception, the code will just continue executing as if nothing had happened. This is likely to fail eventually, as an error condition has now been ignored.
A real horror was to convert the `Format() Function` not in **Delphi** but to work in **PascalScript64** at runtime in a script, so here's the function in **Delphi** in system **RTL**:

```
Function Format(fmt : String; params : array of const) : String;
var
  pdw1, pdw2 : PDWORD;
  i : integer;
  pc : PChar;
begin
  pdw1 := nil;
  if length(params) > 0 then GetMem(pdw1, length(params) * sizeof(Pointer));
  pdw2 := pdw1;
  for i := 0 to high(params) do begin
    pdw2^ := DWORD(PDWORD(@params[i])^);
    inc(pdw2);
  end;
  GetMem(pc, 1024 - 1);
  try
    SetString(Result, pc, wvsprintf(pc, PwideCHAR(fmt),
                         PwideCHAR(pdw1)));   // fix from pchar?
  except
    Result := #0;
  end;
  if (pdw1 <> nil) then FreeMem(pdw1);
  if (pc <> nil) then FreeMem(pc);
end;
```

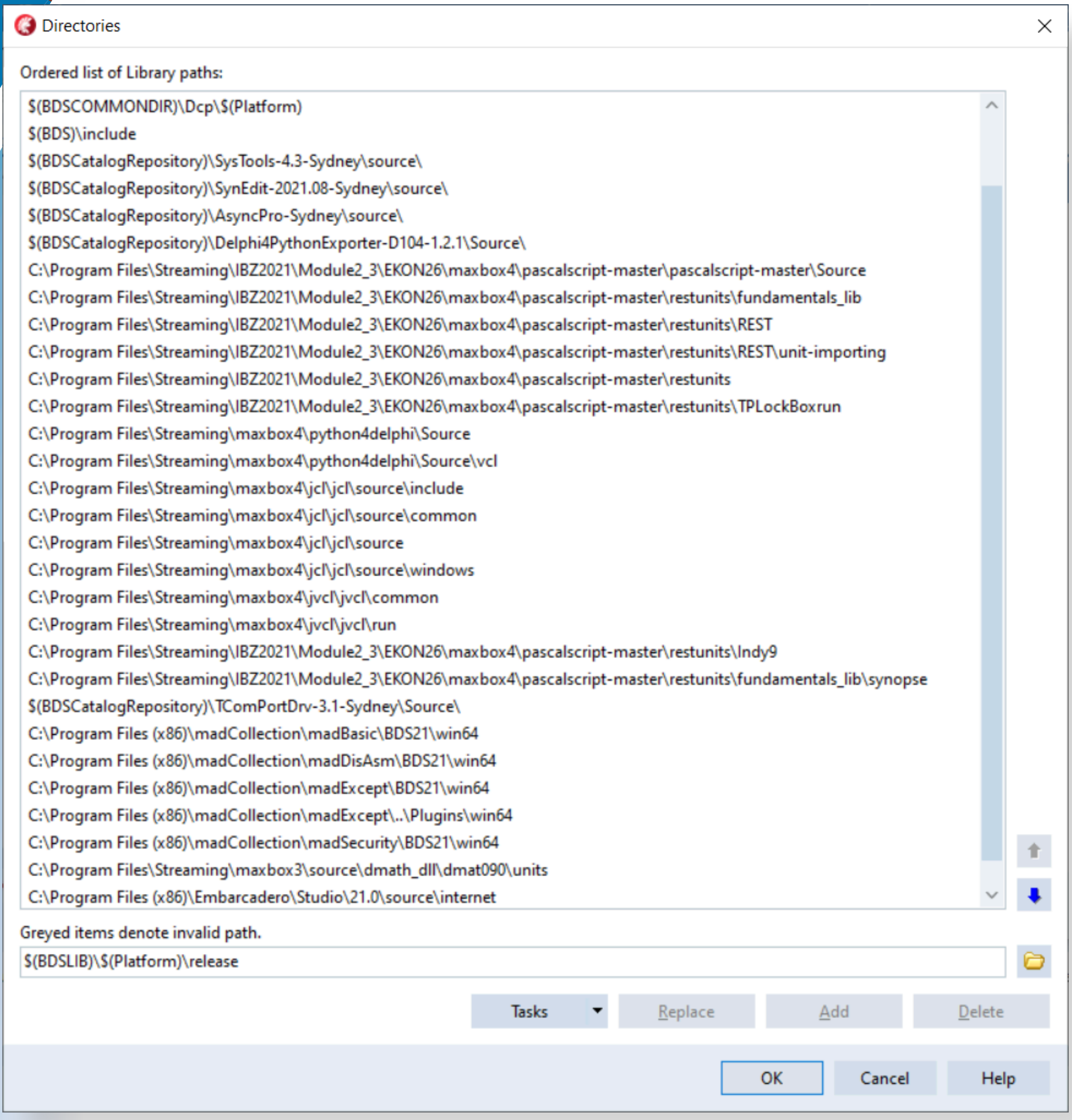
At the core its a `wvsprintf` but the question was unicode available or not? The **Byte Order Mark (BOM)** should be added to files to indicate their encoding and the function returns a string.
After reimport and rebuild and cast to **PwideCHAR** it works now.

COMPATIBILITY COMPILATION

When running a **32bit** process, on a **64bit** version of **Windows**,- having the large address aware flag set - `pointers` in the `2-4GB` range are valid. In that case the request of a `varInt64` could, if I'm not mistaken, result in positive values in the `2-4GB` range being returned. If `NativeInt` is a signed `32bit int`, that would then result in a range violation. I'm not sure if that `NativeInt / NativeUInt` cast in between here is needed at all. `NativeInt` and `NativeUInt` are `signed/ unsigned` and their size is of the targeted platform, hence 32 or 64 bits.
That's why I believe the cast changes are not needed.

Most of the times you deal with pointers or assembler code like:

```
function StrToWord(const Value: String): Word;
begin
   := Word(pointer(@Value[1])^);
end;

function WordToStr(const Value: Word): WordStr;
begin
  SetLength(Result, SizeOf(Value));
  Move(Value, Result[1], SizeOf(Value));
end;
```



Ordered list of Library paths:

```
$(BDSCOMMONDIR)\Dcp\$(Platform)
$(BDS)\include
$(BDSCatalogRepository)\SysTools-4.3-Sydney\source\
$(BDSCatalogRepository)\SynEdit-2021.08-Sydney\source\
$(BDSCatalogRepository)\AsyncPro-Sydney\source\
$(BDSCatalogRepository)\Delphi4PythonExporter-D104-1.2.1\Source\
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\pascalscript-master\Source
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\fundamentals_lib
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\REST
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\REST\unit-importing
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\TPLockBoxrun
C:\Program Files\Streaming\maxbox4\python4delphi\Source
C:\Program Files\Streaming\maxbox4\python4delphi\Source\vcl
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\include
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\common
C:\Program Files\Streaming\maxbox4\jcl\jcl\source
C:\Program Files\Streaming\maxbox4\jcl\jcl\source\windows
C:\Program Files\Streaming\maxbox4\jvcl\jvcl\common
C:\Program Files\Streaming\maxbox4\jvcl\jvcl\run
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\Indy9
C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\restunits\fundamentals_lib\synopse
$(BDSCatalogRepository)\TComPortDrv-3.1-Sydney\Source\
C:\Program Files (x86)\madCollection\madBasic\BDS21\win64
C:\Program Files (x86)\madCollection\madDisAsm\BDS21\win64
C:\Program Files (x86)\madCollection\madExcept\BDS21\win64
C:\Program Files (x86)\madCollection\madExcept\..\Plugins\win64
C:\Program Files (x86)\madCollection\madSecurity\BDS21\win64
C:\Program Files\Streaming\maxbox3\source\dmath_dll\dmat090\units
C:\Program Files (x86)\Embarcadero\Studio\21.0\source\internet
```

Greyed items denote invalid path.

```
$(BDSLIB)\$(Platform)\release
```

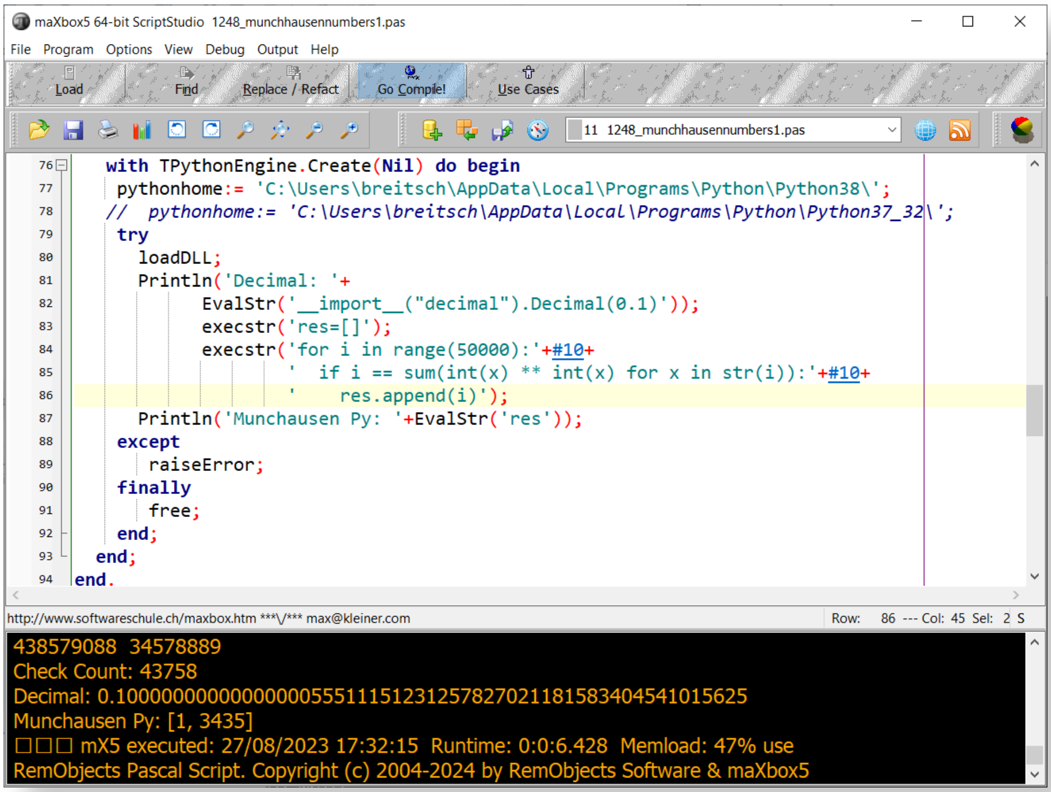
Progress has been made in that I've compiled the **TestApplication** sample with **CrossVCL 1.27** for **Mac64** and **Linux64**.

When I choose a second compile in a **CrossVCL** from the menu I receive the following error.

First chance exception at `$0000000100419E9E.`

```
Exception class EAccessViolation with message 'Access violation at
address 0000000100419E9E, accessing address 00000009017241F8'. Process
TestApplication (5741)
Source Breakpoint at : C:\Program Files\Streaming\IBZ2021\Module2_
3\EKON26\maxbox4\pascalscript-master\pascalscript-
master\Source\uPSRuntime.pas line 2060. Process TestApplication (5741)

Upsruntime.TPSExec.Clear()(0x00000002017350d0)
Upsdebugger.TPSCustomDebugExec.Clear()(0x00000002017350d0)
Upscomponent.TPSScript.Compile()(0x0000000201734c20)
Fmain.TForm1.Compile1Click(System.TObject*)(0x00007ffeefbfe038)
Vcl.Menus.TMenuItem.Click()(0x0000000201734960)
Vcl.Menus.TMenu.DispatchCommand(unsigned short)(0x0000000201734340,2)
Vcl.Forms.TCustomForm.WMCommand(Winapi.Messages.
TWMCommand&)(0x0000000205039ff0,0x00007ffeefbfe878)
:000000010001132B System::TObject::Dispatch(void*)
```

And then maybe by intuition I made a build and the **AD** has gone away and I got my first screen, compiled and script executed:


maXbox5 64-bit ScriptStudio 1248_munchhausennumbers1.pas
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compile!
Use Cases
11 1248_munchhausennumbers1.pas
http://www.softwareschule.ch/maxbox.htm ***\/*** max@kleiner.com
Row: 86 --- Col: 45 Sel: 2 S

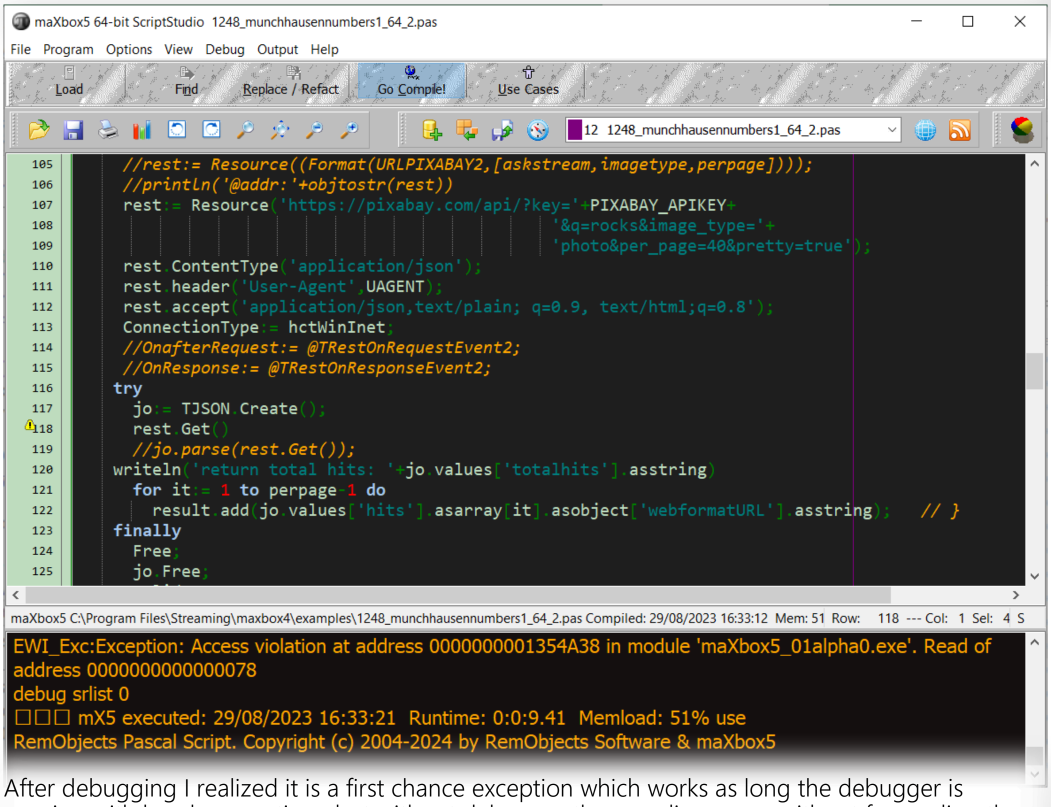
```
 with TPythonEngine.Create(Nil) do begin
  pythonhome:= 'C:\Users\breitsch\AppData\Local\Programs\Python\Python38\';
 //  pythonhome:= 'C:\Users\breitsch\AppData\Local\Programs\Python\Python37_32\';
  try
    loadDLL;
    Println('Decimal: '+
          EvalStr('__import__("decimal").Decimal(0.1)'));
          execstr('res=[]');
          execstr('for i in range(50000):'+#10+
                  '  if i == sum(int(x) ** int(x) for x in str(i)):'+#10+
                  '    res.append(i)');
    Println('Munchausen Py: '+EvalStr('res'));
  except
    raiseError;
  finally
    free;
  end;
 end;
end.
```

```
438579088 34578889
Check Count: 43758
Decimal: 0.1000000000000000055511151231257827021181583404541015625
Munchausen Py: [1, 3435]
□□□ mX5 executed: 27/08/2023 17:32:15  Runtime: 0:0:6.428  Memload: 47% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

One of the unsolved problems is to catch an Access-violation instead of crash the app. Its like the code in `uPSRuntime` could not catch cause of halt or exit like the following:

```
procedure TdynamicDll.Quit;
begin
  if not( csDesigning in ComponentState ) then begin
{$IFDEF MSWINDOWS}
   MessageBox( GetActiveWindow, PChar(GetQuitMessage), 'Error',
                                MB_TASKMODAL or MB_ICONSTOP );
   ExitProcess( 1 );
{$ELSE}
   WriteLn(ErrOutput, GetQuitMessage);
   Halt( 1 );
{$ENDIF}
  end;
end;
```

The weird thing was in one of the previous alpha versions (*see below*) the catch of the **AV** was present but in the meantime the app stuck and exits:



```
     //rest:= Resource((Format(URLPIXABAY2,[askstream,imagetype,perpage])));
     //println('@addr:'+objtostr(rest))
     rest:= Resource('https://pixabay.com/api/?key='+PIXABAY_APIKEY+
                                               '&q=rocks&image_type='+
                                               'photo&per_page=40&pretty=true');
     rest.ContentType('application/json');
     rest.header('User-Agent',UAGENT);
     rest.accept('application/json,text/plain; q=0.9, text/html;q=0.8');
     ConnectionType:= hctWinInet;
     //OnafterRequest:= @TRestOnRequestEvent2;
     //OnResponse:= @TRestOnResponseEvent2;
    try
      jo:= TJSON.Create();
      rest.Get()
      //jo.parse(rest.Get());
    writeln('return total hits: '+jo.values['totalhits'].asstring)
      for it:= 1 to perpage-1 do
        result.add(jo.values['hits'].asarray[it].asobject['webformatURL'].asstring);   // }
    finally
      Free;
      jo.Free;
```

```
maXbox5 C:\Program Files\Streaming\maxbox4\examples\1248_munchhausennumbers1_64_2.pas Compiled: 29/08/2023 16:33:12  Mem: 51  Row:   118 --- Col:  1  Sel:  4 S
```

```
EWI_Exc:Exception: Access violation at address 0000000001354A38 in module 'maXbox5_01alpha0.exe'. Read of
address 0000000000000078
debug srlist 0
□□□ mX5 executed: 29/08/2023 16:33:21  Runtime: 0:0:9.41  Memload: 51% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

maXbox5 64-bit ScriptStudio 1248_munchhausennumbers1_64_2.pas
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compile!
Use Cases
12 1248_munchhausennumbers1_64_2.pas

After debugging I realized it is a first chance exception which works as long the debugger is running with break or continue but without debugger the app disappears without forwarding the **AV** on the output like **AV** at address xyz read of address 000.
You can tell the debugger to ignore certain kinds of exceptions. Figure 3 shows **Delphi's** language-exception options. Add an exception class to the list, and all exceptions of that type and of any descendant types will pass through to your program without **Delphi** interfering. You can use **Delphi's** "`advanced breakpoints`" to disable exception handling around a region of code. To begin, set a **breakpoint** on the line of code where you want the **IDE** to ignore exceptions.

Now let's have a last look at a selected test app/script below with individual texts from your own data to translate. We wrote two useful functions. The first one returns text translated with a target language. The second one accepts one sentence as an argument with language detection as a param "auto". Then it will show text in **JSON** or as file.

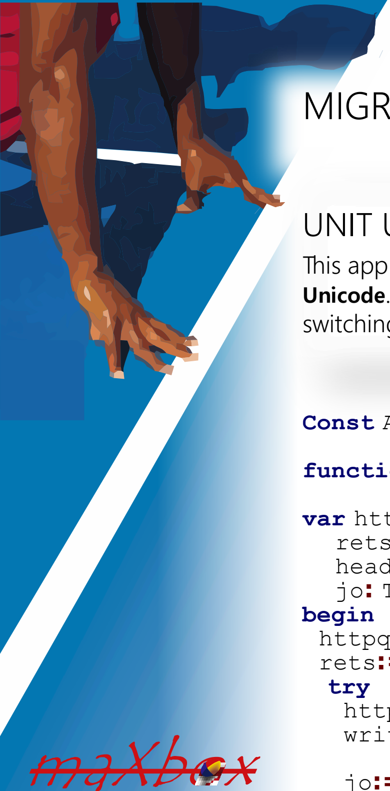
Or you can switch from assembler to Pascal code:

```
{$define GEOMETRY_NO_ASM}

procedure DivMod(dividend : Integer; divisor: Word;
                                   var result, remainder : Word);
{$ifndef GEOMETRY_NO_ASM}
asm
  push  ebx
  mov   ebx, edx
  mov   edx, eax
  shr   edx, 16
  div   bx
  mov   ebx, remainder
  mov   [ecx], ax
  mov   [ebx], dx
  pop   ebx
{$else}
begin
  Result:=Dividend div Divisor;
  Remainder:=Dividend mod Divisor;
{$endif}
end;
```

The error "unit is compiled with a different version of..." is an annoying one. It occurs in a situation like below:

```
    +---------+
    | unit A  |
    +---------+
      |       |
      |       |
      V       |
+---------+   |
| unit B  |   |
+---------+   |
      |       |
      |       |
      V       V
    +---------+
    | unit C  |
    +---------+
```

Both `unit A` and `B` use `unit C` and `unit B` uses `C`. `Unit B` and `C` are compiled and for some reason.
The source of `unit B` is not available.
Now Unit C is changed (*any change will do and is recompiled*).
The `dcu` of `unit C` differs from the `unit C` used by `unit B`,
so `unit B` needs to be recompiled too.
But unfortunately, the source is not available so the compiler gives up.



UNIT UNICODE TESTING

This app allows you to translate or detect text from many different languages and to test **mX5** with **Unicode**. That's why I want this endpoint to be seamlessly integrated into googletrans, with it switching between endpoints if one is facing `4xx/5xx` errors.

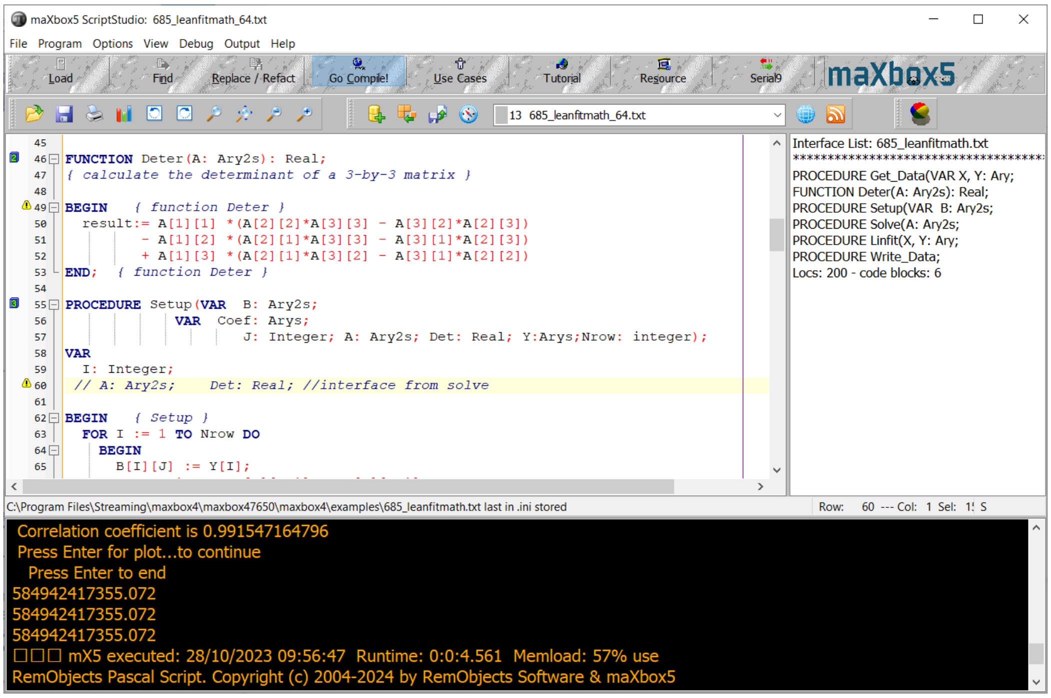
```
Const AURLS = 'https://clients5.google.com/translate_a/t?client=dict-chrome-ex&sl=%s&tl=%s&q=%s';

function Text_to_traslate_API5(AURL, aclient,langorig,langtarget,atext:
                                            string):string;
var httpq: THttpConnectionWinInet;
   rets: TStringStream;
   heads: TStrings; iht: IHttpConnection;
   jo: TJSON; jarr: TJsonArray2;
begin
  httpq:= THttpConnectionWinInet.Create(true);
  rets:= TStringStream.create('');
   try
    httpq.Get(Format(AURLS,[langorig,langtarget,atext]),rets);
    writeln('server: '+Httpq.GetResponseHeader('server'));

    jo:= TJSON.Create();
    jo.parse(rets.datastring)
    jarr:= jo.JsonArray;
    if httpq.getresponsecode=200 Then result:=jarr[0].stringify
      else result:='Failed:'+
          itoa(Httpq.getresponsecode)+Httpq.GetResponseHeader('message');
   except
     writeln('EWI_HTTP: '+ExceptiontoString(exceptiontype,exceptionparam));
   finally
    httpq.free;
    httpq:= Nil;
    rets.Free;
    jo.free;
   end;
end;
```

Google's service, offered free of charge, instantly translates words, phrases, text and web pages between English and over 100 other languages.
That's how we call the function:

```
atext:= 'bonjour mes amis da la ville';
writeln(utf8ToAnsi(Text_to_traslate_API2(AURL,'dict-chrome-
                                       ex','auto','es',atext)));
```

and the result: server: ESF
["Hola mis amigos en la ciudad","fr"]

Google Translate is now a form of **augmented reality** and is adapted for educational purposes.
This application provides users with tools to translate between languages and
they now include an image option; users
take a photograph of a sign, piece of paper,
or other form of written text and receive a translation in the language of their choice.


maXbox5 ScriptStudio: 685_leanfitmath_64.txt
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compile!
Use Cases
Tutorial
Resource
Serial9
maXbox5
13 685_leanfitmath_64.txt

```
FUNCTION Deter(A: Ary2s): Real;
{ calculate the determinant of a 3-by-3 matrix }

BEGIN   { function Deter }
  result:= A[1][1] *(A[2][2]*A[3][3] - A[3][2]*A[2][3])
         - A[1][2] *(A[2][1]*A[3][3] - A[3][1]*A[2][3])
         + A[1][3] *(A[2][1]*A[3][2] - A[3][1]*A[2][2])
END;  { function Deter }

PROCEDURE Setup(VAR  B: Ary2s;
                 VAR  Coef: Arys;
                        J: Integer; A: Ary2s; Det: Real; Y:Arys;Nrow: integer);
VAR
  I: Integer;
 // A: Ary2s;    Det: Real; //interface from solve

BEGIN   { Setup }
  FOR I := 1 TO Nrow DO
    BEGIN
      B[I][J] := Y[I];
```

```
Interface List: 685_leanfitmath.txt
************************************
PROCEDURE Get_Data(VAR X, Y: Ary;
FUNCTION Deter(A: Ary2s): Real;
PROCEDURE Setup(VAR  B: Ary2s;
PROCEDURE Solve(A: Ary2s;
PROCEDURE Linfit(X, Y: Ary;
PROCEDURE Write_Data;
Locs: 200 - code blocks: 6
```

```
C:\Program Files\Streaming\maxbox4\maxbox47650\maxbox4\examples\685_leanfitmath.txt last in .ini stored     Row:   60 --- Col:  1  Sel:  1! S
```

```
 Correlation coefficient is 0.991547164796
 Press Enter for plot...to continue
  Press Enter to end
584942417355.072
584942417355.072
584942417355.072
□□□ mX5 executed: 28/10/2023 09:56:47  Runtime: 0:0:4.561  Memload: 57% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

This visual technique above used to help with the understanding about what individual texts represent is called semantic analysis. About the topic: https://en.wikipedia.org/wiki/Semantic_analysis_(linguistics)

I found another endpoint to test with unicode within the source code of one of the google translate extensions on VSCode too.

```
"https://translate.googleapis.com/translate_a/single?client=gtx&dt=t + params"
// where the params are:
{
  "sl": source language,
  "tl": destination language,
  "q": the text to translate
}
```

The results looks something like this:

```
[[["こんにちは、今日はお元気ですか？","Hello, how are you today?",null,null,3,null,null,[[]
]
,[[["9588ca5d94759e1e85ee26c1b641b1e3","kgmt_en_ja_2020q3.md"]
]]
]]
,null,"en",null,null,null,null,[]
]
```

for the query: https://translate.googleapis.com/translate_a/single?client=gtx&dt=t&sl=en&tl=ja&q=Hello, how are you today?
And something like this:

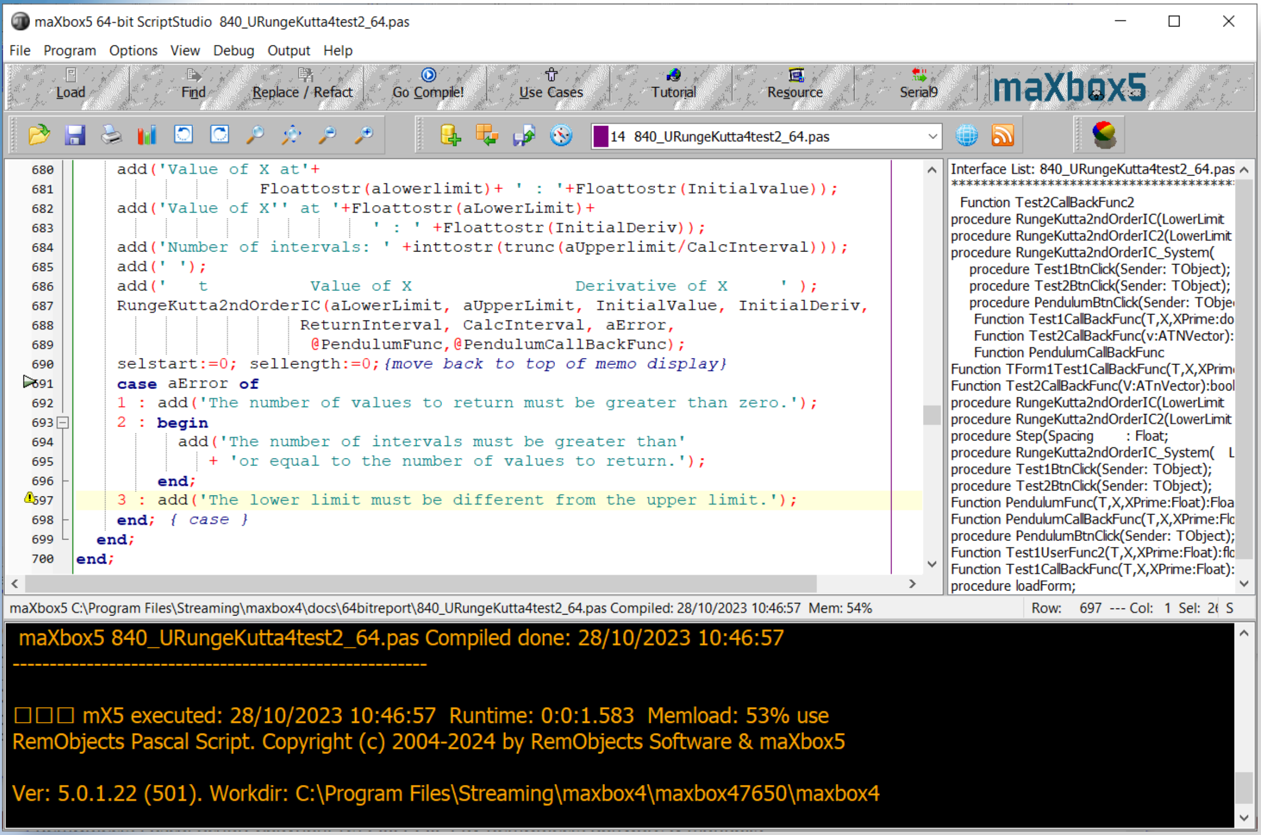
```
[[["Bonjour","Hello",null,null,1]]
,null,"en",null,null,null,null,[]
]
```

String unicode `(\uxxx)` encoding and decoding.

After some testing with request headers and **F12** tools – Inspect (see below),
I found the solution for the garbled text it can be.
Simply set the **User-Agent header** to the one that **Google Chrome** uses.

EXAMPLE:

import requests
```
word = 'اذا لعفت اذا لما'
url = "https://clients5.google.com/translate_a/
t?client=dict-chrome-ex&sl=auto&tl=en&q=" + word
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/
537.36'}

try:
    request_result = requests.get(url, headers=headers).json()
    print(request_result)
    print('[In English]: ' + request_result['alternative_
translations'][0]['alternative'][0]['word_postproc'])
    print('[Language Dectected]: ' + request_result['src'])
except:
     pass
```


maXbox5 64-bit ScriptStudio 840_URungeKutta4test2_64.pas

File Program Options View Debug Output Help

Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9 maXbox5

14 840_URungeKutta4test2_64.pas

```
      add('Value of X at'+
                  Floattostr(alowerlimit)+ ' : '+Floattostr(Initialvalue));
      add('Value of X'' at '+Floattostr(aLowerLimit)+
                              ' : ' +Floattostr(InitialDeriv));
      add('Number of intervals: ' +inttostr(trunc(aUpperlimit/CalcInterval)));
      add(' ');
      add('   t           Value of X                Derivative of X     ' );
      RungeKutta2ndOrderIC(aLowerLimit, aUpperLimit, InitialValue, InitialDeriv,
                           ReturnInterval, CalcInterval, aError,
                            @PendulumFunc,@PendulumCallBackFunc);
      selstart:=0; sellength:=0;{move back to top of memo display}
      case aError of
      1 : add('The number of values to return must be greater than zero.');
      2 : begin
            add('The number of intervals must be greater than'
               + 'or equal to the number of values to return.');
          end;
      3 : add('The lower limit must be different from the upper limit.');
      end; { case }
    end;
end;
```

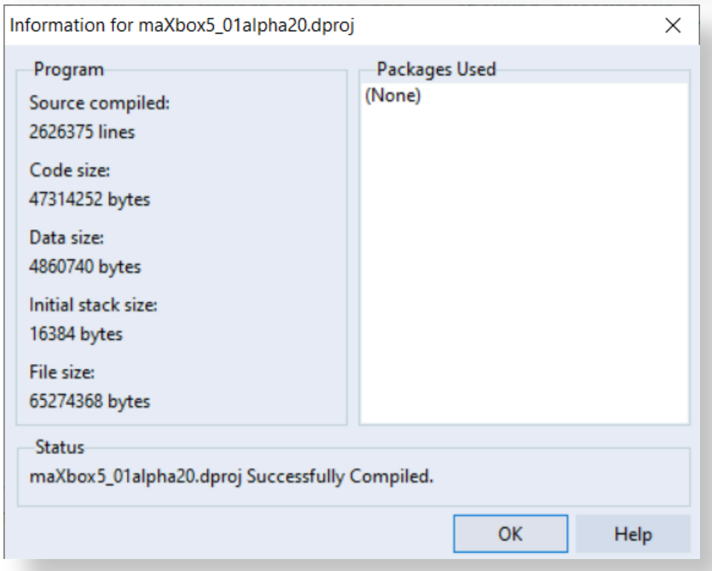
```
Interface List: 840_URungeKutta4test2_64.pas
****************************************
  Function Test2CallBackFunc2
procedure RungeKutta2ndOrderIC(LowerLimit
procedure RungeKutta2ndOrderIC2(LowerLimit
procedure RungeKutta2ndOrderIC_System(
    procedure Test1BtnClick(Sender: TObject);
    procedure Test2BtnClick(Sender: TObject);
    procedure PendulumBtnClick(Sender: TObje
     Function Test1CallBackFunc(T,X,XPrime:do
     Function Test2CallBackFunc(v:ATNVector):
     Function PendulumCallBackFunc
Function TForm1Test1CallBackFunc(T,X,XPrim
Function Test2CallBackFunc(V:ATnVector):bool
procedure RungeKutta2ndOrderIC(LowerLimit
procedure RungeKutta2ndOrderIC2(LowerLimit
procedure Step(Spacing      : Float;
procedure RungeKutta2ndOrderIC_System(   L
procedure Test1BtnClick(Sender: TObject);
procedure Test2BtnClick(Sender: TObject);
Function PendulumFunc(T,X,XPrime:Float):Floa
Function PendulumCallBackFunc(T,X,XPrime:Flo
procedure PendulumBtnClick(Sender: TObject);
Function Test1UserFunc2(T,X,XPrime:Float):flo
Function Test1CallBackFunc(T,X,XPrime:Float):
procedure loadForm;
```

maXbox5 C:\Program Files\Streaming\maxbox4\docs\64bitreport\840_URungeKutta4test2_64.pas Compiled: 28/10/2023 10:46:57 Mem: 54% Row: 697 --- Col: 1 Sel: 2(S

```
 maXbox5 840_URungeKutta4test2_64.pas Compiled done: 28/10/2023 10:46:57
-------------------------------------------------------

□□□ mX5 executed: 28/10/2023 10:46:57  Runtime: 0:0:1.583  Memload: 53% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5

Ver: 5.0.1.22 (501). Workdir: C:\Program Files\Streaming\maxbox4\maxbox47650\maxbox4
```

CONCLUSION:

Of course the 64-bit-box is not finished yet. The current version 5.0.1.22 is an early beta Version.
As a next step method pointers (*func pointers*) and `TEncoding` are on the list.
Also on discovering a `function` marked with the overload directive, it prompts for a new `function` name, and then generates wrapper code that maps the new method name to the original version, but in a redirection we get an **AV**.
In general, there is unlikely to be much benefit beyond an 32-bit, possibly, a small speed increase, more memory and more registers.
Don't rely on 64-bit to speed up a slow application though:
you will still need to make algorithmic changes for large speed boosts.
64-bit isn't a magic bullet. Other than that, you only need to change if you've already encountered one of the limits imposed by 32-bit or you want to develop plugins for 64-bit app or just be compatible with a 64-bit operation system.


Information for maXbox5_01alpha20.dproj

Program
Source compiled:
2626375 lines
Code size:
47314252 bytes
Data size:
4860740 bytes
Initial stack size:
16384 bytes
File size:
65274368 bytes

Packages Used
(None)

Status
maXbox5_01alpha20.dproj Successfully Compiled.

REFERENCES:
Compiled Project:
`https://github.com/maxkleiner/maXbox4/releases/download/V4.2.4.80/maxbox5.zip`

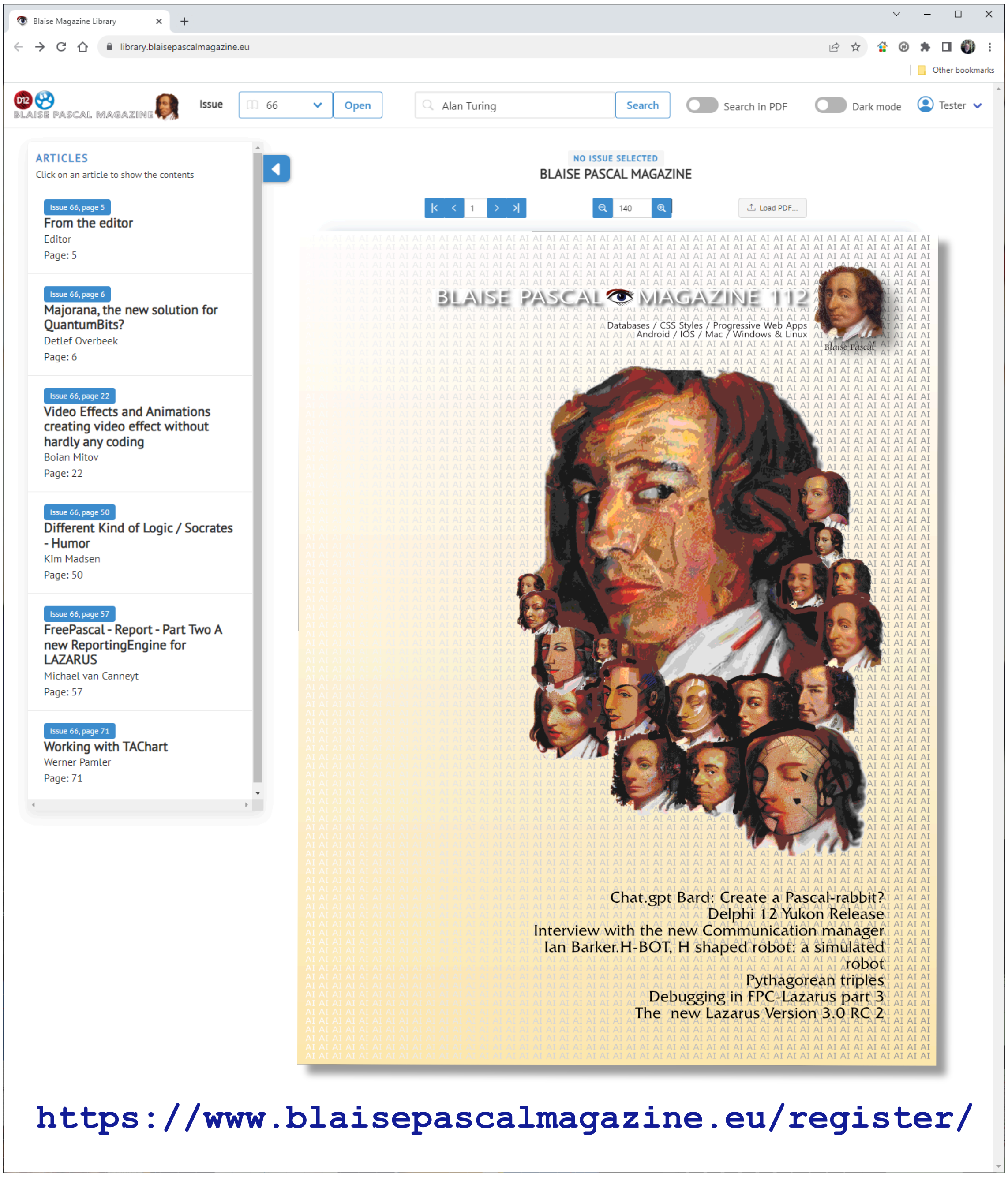
Preparation:
`https://stackoverflow.com/questions/4051603/how-should-i-prepare-my-32-bit-delphi-programs-for-an-eventual-64-bit-compiler`

Doc and Tool: `https://maxbox4.wordpress.com`

THE NEW SUBSCRIPTION MODEL BLAISE PASCAL MAGAZINE

1. SUBSCRIPTION: PER YEAR - no changes: issues starting at the latest issue available +1 year / code included + issues downloadable € 70,00 or without Vat 64,22. For all countries INCLUDING FREE INTERNET LIBRARY FOR ALL MAGAZINES

2 LIB-STICK USB-CARD: all issues / code included. same interface as the internet library. € 120,00 FOR ALL COUNTRIES including 1 year subscription



ARTICLES
Click on an article to show the contents

Issue 66, page 5
From the editor
Editor
Page: 5

Issue 66, page 6
Majorana, the new solution for QuantumBits?
Detlef Overbeek
Page: 6

Issue 66, page 22
Video Effects and Animations creating video effect without hardly any coding
Boian Mitov
Page: 22

Issue 66, page 50
Different Kind of Logic / Socrates - Humor
Kim Madsen
Page: 50

Issue 66, page 57
FreePascal - Report - Part Two A new ReportingEngine for LAZARUS
Michael van Canneyt
Page: 57

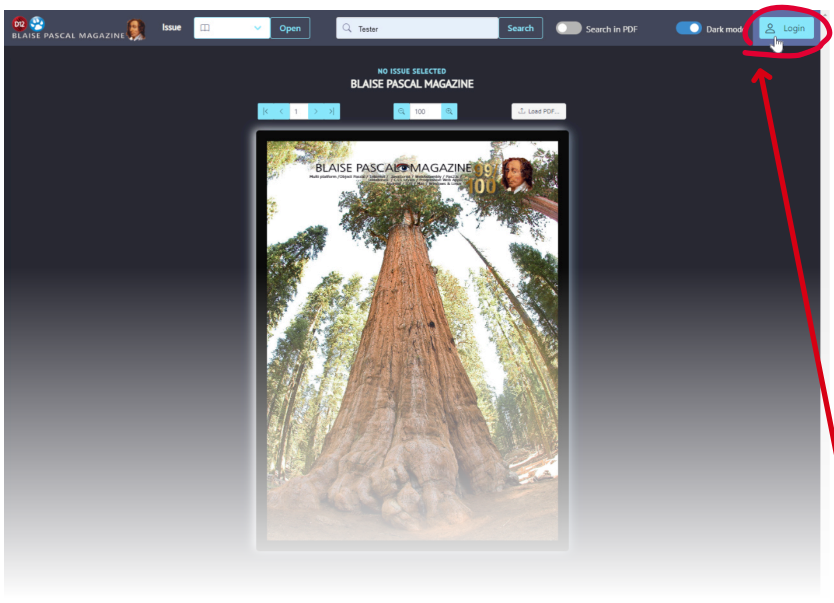
Issue 66, page 71
Working with TAChart
Werner Pamler
Page: 71

Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

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

USE WHERE EVER THE INTERNET IS

THE NEW FREE EXTRA INTERNET PDF LIBRARY SUBSCRIPTION

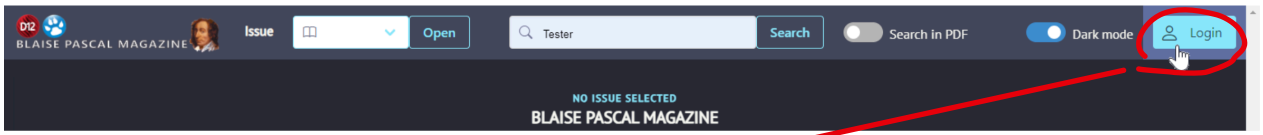


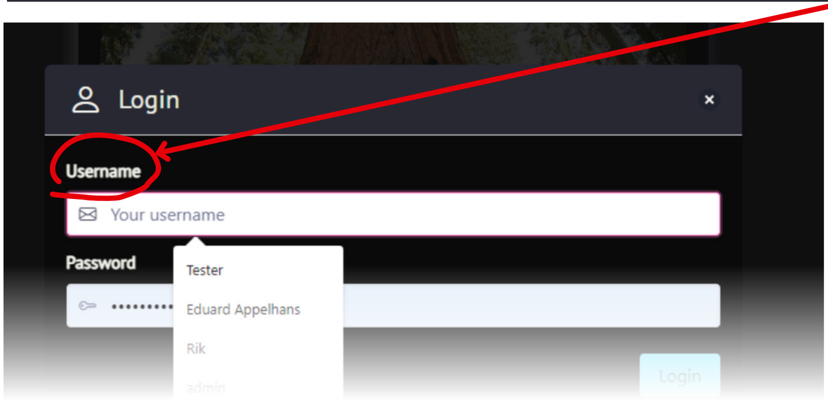
Is a free addition to your normal subscription. You can view and search over all issues, as well in all issues.
This article treys to help you to see what's possible. It is available through the internet and can open any pdf.
If you have a subscription (download or printed) you can use it for free for the period of one year., the same period of your subscription.

You will automatically receive a login and a password.

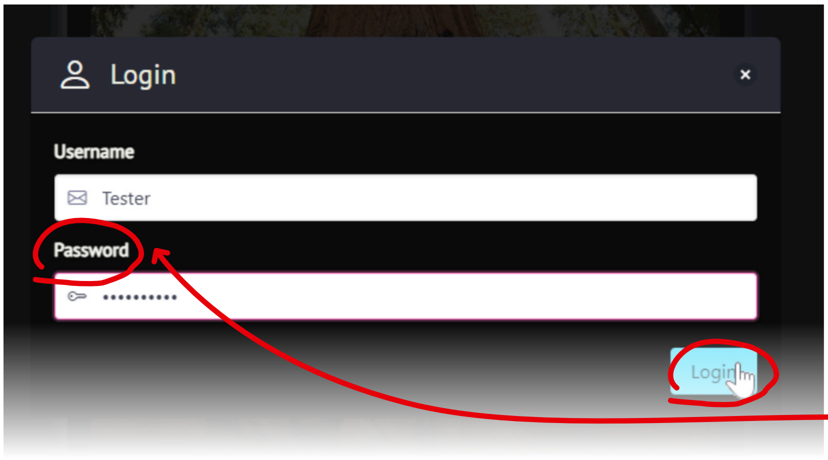
If you have additional request or want to suggest extra or improvements let me know.

❶ To start:
at the right top corner you can login.



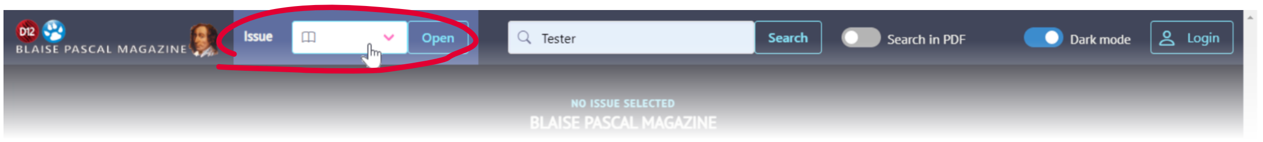


❷ Insert your username. You received that with the correspondence about the subscription.

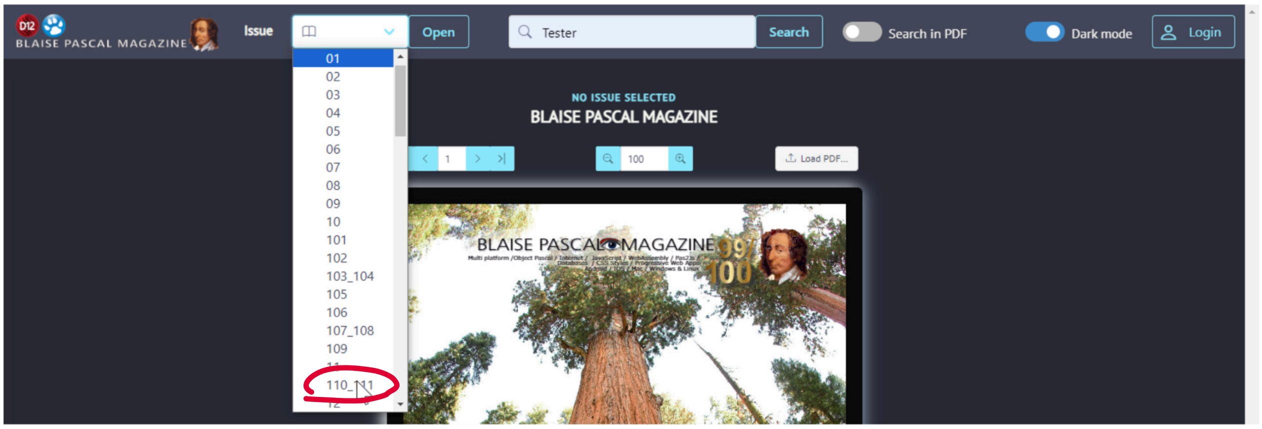


❸ Enter the password. Click on Login

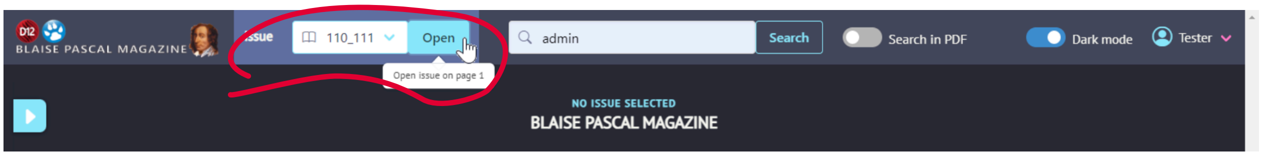
INTERNET PDF LIBRARY SUBSCRIPTION



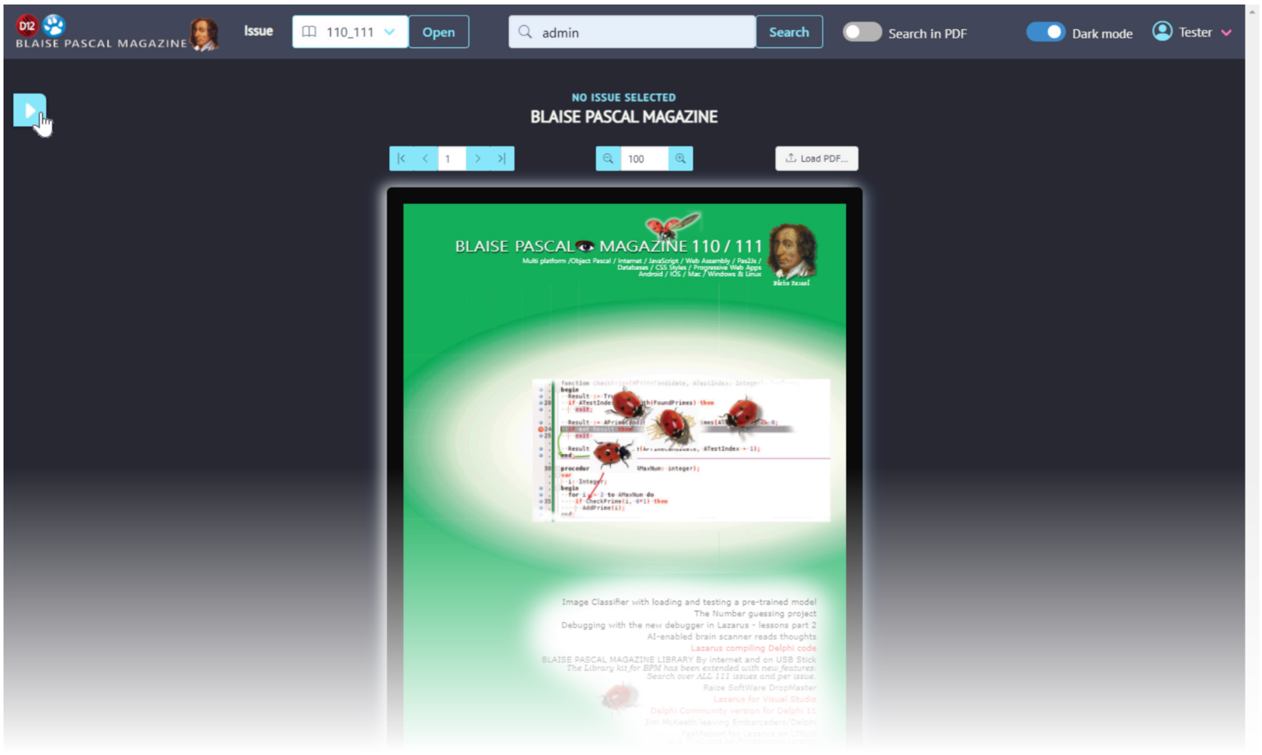
❹ To start opening an issue: click on the dropdown list



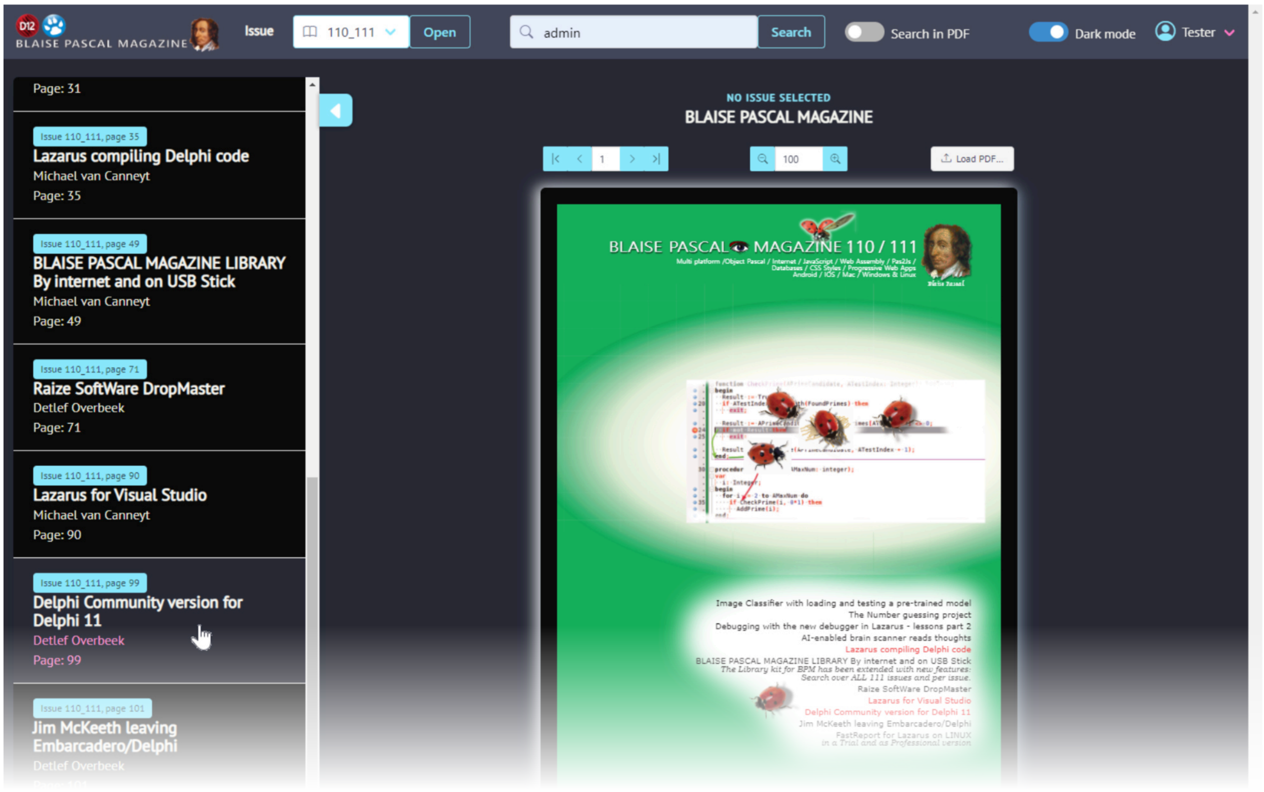
❺ Choose the number of the issue you want to view



❻ After that click on open.It needs some time and will open the first page of the issue



❼ The first page becomes available.



Page: 31

Issue 110_111, page 35
Lazarus compiling Delphi code
Michael van Canneyt
Page: 35

Issue 110_111, page 49
BLAISE PASCAL MAGAZINE LIBRARY By internet and on USB Stick
Michael van Canneyt
Page: 49

Issue 110_111, page 71
Raize SoftWare DropMaster
Detlef Overbeek
Page: 71

Issue 110_111, page 90
Lazarus for Visual Studio
Michael van Canneyt
Page: 90

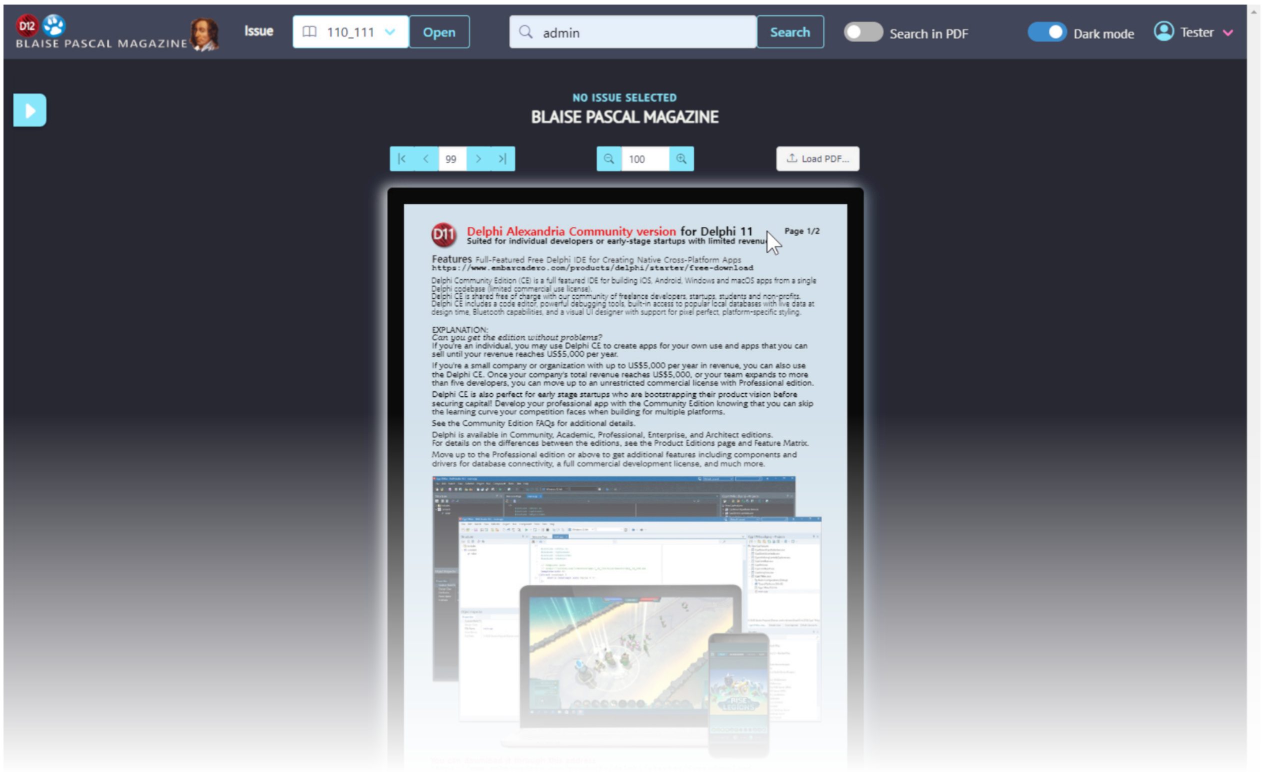
Issue 110_111, page 99
Delphi Community version for Delphi 11
Detlef Overbeek
Page: 99

Issue 110_111, page 101
Jim McKeeth leaving Embarcadero/Delphi
Detlef Overbeek

BLAISE PASCAL MAGAZINE 110 / 111
Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web Apps Android / IOS / Mac / Windows & Linux

Image Classifier with loading and testing a pre-trained model
The Number guessing project
Debugging with the new debugger in Lazarus - lessons part 2
AI-enabled brain scanner reads thoughts
Lazarus compiling Delphi code
BLAISE PASCAL MAGAZINE LIBRARY By internet and on USB Stick
The Library kit for BPM has been extended with new features: Search over ALL 111 issues and per issue.
Raize SoftWare DropMaster
Lazarus for Visual Studio
Delphi Community version for Delphi 11
Jim McKeeth leaving Embarcadero/Delphi
FastReport for Lazarus on LINUX
in a Trial and as Professional version

❼ The list at the left shows the articles of that issue



Delphi Alexandria Community version for Delphi 11 Page 1/2
Suited for individual developers or early-stage startups with limited revenu

Features Full-Featured Free Delphi IDE for Creating Native Cross-Platform Apps
`https://www.embarcadero.com/products/delphi/starter/free-download`

Delphi Community Edition (CE) is a full featured IDE for building iOS, Android, Windows and macOS apps from a single Delphi codebase (limited commercial use license).
Delphi CE is shared free of charge with our community of freelance developers, startups, students and non-profits.
Delphi CE includes a code editor, powerful debugging tools, built-in access to popular local databases with live data at design time, Bluetooth capabilities, and a visual UI designer with support for pixel perfect, platform-specific styling.

EXPLANATION:
Can you get the edition without problems?
If you're an individual, you may use Delphi CE to create apps for your own use and apps that you can sell until your revenue reaches US$5,000 per year.

If you're a small company or organization with up to US$5,000 per year in revenue, you can also use the Delphi CE. Once your company's total revenue reaches US$5,000, or your team expands to more than five developers, you can move up to an unrestricted commercial license with Professional edition.

Delphi CE is also perfect for early stage startups who are bootstrapping their product vision before securing capital! Develop your professional app with the Community Edition knowing that you can skip the learning curve your competition faces when building for multiple platforms.

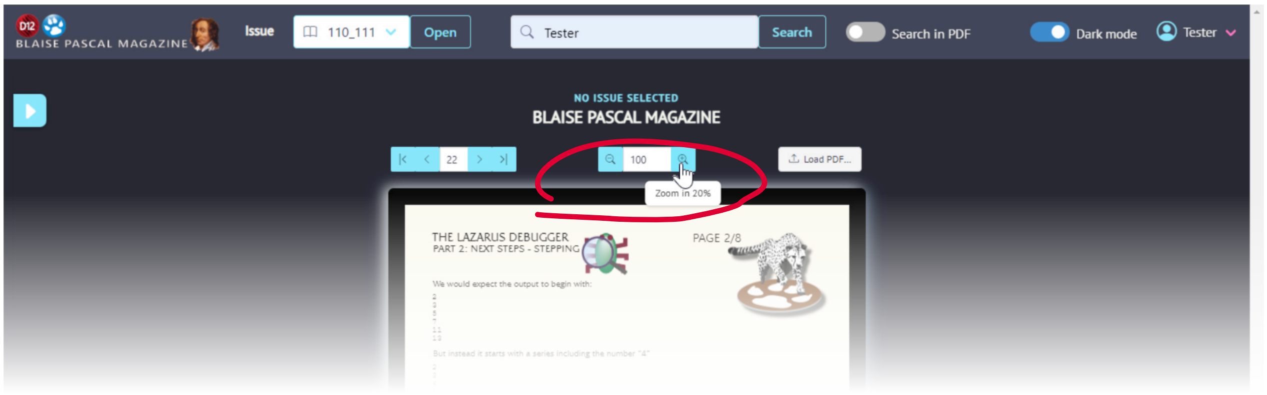
See the Community Edition FAQs for additional details.

Delphi is available in Community, Academic, Professional, Enterprise, and Architect editions.
For details on the differences between the editions, see the Product Editions page and Feature Matrix.

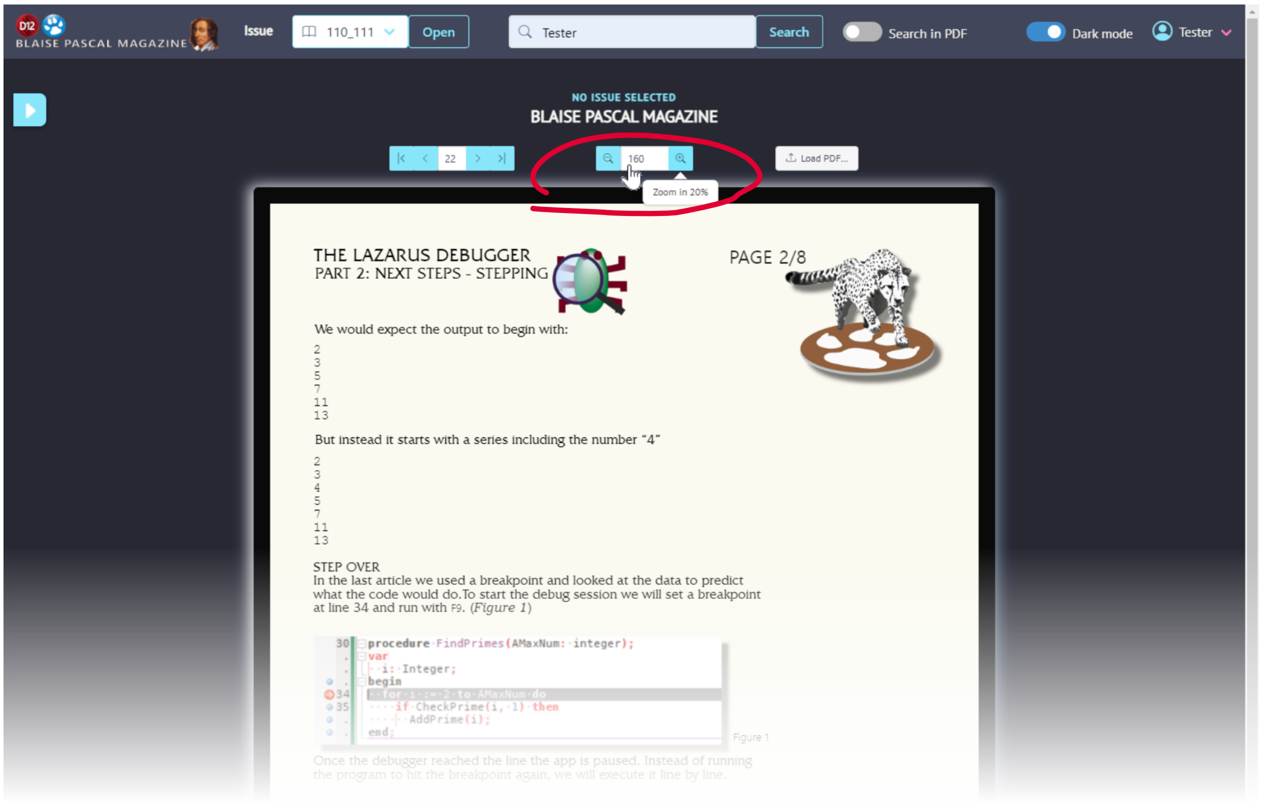
Move up to the Professional edition or above to get additional features including components and drivers for database connectivity, a full commercial development license, and much more.

❽ The article chosen starts up

INTERNET PDF LIBRARY SUBSCRIPTION



❾ You can enlarge the size of the pdf it self



THE LAZARUS DEBUGGER
PART 2: NEXT STEPS - STEPPING

PAGE 2/8

We would expect the output to begin with:

```
2
3
5
7
11
13
```

But instead it starts with a series including the number “4”

```
2
3
4
5
7
11
13
```

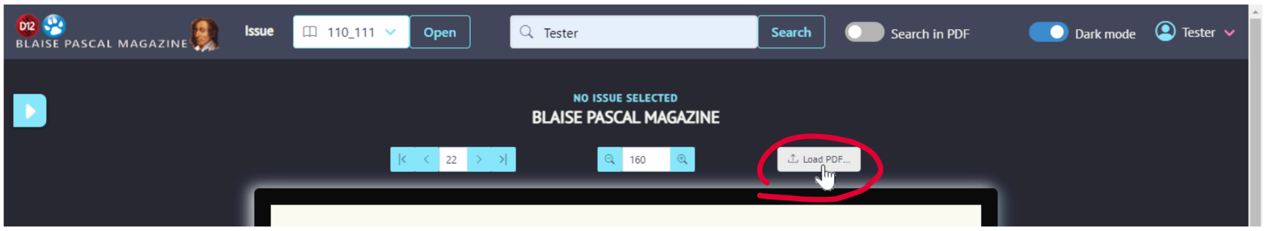
STEP OVER
In the last article we used a breakpoint and looked at the data to predict what the code would do.To start the debug session we will set a breakpoint at line 34 and run with F9. (*Figure 1*)

```
30 procedure FindPrimes(AMaxNum: integer);
 .  var
 .    i: Integer;
 .  begin
34   for i := 2 to AMaxNum do
35     if CheckPrime(i, 1) then
 .       AddPrime(i);
 .  end;
```

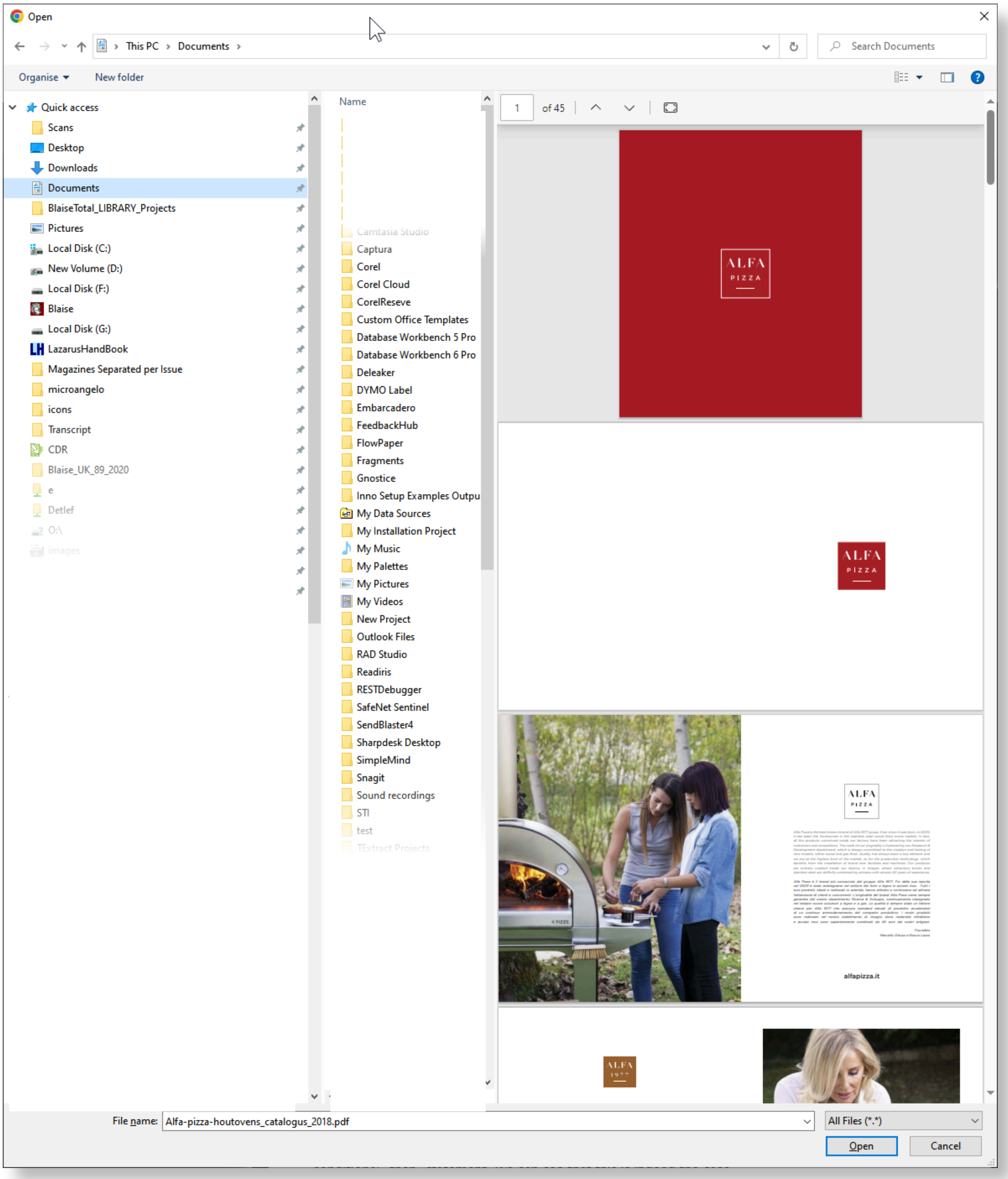
Figure 1

Once the debugger reached the line the app is paused. Instead of running the program to hit the breakpoint again, we will execute it line by line.

❿ It is easier to see the details....



⓫ You can also load any kind of PDF, from your Hard disk, your USB stick or any other source...



Open

This PC › Documents ›

Search Documents

Organise New folder

Quick access
- Scans
- Desktop
- Downloads
- Documents
- BlaiseTotal_LIBRARY_Projects
- Pictures
- Local Disk (C:)
- New Volume (D:)
- Local Disk (F:)
- Blaise
- Local Disk (G:)
- LazarusHandBook
- Magazines Separated per Issue
- microangelo
- icons
- Transcript
- CDR
- Blaise_UK_89_2020
- e
- Detlef
- O:\
- images

Name
- Camtasia Studio
- Captura
- Corel
- Corel Cloud
- CorelReseve
- Custom Office Templates
- Database Workbench 5 Pro
- Database Workbench 6 Pro
- Deleaker
- DYMO Label
- Embarcadero
- FeedbackHub
- FlowPaper
- Fragments
- Gnostice
- Inno Setup Examples Outpu
- My Data Sources
- My Installation Project
- My Music
- My Palettes
- My Pictures
- My Videos
- New Project
- Outlook Files
- RAD Studio
- Readiris
- RESTDebugger
- SafeNet Sentinel
- SendBlaster4
- Sharpdesk Desktop
- SimpleMind
- Snagit
- Sound recordings
- STI
- test
- TExtract Projects

1 of 45

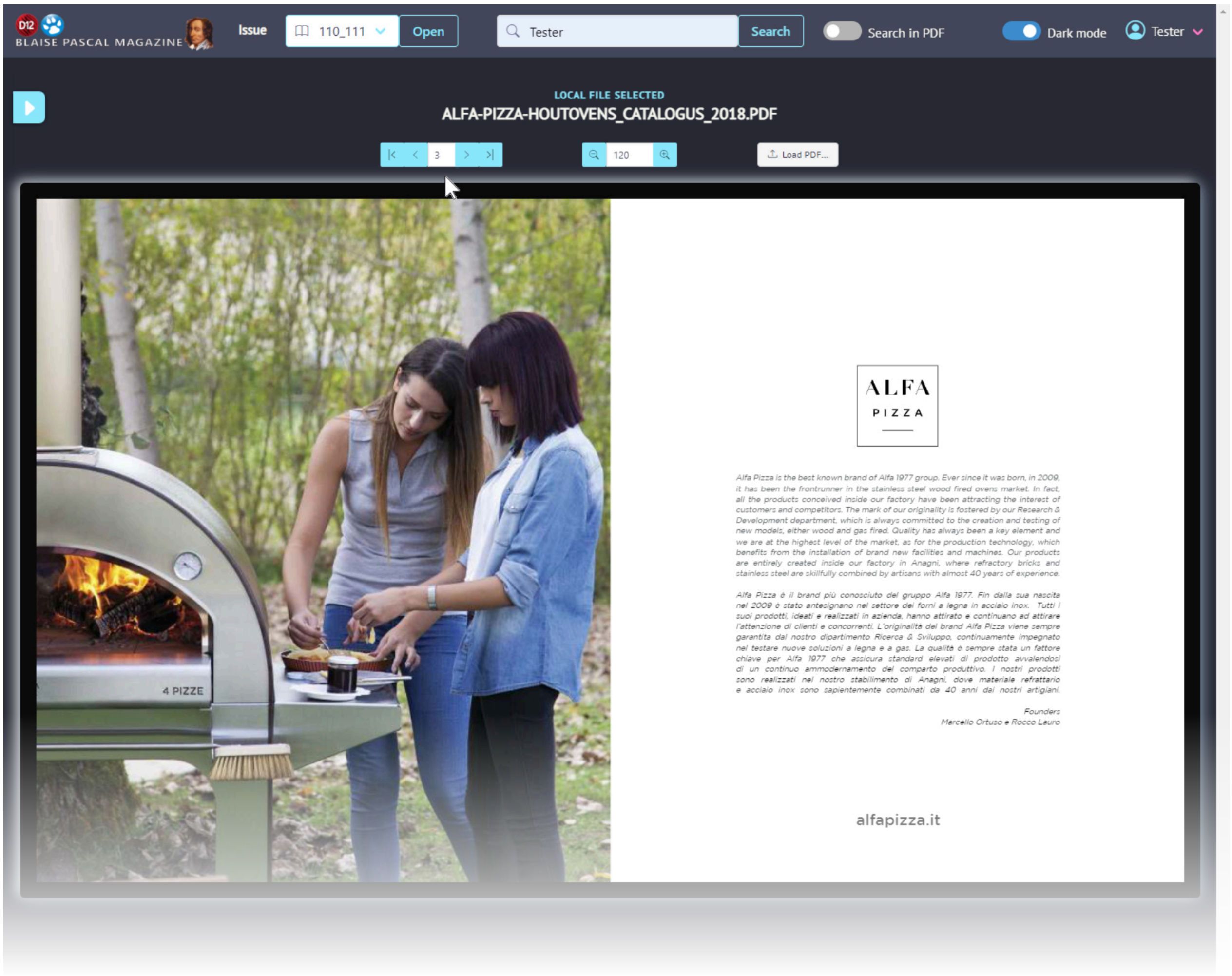
File name: Alfa-pizza-houtovens_catalogus_2018.pdf

All Files (*.*)

Open Cancel

⓬ You can also load any kind of PDF, from your Hard disk, your USB stick or any other source...

THE NEW FREE EXTRA INTERNET PDF LIBRARY SUBSCRIPTION



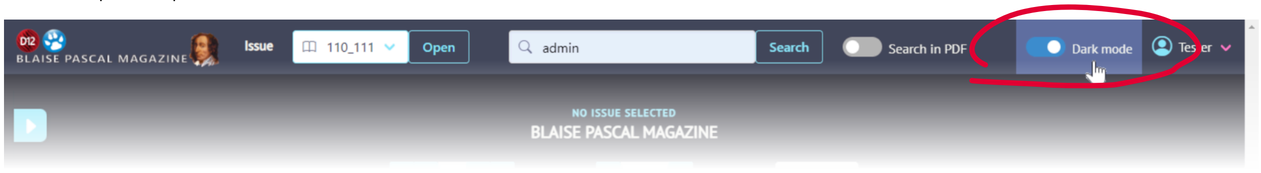
Alfa Pizza is the best known brand of Alfa 1977 group. Ever since it was born, in 2009, it has been the frontrunner in the stainless steel wood fired ovens market. In fact, all the products conceived inside our factory have been attracting the interest of customers and competitors. The mark of our originality is fostered by our Research & Development department, which is always committed to the creation and testing of new models, either wood and gas fired. Quality has always been a key element and we are at the highest level of the market, as for the production technology, which benefits from the installation of brand new facilities and machines. Our products are entirely created inside our factory in Anagni, where refractory bricks and stainless steel are skillfully combined by artisans with almost 40 years of experience.

Alfa Pizza è il brand più conosciuto del gruppo Alfa 1977. Fin dalla sua nascita nel 2009 è stato antesignano nel settore dei forni a legna in acciaio inox. Tutti i suoi prodotti, ideati e realizzati in azienda, hanno attirato e continuano ad attirare l'attenzione di clienti e concorrenti. L'originalità del brand Alfa Pizza viene sempre garantita dal nostro dipartimento Ricerca & Sviluppo, continuamente impegnato nel testare nuove soluzioni a legna e a gas. La qualità è sempre stata un fattore chiave per Alfa 1977 che assicura standard elevati di prodotto avvalendosi di un continuo ammodernamento del comparto produttivo. I nostri prodotti sono realizzati nel nostro stabilimento di Anagni, dove materiale refrattario e acciaio inox sono sapientemente combinati da 40 anni dai nostri artigiani.

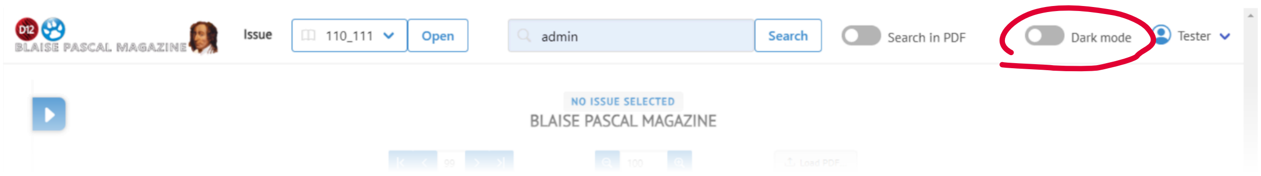
Founders
Marcello Ortuso e Rocco Lauro

alfapizza.it

⓭ The opened pdf file

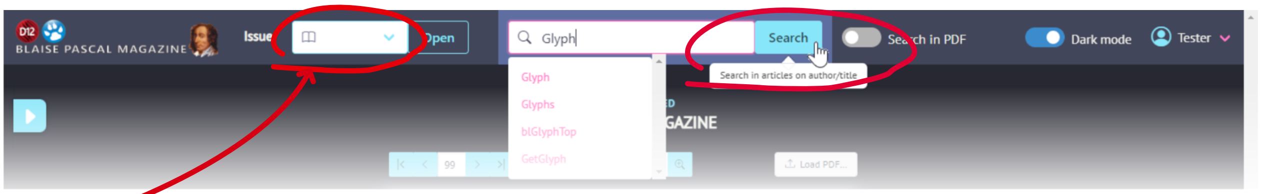


⓮ Dark mode if you prefer

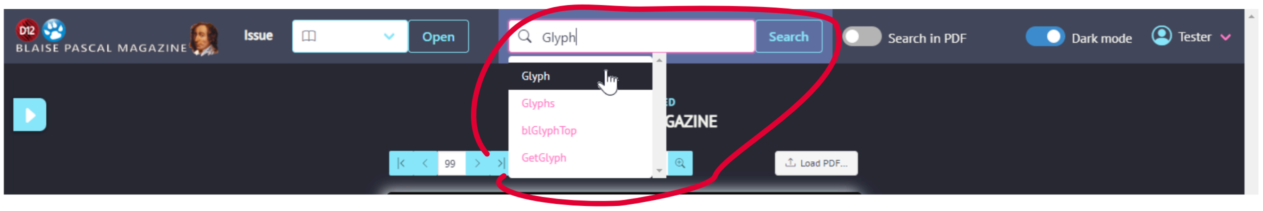


⓯ Light is of course possible

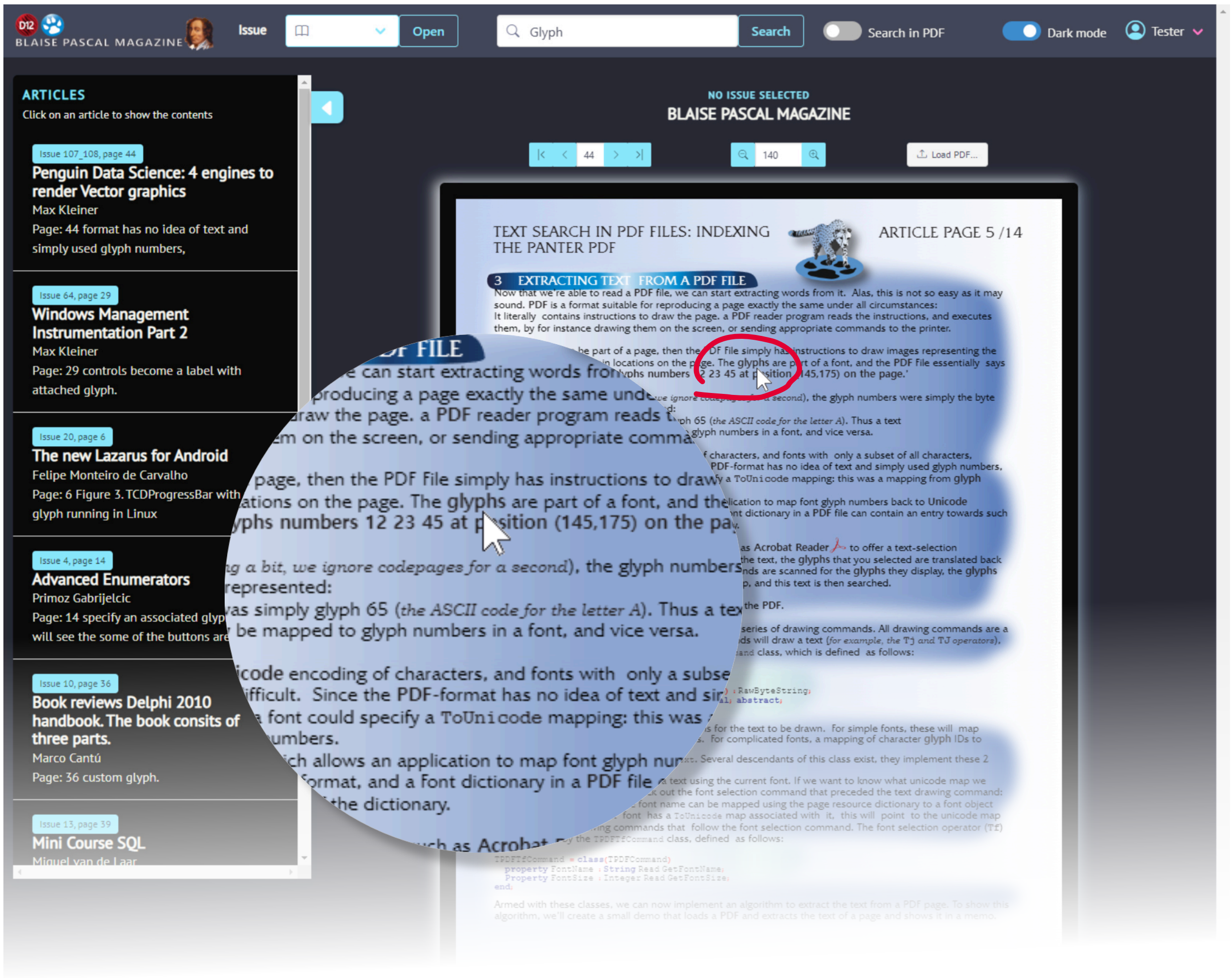
THE NEW FREE EXTRA INTERNET PDF LIBRARY SUBSCRIPTION



⑮ This list needs to be empty if you want to search for text elements over **ALL** issues.
This must be done before you can enter the text that you want to look for



⑯ After that you can enter the word or text



ARTICLES
Click on an article to show the contents

Issue 107_108, page 44
Penguin Data Science: 4 engines to render Vector graphics
Max Kleiner
Page: 44 format has no idea of text and simply used glyph numbers,

Issue 64, page 29
Windows Management Instrumentation Part 2
Max Kleiner
Page: 29 controls become a label with attached glyph.

Issue 20, page 6
The new Lazarus for Android
Felipe Monteiro de Carvalho
Page: 6 Figure 3. TCDProgressBar with glyph running in Linux

Issue 4, page 14
Advanced Enumerators
Primoz Gabrijelcic
Page: 14 specify an associated glyph will see the some of the buttons are

Issue 10, page 36
Book reviews Delphi 2010 handbook. The book consits of three parts.
Marco Cantú
Page: 36 custom glyph.

Issue 13, page 39
Mini Course SQL
Miguel van de Laar

TEXT SEARCH IN PDF FILES: INDEXING THE PANTER PDF — ARTICLE PAGE 5 /14

3 EXTRACTING TEXT FROM A PDF FILE

Now that we're able to read a PDF file, we can start extracting words from it. Alas, this is not so easy as it may sound. PDF is a format suitable for reproducing a page exactly the same under all circumstances:
It literally contains instructions to draw the page. a PDF reader program reads the instructions, and executes them, by for instance drawing them on the screen, or sending appropriate commands to the printer.

page, then the PDF File simply has instructions to draw images representing the locations on the page. The glyphs are part of a font, and the PDF File essentially says glyphs numbers 12 23 45 at position (145,175) on the page.'

g a bit, we ignore codepages for a second), the glyph numbers were simply the byte represented:
was simply glyph 65 (*the ASCII code for the letter A*). Thus a text be mapped to glyph numbers in a font, and vice versa.

icode encoding of characters, and fonts with only a subset of all characters, ifficult. Since the PDF-format has no idea of text and simply used glyph numbers, a font could specify a ToUnicode mapping: this was a mapping from glyph umbers.
ich allows an application to map font glyph numbers back to Unicode format, and a font dictionary in a PDF file can contain an entry towards such the dictionary.

uch as Acrobat Reader to offer a text-selection the text, the glyphs that you selected are translated back nds are scanned for the glyphs they display, the glyphs p, and this text is then searched.

the PDF.

series of drawing commands. All drawing commands are a ds will draw a text (*for example, the* Tj *and* TJ *operators*), and class, which is defined as follows:

```
) :RawByteString;
al; abstract;
```

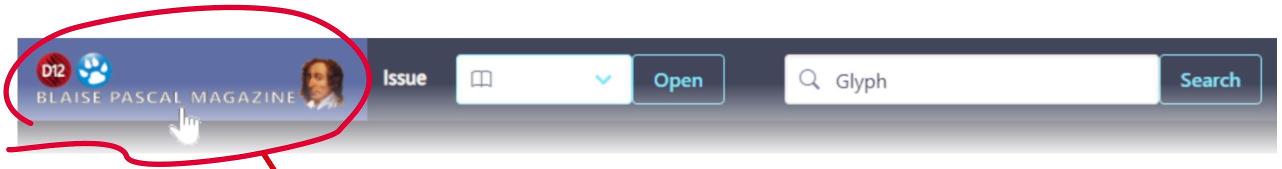
s for the text to be drawn. For simple fonts, these will map s. For complicated fonts, a mapping of character glyph IDs to
xt. Several descendants of this class exist, they implement these 2
a text using the current font. If we want to know what unicode map we k out the font selection command that preceded the text drawing command: font name can be mapped using the page resource dictionary to a font object font has a ToUnicode map associated with it, this will point to the unicode map ing commands that follow the font selection command. The font selection operator (Tf) y the TPDFTfCommand class, defined as follows:

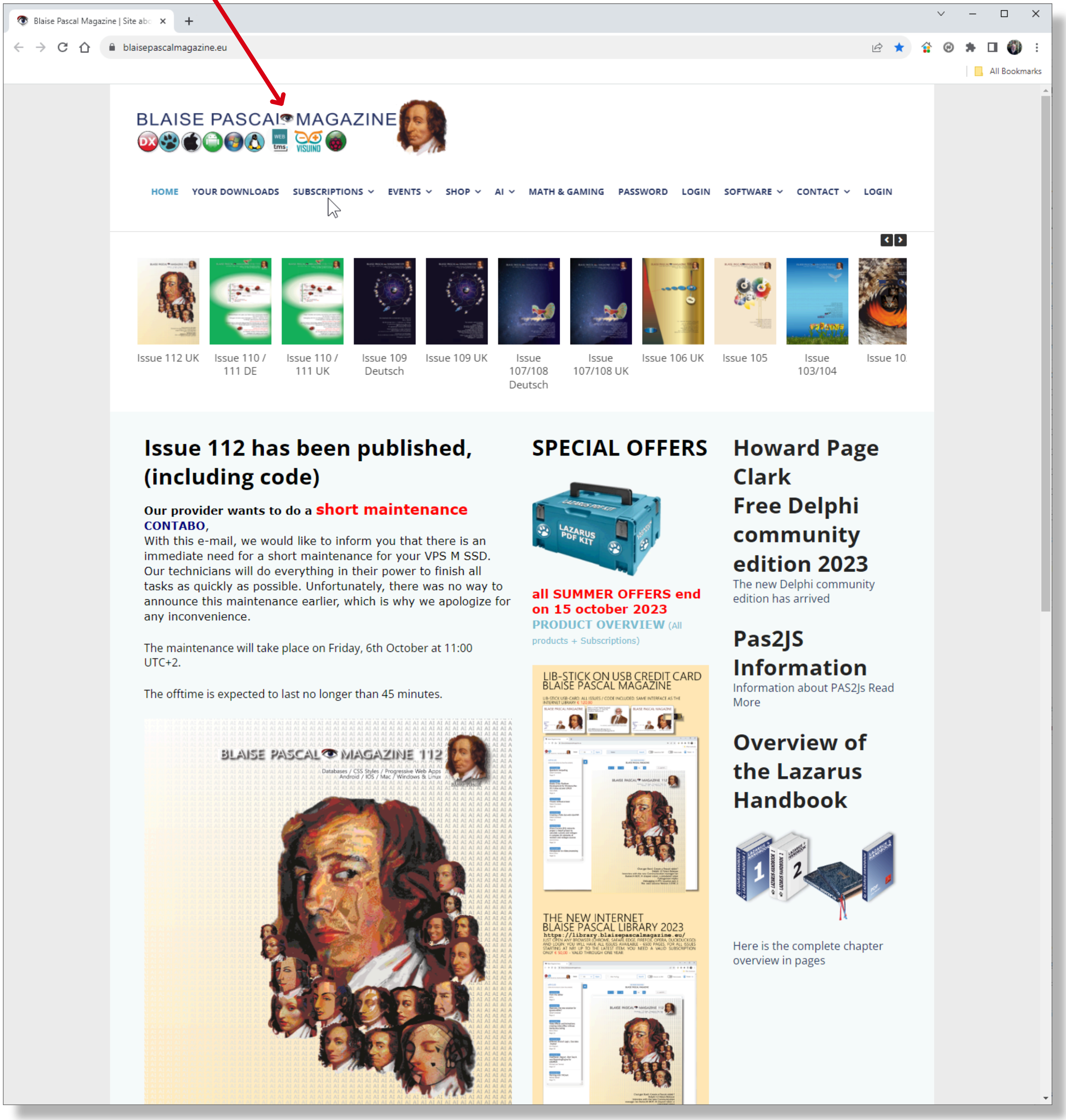
```
TPDFTfCommand = class(TPDFCommand)
  property FontName : String Read GetFontName;
  Property FontSize : Integer Read GetFontSize;
end;
```

Armed with these classes, we can now implement an algorithm to extract the text from a PDF page. To show this algorithm, we'll create a small demo that loads a PDF and extracts the text of a page and shows it in a memo.

⑰ Here is an enlargement of the word in the text that was chosen from the list at the left.
The Article comes up directly

THE NEW FREE EXTRA INTERNET PDF LIBRARY SUBSCRIPTION





Issue 112 has been published, (including code)

Our provider wants to do a short maintenance CONTABO,
With this e-mail, we would like to inform you that there is an immediate need for a short maintenance for your VPS M SSD. Our technicians will do everything in their power to finish all tasks as quickly as possible. Unfortunately, there was no way to announce this maintenance earlier, which is why we apologize for any inconvenience.

The maintenance will take place on Friday, 6th October at 11:00 UTC+2.

The offtime is expected to last no longer than 45 minutes.

SPECIAL OFFERS

all SUMMER OFFERS end on 15 october 2023
PRODUCT OVERVIEW (All products + Subscriptions)

LIB-STICK ON USB CREDIT CARD BLAISE PASCAL MAGAZINE

THE NEW INTERNET BLAISE PASCAL LIBRARY 2023
https://library.blaisepascalmagazine.eu/

Howard Page Clark Free Delphi community edition 2023
The new Delphi community edition has arrived

Pas2JS Information
Information about PAS2Js Read More

Overview of the Lazarus Handbook

Here is the complete chapter overview in pages

BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
LAZARUS PDF KIT

❽ If you click on the logo you will go straight to the website of Blaise Pascal Magazine

BLAISE PASCAL MAGAZINE 112

Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Price: € 75,00
LAZARUS HANDBOOK
POCKET + PDF AND
SUBSCRIPTION
ex Vat and Shipping

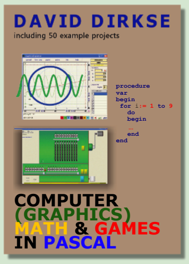
Cha bbit?
lease
Interview with the new C arker.
H-BOT ed robot
arus part 3
rsion 3.0 RC 2

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

COUNTING BINARY PUZZLE SOLUTIONS

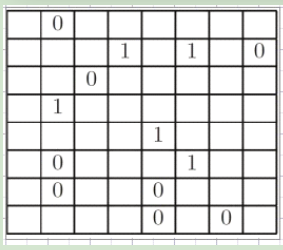
BY DAVID DIRKSE

STARTER EXPERT

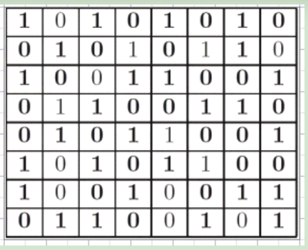


INTRODUCTION

A Visitor of my website (`davdata.nl/math`) wondered about the number of solutions of a binary puzzle. Below pictured is a binary puzzle as found in newspapers. Left is the original puzzle, right is the solved state.



	0						
			1		1		0
		0					
	1						
				1			
	0				1		
	0			0			
				0		0	



1	0	1	0	1	0	1	0
0	1	0	1	0	1	1	0
1	0	0	1	1	0	0	1
0	1	1	0	0	1	1	0
0	1	0	1	1	0	0	1
1	0	1	0	1	1	0	0
1	0	0	1	0	0	1	1
0	1	1	0	0	1	0	1

The problem is to fill the empty cells with a 0 or a 1 digit under following restrictions:

- A row or column may not have more than two consecutive zeros or ones
- Each row and each column must have an equal amount of ones and zeros
- No two columns and no two rows may be equal

Binary puzzles come in `4x4, 6x6, 8x8` (as above),`10x10, 12x12, 14x14` size.
A good puzzle has only one solution.

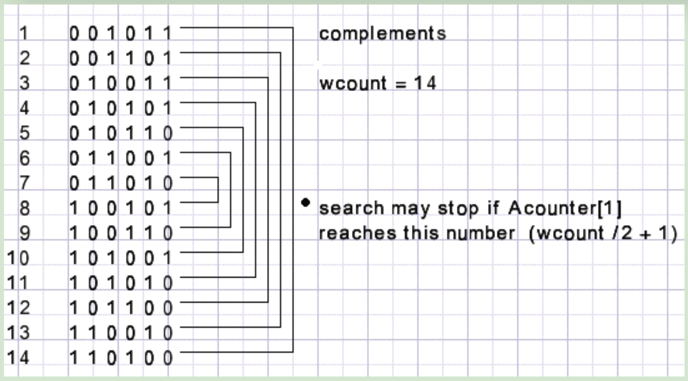
QUESTIONS ARE:

- How to test a puzzle for uniqueness
- Which digits may be removed in a solved puzzle to preserve a unique solution
- How many puzzles are possible for a nxn size puzzle?

The last question may be restated as

- How many solutions has a puzzle with only empty cells?
  This Delphi project tries to find the answer to the last question.
  As in many cases, this problem may be approached in an analytical or in a numerical way.
  The numerical approach is used. All possible solutions are generated and counted.
  Take a 6x6 puzzle as example. 6 bits represent a number ranging 0 to 63.

Because of the restrictions, only 14 of these numbers are valid, see below:

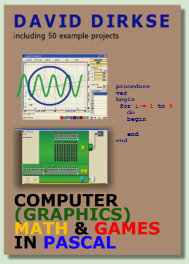


1	0 0 1 0 1 1
2	0 0 1 1 0 1
3	0 1 0 0 1 1
4	0 1 0 1 0 1
5	0 1 0 1 1 0
6	0 1 1 0 0 1
7	0 1 1 0 1 0
8	1 0 0 1 0 1
9	1 0 0 1 1 0
10	1 0 1 0 0 1
11	1 0 1 0 1 0
12	1 0 1 1 0 0
13	1 1 0 0 1 0
14	1 1 0 1 0 0

complements

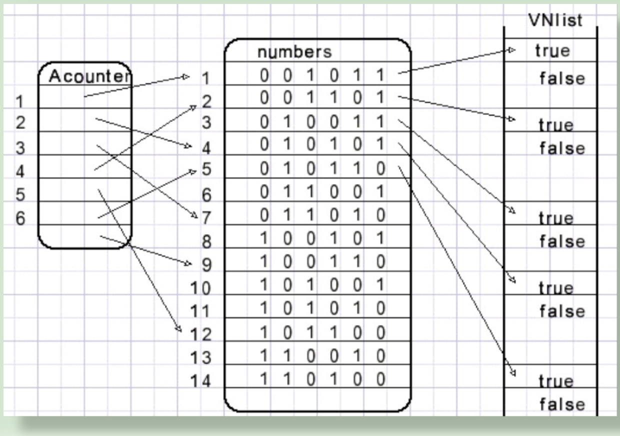
wcount = 14

• search may stop if Acounter[1] reaches this number (wcount /2 + 1)


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

To avoid superfluous work, these valid numbers are generated once and saved in array `numbers[1.. ]` Instead of using the 6 bit values, we may refer to rows and columns using the index of the `numbers[ ] array`.

When filling rows with valid numbers, columns may result with invalid numbers.
So, it is convenient to register per `number [0..63]` if it represents a valid number.
`Array VNlist[number]` of Boolean has value true for a valid number
.
To generate all possible solutions a counter system is needed.
This counter is called `Acounter[1..bitcount ]`
which holds indexes to the numbers array.
Each element of the Acounter may be considered a digit of number Acounter.

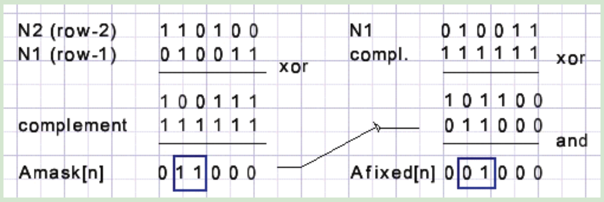


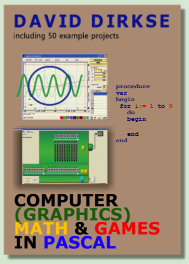
For an **nxn** puzzle, `Acounter` has n elements.
The VN list has **2^n** elements.
By using the numbers array, all values are valid.
In the selection of a new row, we already may avoid columns with more than two consecutive zeros or ones . For this purpose there are

```
Var  bitcountmask : word;
//  2^n – 1;    111111 for a 6x6 game, 11111111 for a 8x8 game
        Amask : array[1..14] of word;
        AFixed : array[1..14] of word;
For row n  {n > 2}
```

```
procedure makeFixedbits(n : byte);
var N1,N2,X : word;
begin
  N1 := numbers[Acounter[n-1]];
  N2 := numbers[Acounter[n-2]];
  X  := N1 xor N2;
  Amask [n] := x  xor bitcountmask;
  AFixed[n] := N1 xor bitcountmask;
end;
```

EXAMPLE (6*6 PUZZLE)




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

For row n bits 3 , 4 must be 0,1 to avoid 3 consecutive ones or zeros in a column.
When all 6 row numbers are selected these checks must be made

❶ A number may occur only once in the rows

❷ Each column must have an equal amount of ones and zeros

❸ A column may not have three (or more) consecutive ones and zeros

❹ A number may occur only once in the columns

❶ Each new row is compared to the previous rows to avoid reoccurrence.

❷ For the column checks, the columns have to be written as rows which is done by mirroring over the right top to left bottom diagonal.
This is a time consuming process. So, before writing the columns as rows the rows are summed and this sum is checked to be `(n/2)(2^n - 1) = 189` for 6x6 puzzles.
A 6x6 puzzle has three 1's per column so the sum is `3*(111111)bin = 3*63 = 189`.
Only if this test is passed, the columns are written as rows.

❸ A valid column number is simply indicated by the VNlist[number] being true.

❹ The check for multiple occurrence is done by comparing all numbers.

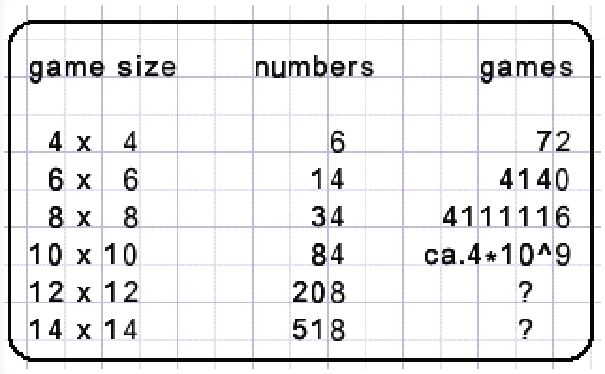
This row sum check saves 65% of the time for a 8x8 puzzle.

NOTICE that the row numbers and the solutions appear in complements.
So for each solution there is another one with all cells complemented.

Half the time is saved to count only the first 50% of the numbers in row 1 and counting solutions by increments of 2. For 8x8 puzzles the number of solutions is counted in 4.3 seconds.

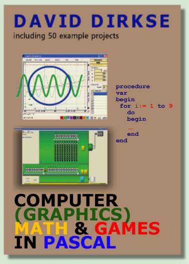
In case of a 10x10 puzzle, about 4 million solutions were counted per minute.
Expected counting time for all solutions is 16 hours.

RESULTS



game size	numbers	games
4 x 4	6	72
6 x 6	14	4140
8 x 8	34	4111116
10 x 10	84	ca.4*10^9
12 x 12	208	?
14 x 14	518	?

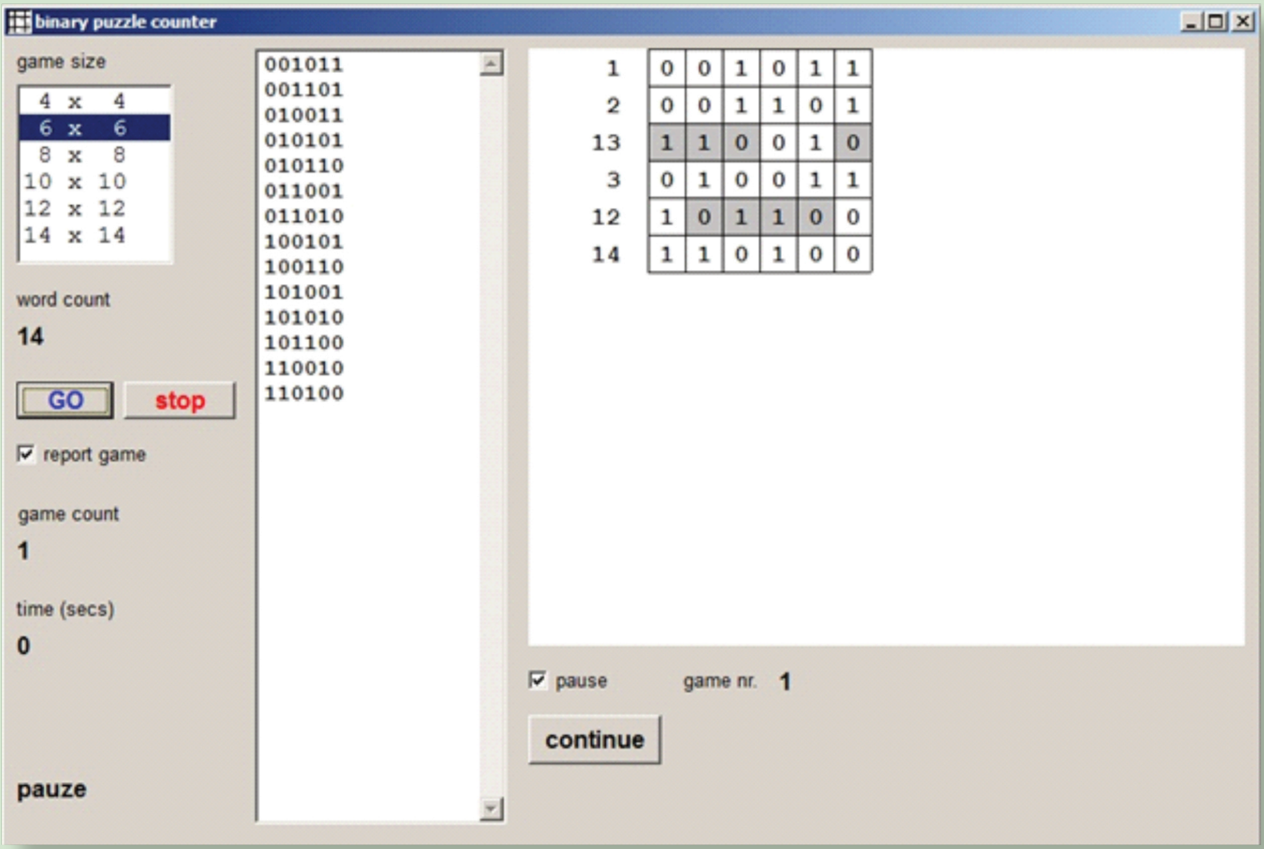
It looks like the number of solutions for empty puzzles increases by a factor 1000 for each next (even) nxn size.


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

THE PROGRAM

Heart of the program is:
`function NextAcount : boolean;`
Which updates the `Acounter` and returns true if a new value is set (*no overflow*) `Wcount` (*word count*) is the number of valid values in array `numbers[]`. `LInc` is a label.

`Bitcount = 4` for 4x4 puzzles, 8 for 8x8.....
A `false exit` (*end of search*) occurs if `Acounter[1]` updates to a number which has it's MSB set.
This prevents doubling the search time while only counting complements of earlier detected solutions.
A `true exit` takes place if `Acounter[bitcount]` is updated without conflict.



```
001011
001101
010011
010101
010110
011001
011010
100101
100110
101001
101010
101100
110010
110100
```

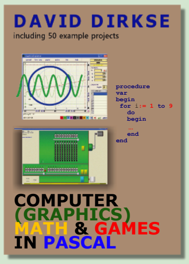
word count 14

game count 1

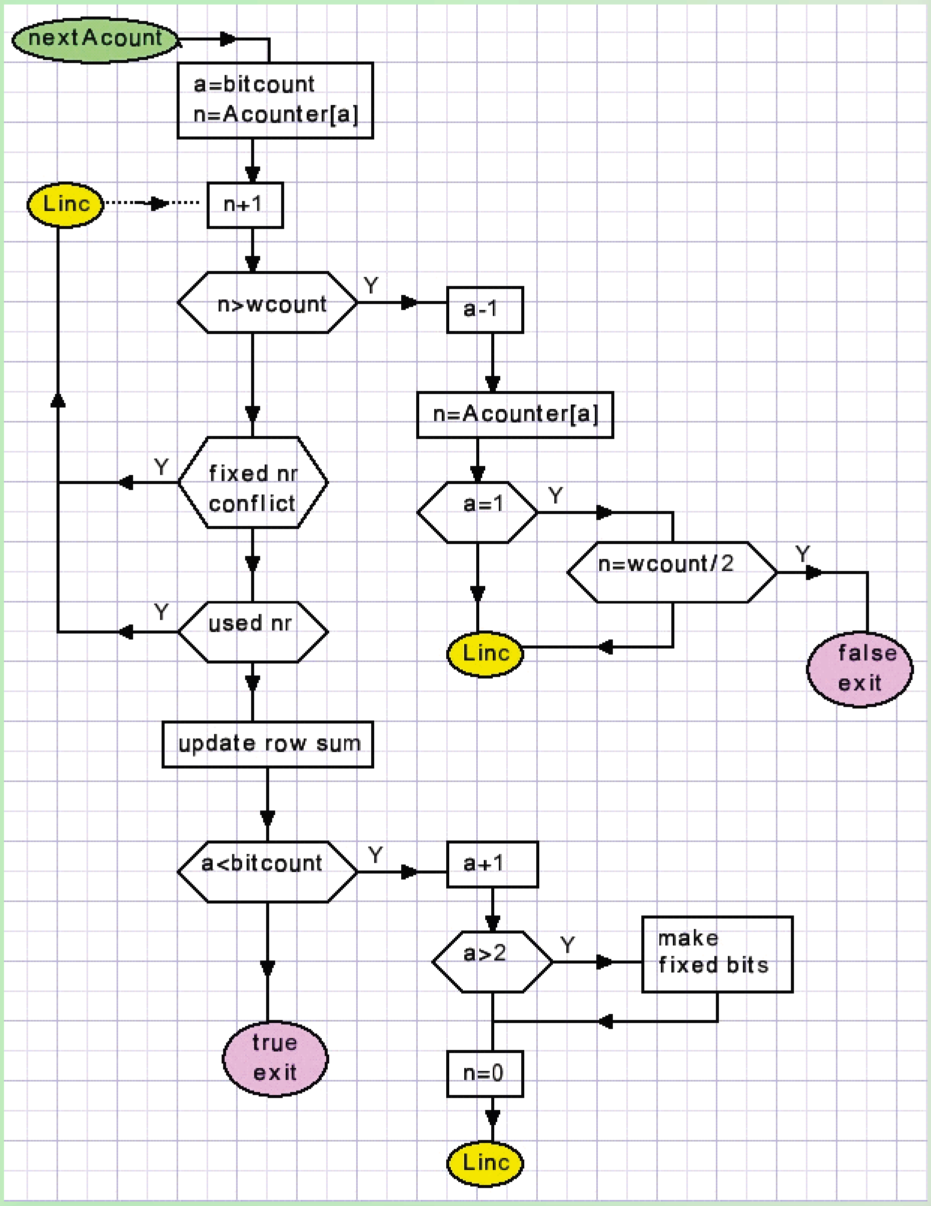
time (secs) 0

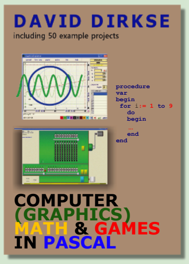
game nr. 1

1	0	0	1	0	1	1
2	0	0	1	1	0	1
13	1	1	0	0	1	0
3	0	1	0	0	1	1
12	1	0	1	1	0	0
14	1	1	0	1	0	0


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

This is the flowchart:





Notice that in this case the use of a label and GOTO statements generates far more readable code than structured programming statements `while` or `repeat`.

Other functions and procedures:

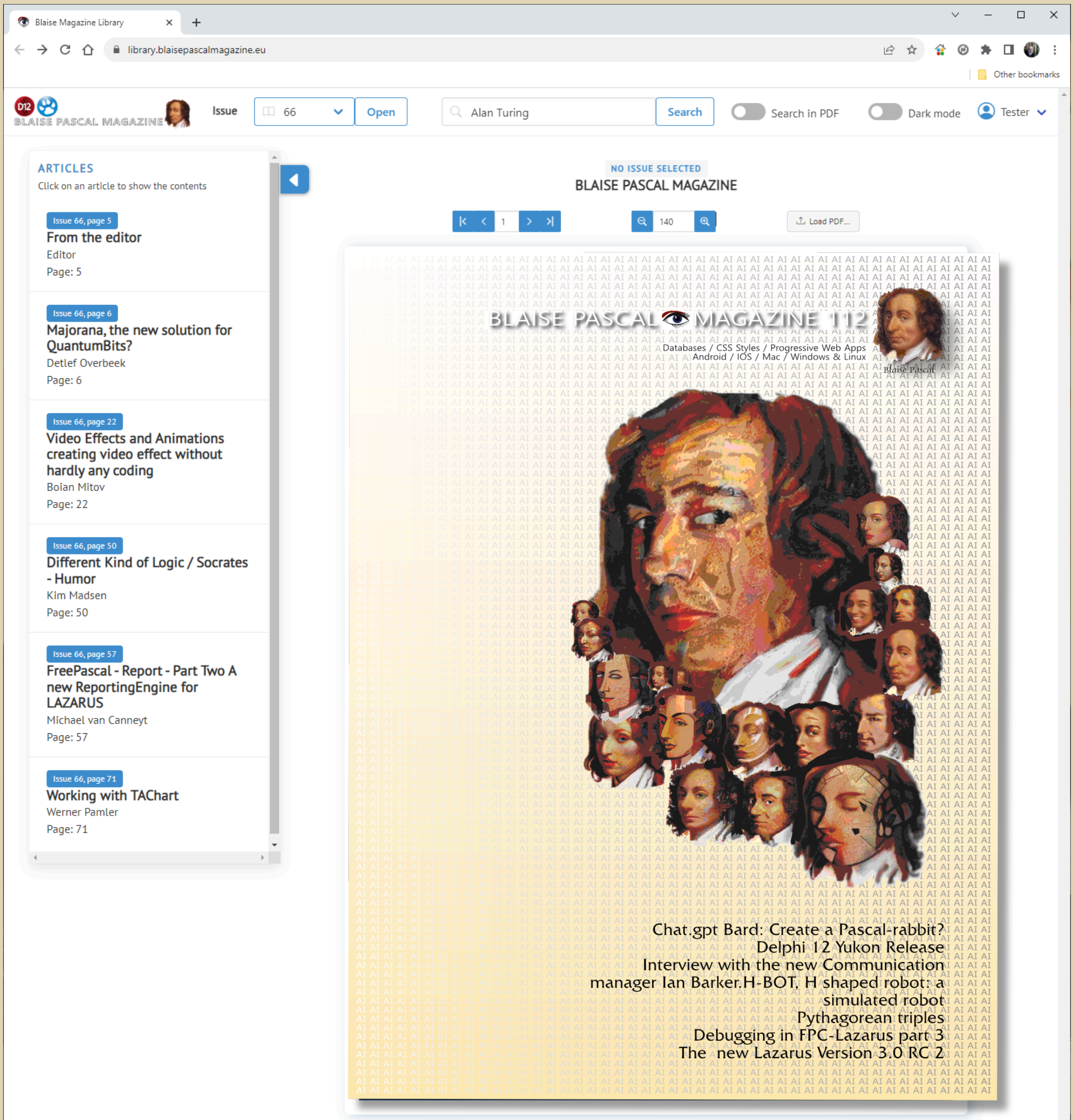
- `function testValid(w : word) : boolean;`
  Returns true is a number (w) is valid (see restrictions)
- `procedure MakeNumbers;`
  Generates the numbers[ ] list and the VNlist[ ].
- `procedure makeFixedbits(ai : byte);`
  Sets the AMask[ai] and AFixed[ai] entries to avoid wrong numbers in a column.
- `function UsedWord(a : byte; wix : word) : boolean;`
  Returns true if index wix not present in Acounter[1..a-1]. Prevents identical rows.
- `procedure setGame1;`
  Called at t the start to supply the 1st game.
- `procedure makeVGame;`
  Transfers columns to rows in array VGame[ ].
- `function checkVGame : boolean;`
  Returns true if VGame array contains valid numbers.

This concludes the description of the binary puzzle solutions counter.
Please refer to the source code for details.

THE NEW SUBSCRIPTION MODEL OF BLAISE PASCAL MAGAZINE

1. SUBSCRIPTION: PER YEAR - NOTHING CHANGES ISSUES STARTING AT THE LATEST ISSUE AVAILABLE +1 YEAR / CODE INCLUDED € 70,00 FOR ALL COUNTRIES INCLUDED INTERNET (LIBRARY) USE FOR ALL MAGAZINES FROM 1- THE LATEST ISSUE FOR ALL COUNTRIES

2. LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY.€ 120,00 FOR ALL COUNTRIES



ARTICLES
Click on an article to show the contents

Issue 66, page 5
From the editor
Editor
Page: 5

Issue 66, page 6
Majorana, the new solution for QuantumBits?
Detlef Overbeek
Page: 6

Issue 66, page 22
Video Effects and Animations creating video effect without hardly any coding
Boian Mitov
Page: 22

Issue 66, page 50
Different Kind of Logic / Socrates - Humor
Kim Madsen
Page: 50

Issue 66, page 57
FreePascal - Report - Part Two A new ReportingEngine for LAZARUS
Michael van Canneyt
Page: 57

Issue 66, page 71
Working with TAChart
Werner Pamler
Page: 71

BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
Blaise Pascal

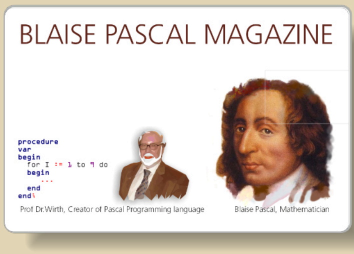
Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

https://www.blaisepascalmagazine.eu/product-overview/

USE WHERE EVER THE INTERNET IS AVAILABLE

LIB-STICK ON USB CREDIT CARD BLAISE PASCAL MAGAZINE

LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY € 120,00

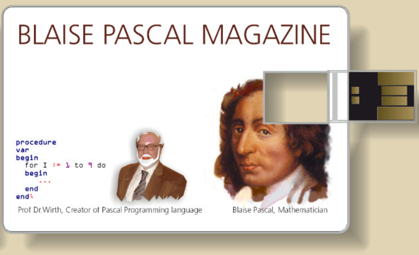

BLAISE PASCAL MAGAZINE
Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician

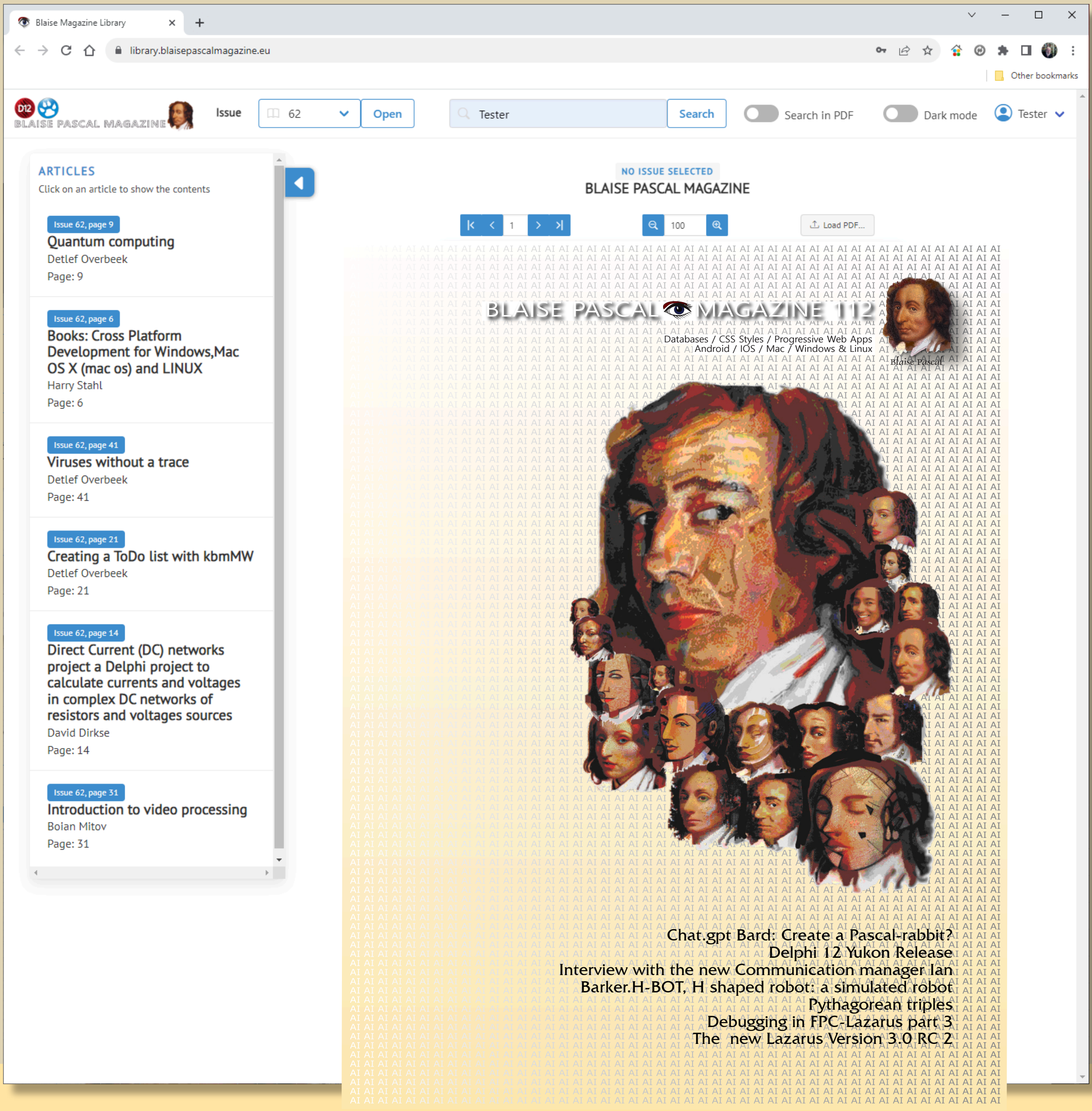
```
procedure
var
begin
  for I := 1 to 9 do
  begin
    ...
  end
end;
```

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



ARTICLES
Click on an article to show the contents

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

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

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

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

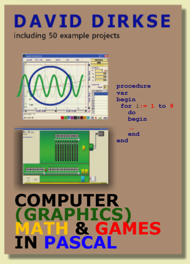
Issue 62, page 14
Direct Current (DC) networks project a Delphi project to calculate currents and voltages in complex DC networks of resistors and voltages sources
David Dirkse
Page: 14

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

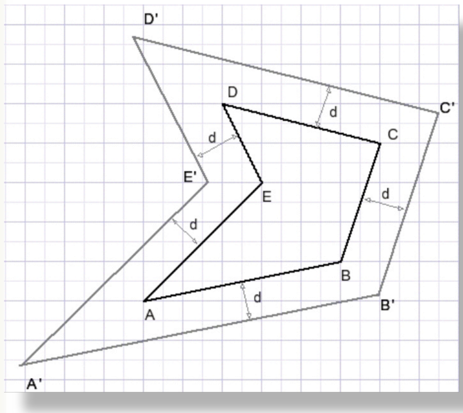
BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

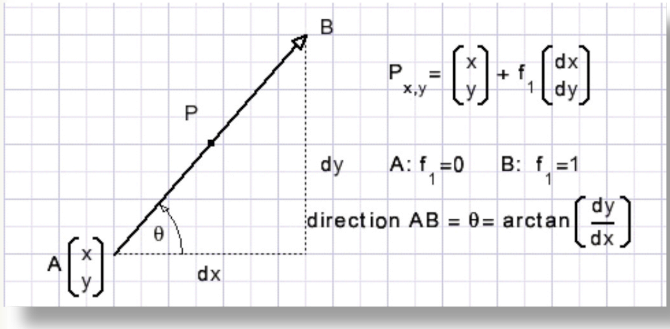
STARTER EXPERT 7 D11


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

A visitor of my website `davdata.nl/math` asked: "how to expand a polygon?".
Below is pictured polygon ABCDE and the expansion A'B'C'D'E'



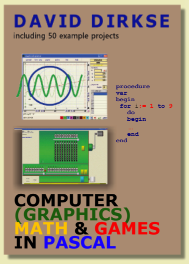
All edges are shifted outward over a distance d.
The new vertices A'....E' are the intersections of the shifted edges AB, BC,......EA.
How to calculate such an expansion?
Edges AB, BC... are vectors, they are defined by their length and direction.
Vector AB (see picture below):
Each value of f1 defines a point on AB. Point A : f1=0, point B : f1=1.
f1 > 1 defines a point on the extension of AB,past B.
f1 < 0 defines a point before A.


B
P
$P_{x,y} = \begin{pmatrix} x \\ y \end{pmatrix} + f_1 \begin{pmatrix} dx \\ dy \end{pmatrix}$
dy
A: f_1=0
B: f_1=1
direction AB = θ= $\arctan\left(\frac{dy}{dx}\right)$
θ
A $\begin{pmatrix} x \\ y \end{pmatrix}$
dx

The polygon is described as a list of vectors.

```
Const maxpolypoint = 40;      // maximal number of vertices
type  Tvector = record
             x,y,dx,dy : single;
             dir : double;       // direction 0..2*pi
             modulus : single; // length=sqrt(sqr(dx)+sqr(dy))
             end;
      TVectorList = array[1..maxpolypoint] of TVector;
var
      vectorlist : TVectorlist;
```

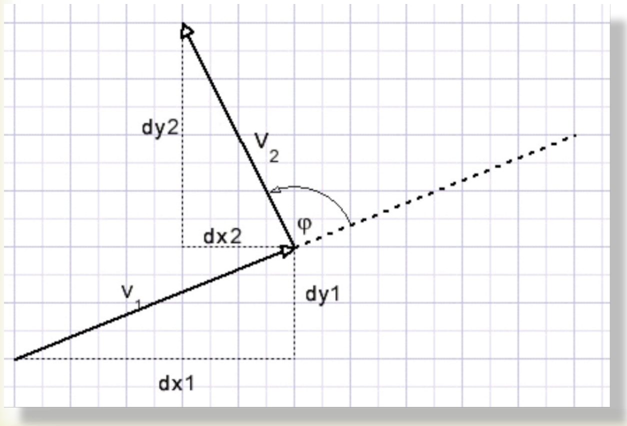
The direction is measured in radians.
Horizontal right is direction 0.



DIRECTION OF A VECTOR:

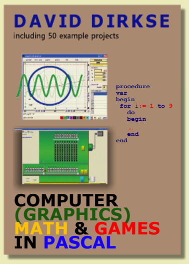
```
Const pi05 = 0.5*pi;
    pi15 = 1.5*pi;
    pi2 = 2*pi;
function VDir(deltaX,deltaY : double) : double;
// return direction of vector in radians
// (+,0) = 0; (0,+) = 0.5pi ; (-,0) = pi ; (0,-) = 1.5pi
begin
   if deltaX = 0 then
     begin
       if deltaY > 0 then result := pi05 else result := pi15;
       exit;
     end;
   result := arctan((deltaY)/(deltaX));
   if deltaX < 0 then result := result + pi;
   if result < 0 then result := result + pi2;
end;
```

Direction difference (angle) between vectors:
Next picture shows the angle between vectors v1 and v2.



```
function V12angle(dir1,dir2 : double) : double;
// dir1,dir2 : direction in radians
// return angle between vectors v1,v2 in radians
// -pi.....+pi
begin
  result := dir2 - dir1;
  if result > pi then result := result-pi2
  else if result < -pi then result := result+pi2;
end;
```

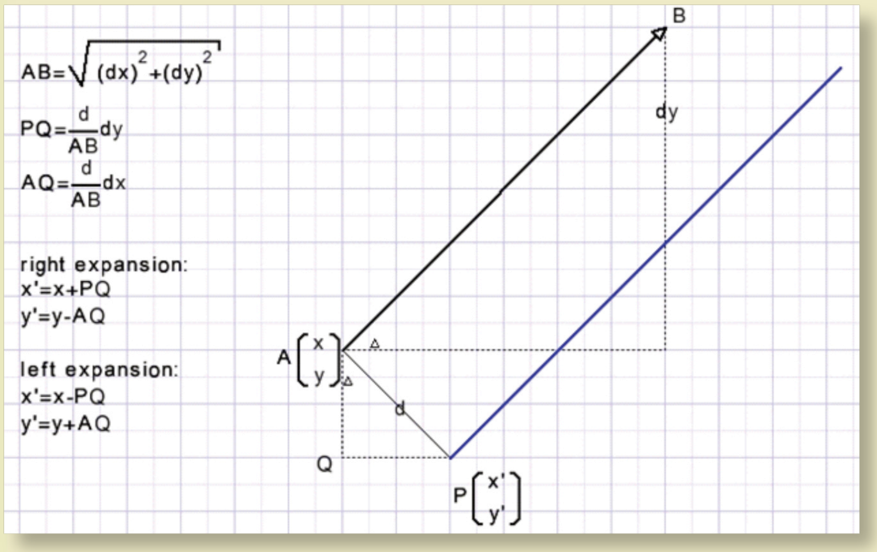
When shifting an edge, say AB, problem is: “left or right”?
Moving around the polygon starting at A route ABCDE or route AEDCB may be taken.
For the first route, edges have to be shifted right for expansion, the second route needs left shifts for expansion.
polygons may be traversed CW or CCW.
Summing the angles (*direction differences*) provides the answer.



```
function SumAngles : double;
//add angles between vectors
//vcount is number of vectors
var i : byte;
begin
  result := 0;
  for i := 1 to vcount-1 do
    result := result + V12Angle(vectorlist[i].dir,
                                            vectorlist[i+1].dir);
  result := result + V12Angle(vectorlist[vcount].dir,
                                            vectorlist[1].dir);
end;
```

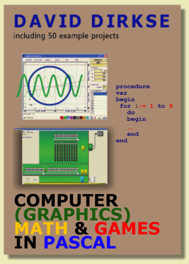
A sum of pi2 indicates CCW traversion, -pi2 shows CW traversion.
CCW traversion needs "right" expansion. CW traversion needs "left" expansion.

Right expansion by distance d.

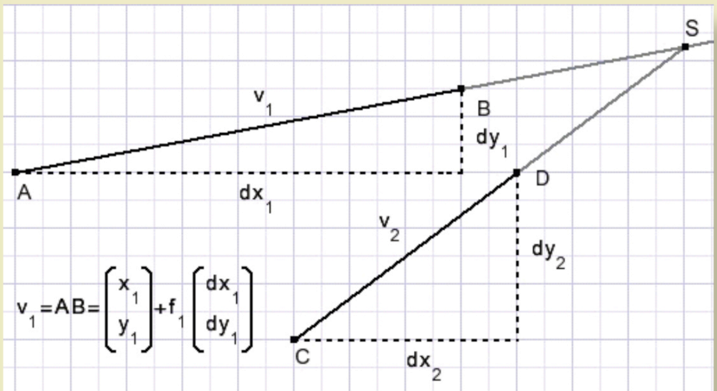


The new (expanded) points are the intersections of the shifted (blue) vectors.

```
Const offset = 0.025; //displacement of 1 pixel,scale 40 pixels/cm.
procedure shiftvector(var v : Tvector; R : boolean);
// R : true for right shift
var dd, m: single;
begin
  if R then m := 1 else m := -1;
    with v do
    begin
      dd := offset/modulus;
      x  := x+dd*dy*m;
      y  := y-dd*dx*m;
    end;
end;
```


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

INTERSECTIONS
Calculation of the intersection S of vectors AB en CD



$$v_1 = AB = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + f_1 \begin{pmatrix} dx_1 \\ dy_1 \end{pmatrix}$$

$$v_2 = CD = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + f_2 \begin{pmatrix} dx_2 \\ dy_2 \end{pmatrix}$$

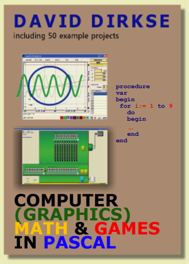
$$S : \begin{cases} x_1 + f_1.dx_1 = x_2 + f_2.dx_2 & *dy_2 \\ y_1 + f_1.dy_1 = y_2 + f_2.dy_2 & *dx_2 \end{cases} \quad -$$

$$f_1 \underbrace{(dx_1.dy_2 - dy_1.dx_2)}_{d} = dy_2 \underbrace{(x_2 - x_1)}_{vx} - dx_2 \underbrace{(y_2 - y_1)}_{vy}$$

$$f_1 = \frac{(vx.dy_2 - vy.dx_2)}{d} \qquad f_2 = \frac{(vx.dy_1 - vy.dx_1)}{d}$$

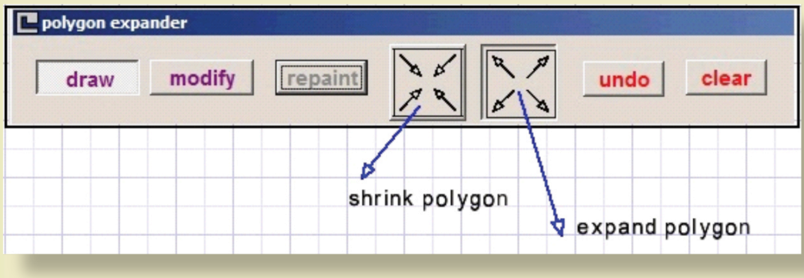
Coordinates of S:

$$S \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + f_1 \begin{pmatrix} dx_1 \\ dy_1 \end{pmatrix}$$

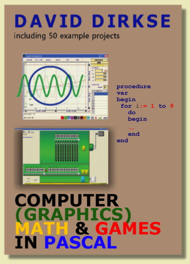

DAVID DIRKSE
including 50 example projects
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

```
Const frnd = 1e-6; // floating point rounding

Var  fvalid : boolean = false; // false if vectors are parallel
     f1,f2 : single;

procedure vrsect(const v1,v2 : TVector);
// calculate intersection of vectors v1,v2
// line1 = (v1.x1,v1.y1) +f1*(v1.dx,v1.dy)
// line2 = (v2.x1,v2.y1) +f2*(v2.dx,v2.dy)
// return f1,f2,fvalid
var d,vx,vy : single;
begin
   d := v1.dx*v2.dy - v1.dy*v2.dx; // discriminant
   if d = 0 then begin
                  fvalid := false; exit;
   end;

   fvalid := true;
   vx := v2.x - v1.x;
   vy := v2.y - v1.y;
   f1 := (vx*v2.dy - vy*v2.dx)/d;
   f2 := (vx*v1.dy - vy*v1.dx)/d;

   if abs(f1) < frnd then f1 := 0;  // round to 1e-6
   if abs(f2) < frnd then f2 := 0;
   if abs(f1-1) < frnd then f1 := 1;
   if abs(f2-1) < frnd then f2 := 1;
end;
```

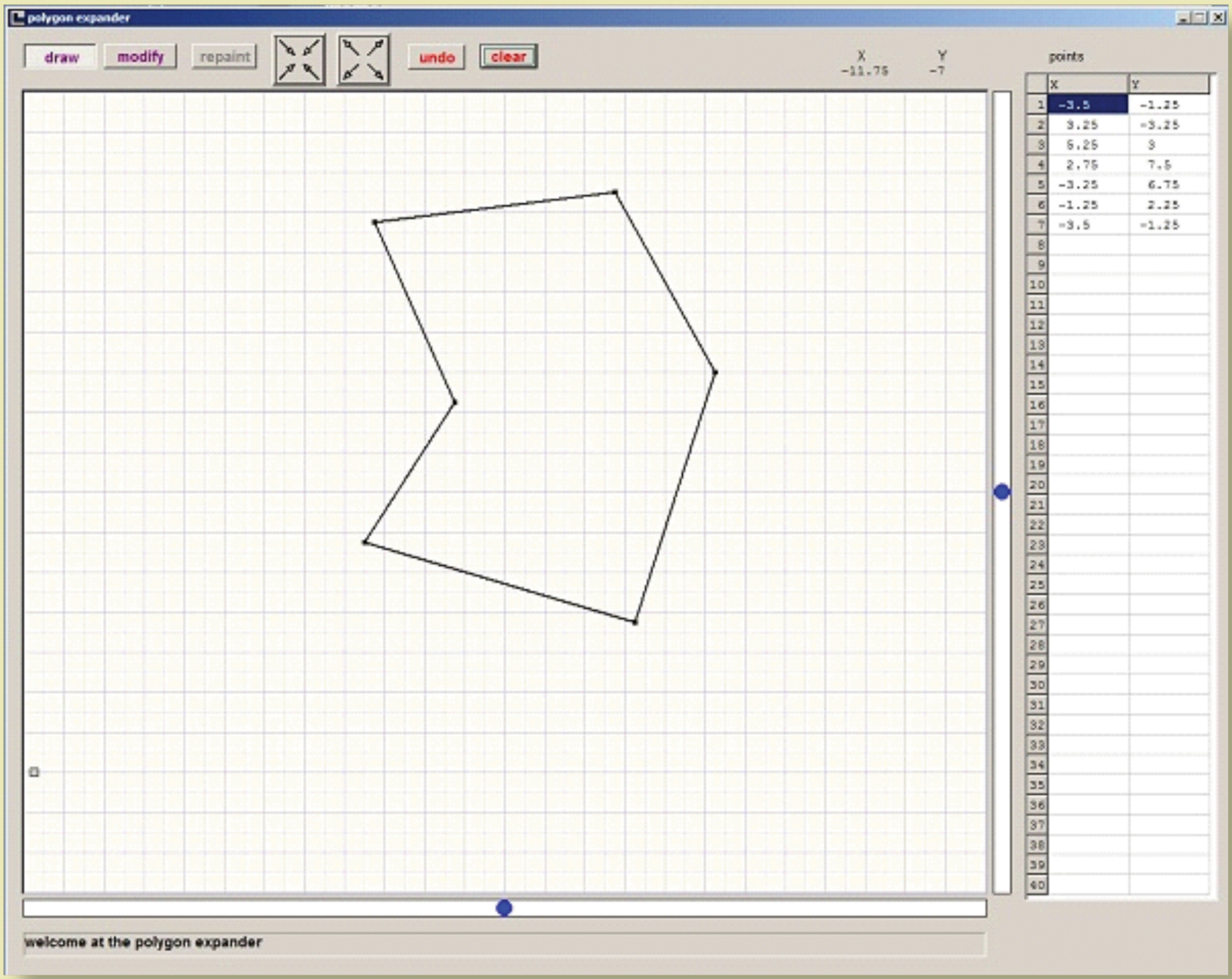
PROGRAM



Draw : mouse down and move to draw vector.
Modify : mouse on point, mouse down and move to change vector.
Other buttons are self explanatory.
NOTE : cursor increments at 10 pixel intervals unless **SHIFT** key is hold down.


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

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



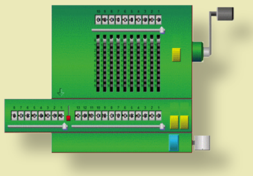
	X	Y
1	-3.5	-1.25
2	3.25	-3.25
3	6.25	3
4	2.75	7.5
5	-3.25	6.75
6	-1.25	2.25
7	-3.5	-1.25
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		

welcome at the polygon expander

This concludes the polygon expansion description.

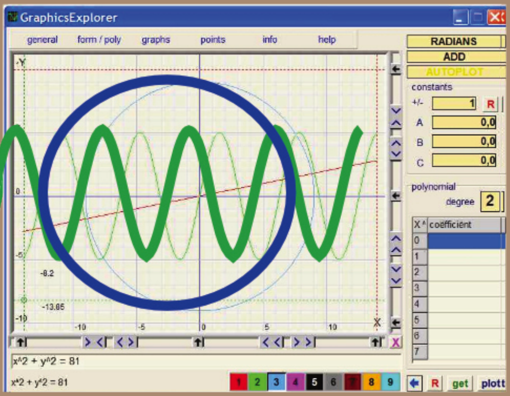
ADVERTISEMENT

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



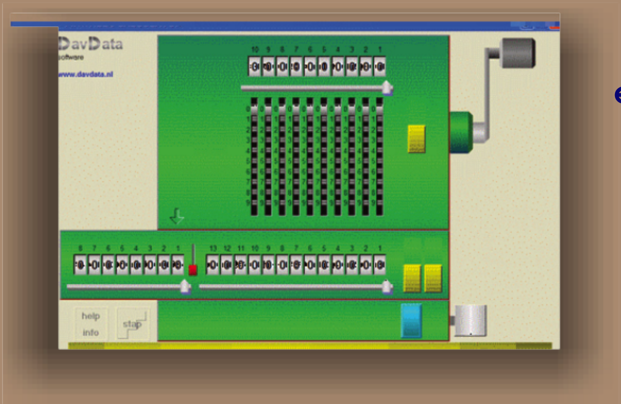
DAVID DIRKSE

including 50 example projects



x^2 + y^2 = 81

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



COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

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

ease see our products here:
ttps://www.nexusdb.com

Our YouTube channel: https://www.youtube.com/@nexusdb9051

THE BAD WAY TO PLAY CHESS: 3D PHYSICS FUN USING CASTLE GAME ENGINE (PART 2)

BY MICHALIS KAMBURELIS

Starter Expert D12

Overview of this article

❶ INTRODUCTION

Welcome to the second part of the article about creating a simple 3D physics game using **Castle Game Engine**.

Castle Game Engine is a cross-platform (*desktop, mobile, consoles*) 3D and 2D game engine using modern **Pascal**. It's **free and open-source** and works with both **FPC** and **Delphi**.

In the first part, we learned how to use the visual editor and we have designed a chessboard with chess pieces. Then we used physics to throw the chess piece, such that it collides and knocks down other chess pieces. Remember this is a bad way to play chess. But it's really fun!
If you have missed the first part, you can still "jump in" at this point.
You can search for the last **issue 112** at the **Blaise Pascal Magazine** web site, search on the **Internet Library** version of your subscription, or just download **Castle Game Engine** from `https://castle-engine.io/` and either set up the chessboard and chess pieces yourself, or use our ready example project from
`https://github.com/castle-engine/bad-chess/`
in the subdirectory `project/version_1_designed_in_editor`.
This project version is a good starting point for this article part.

We encourage you to follow this article and perform all the steps yourself,
to create a similar toy. If you ever get stuck, you can look at the finished project. It is available in the `subdirectory project/version_2_with_code` in the same repository, `https://github.com/castle-engine/bad-chess/` It's the final project, with everything described in this article done and working.
And if you really just want to **play the worst version of chess, right now, you can download the ready compiled game** (for **Linux** or **Windows**) from
`https://castle-engine.itch.io/bad-chess` . Enjoy!

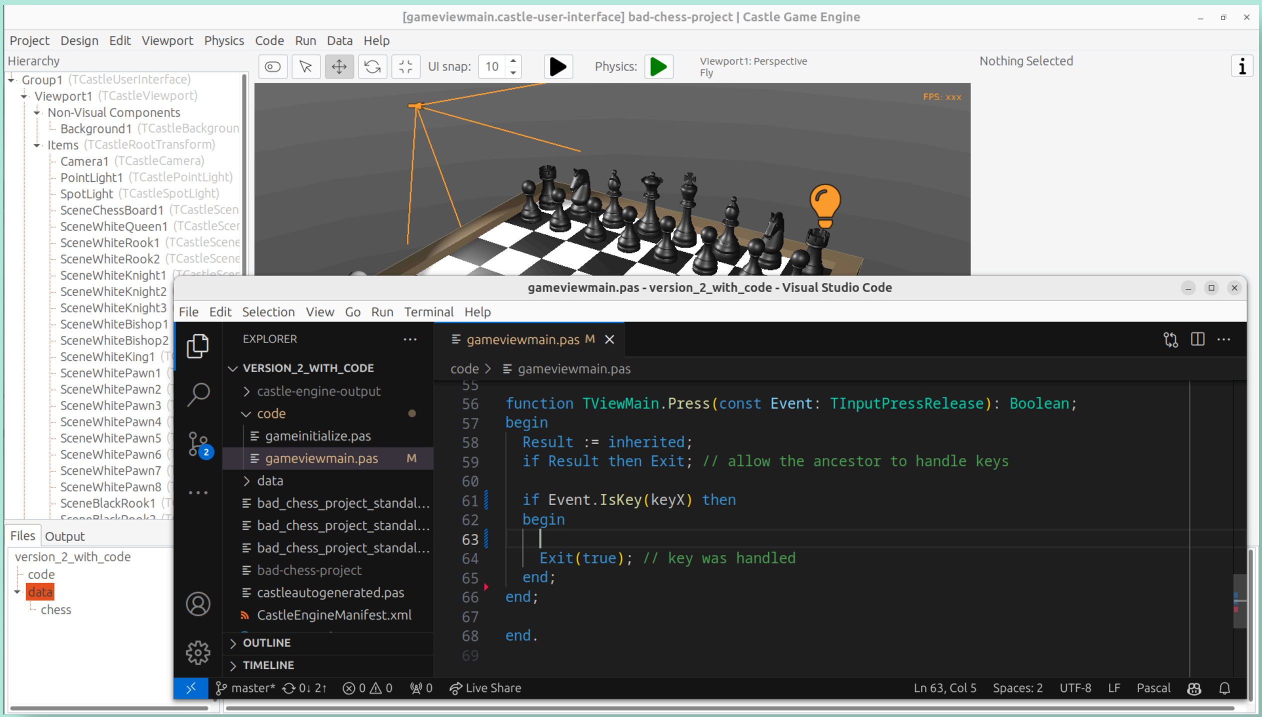
❷ CODING THE GAME

The focus of this part is to learn how to use **Pascal** code to make things happen in your game.

The core of **Castle Game Engine** is just a set of **Pascal** units that can be compiled using **FPC** and **Delphi**. Thus the games we create are also just regular **Pascal** programs that happen to use a few **Castle Game Engine** units. This means that you can use the workflow you already know and like, with whatever **Pascal** text editor and compiler you prefer.

In particular we support **Delphi, Lazarus, VS Code** or any other custom editor (like **Emacs**).
We have a dedicated documentation with some IDE-specific hints on
`https://castle-engine.io/manual_ide.php` . Basically just open in **Castle Game Engine** editor the panel **"Preferences → Code Editor"**, configure there which **Pascal IDE** you use, and everything should work out-of-the-box. If you double-click on a Pascal file from **CGE** editor, it will open in the text editor you configured.

Specifically for **VS Code** users, the page `https://castle-engine.io/vscode` contains information how to setup **VS Code** with **Castle Game Engine LSP** server to get great code completion. We are working right now on a dedicated **Castle Game Engine** extension for **VS Code** that will make this integration even easier.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Hierarchy
Nothing Selected
gameviewmain.pas - version_2_with_code - Visual Studio Code
EXPLORER
VERSION_2_WITH_CODE
gameviewmain.pas
Ln 63, Col 5
Pascal

```
function TViewMain.Press(const Event: TInputPressRelease): Boolean;
begin
  Result := inherited;
  if Result then Exit; // allow the ancestor to handle keys

  if Event.IsKey(keyX) then
  begin

    Exit(true); // key was handled
  end;
end;

end.
```

NOTE that, while the focus of this chapter is to write Pascal code, we do not stop using the **Castle Game Engine Editor.** There are a few things you can do in the editor to make the design "friendly" to the code manipulation and we will explore them in this article.
So writing Pascal code, and editing the design visually, go hand-in-hand.

❸ EXERCISES

3.1. HANDLE A KEY PRESS TO CHANGE POSITION OF AN OBJECT

Let's start simple. First goal: When the user presses a key x,
we want to move the black king chess piece a bit higher. It's a simple test that we can:

- **React** to user input (key press).
- **In response**, do something interesting in 3D world (*move a chess piece*).

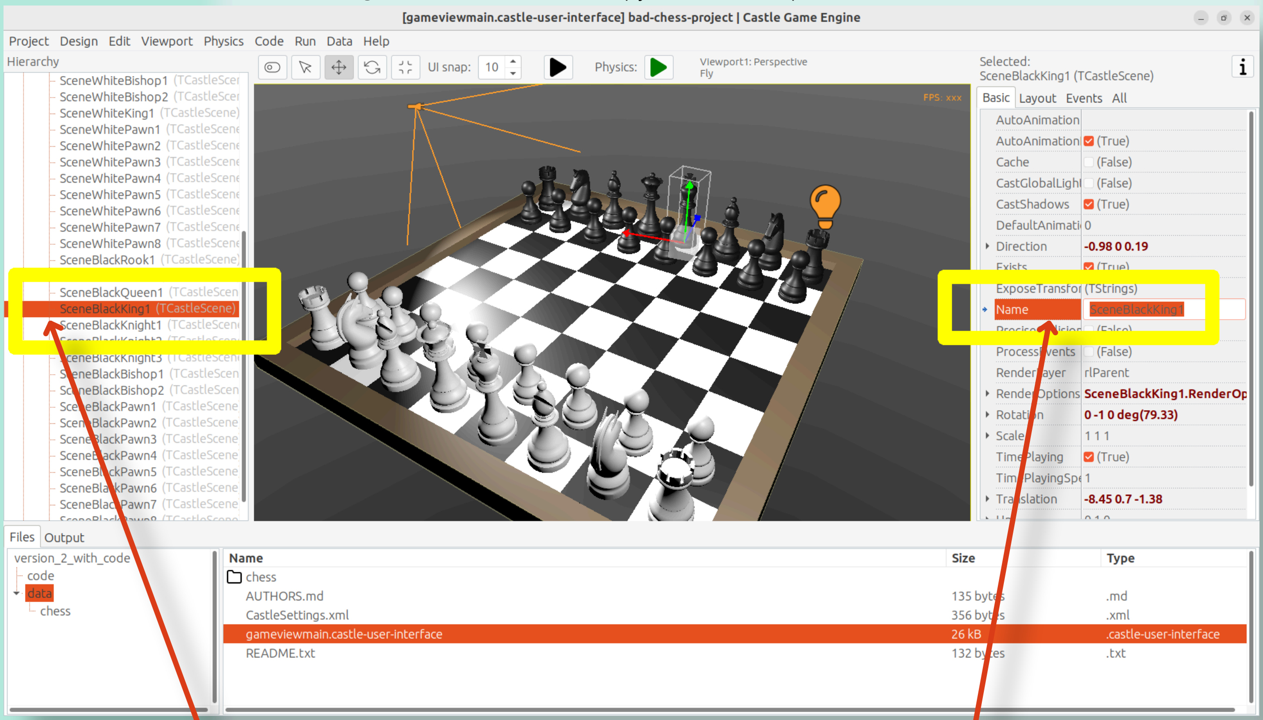
Most of the code you write in **Castle Game Engine** is placed in a unit associated with a view.
We talked about what is a view in **Castle Game Engine** in the previous article part, the short recap is that you use views similar to how you use forms in a typical **Delphi FMX / VCL** or **Lazarus LCL** application: a view is a visual design (*in data/gameviewmain.castle-user-interface*) and associated code (*in* `code/gameviewmain.pas`).

So let's open the file `code/gameviewmain.pas` in your favorite **Pascal IDE**.
In the **Castle Game Engine Editor**, you can just use the bottom "Files" panel.
Enter the code subdirectory and double-click on the `gameviewmain.pas` file.
Alternatively, you can just open your **Pascal IDE** and from it open the **Pascal project**.
The basic project files (*like* `my_project.dproj` *for Delphi or* `my_project.lpi` *for Lazarus*) have been already generated for you.
Keep the **Castle Game Engine** visual editor open too, with our view design `data/gameviewmain.castle-user-interface` . We will occasionally adjust or consult our visual design, to make sure it is useful for our code logic.

For start, we want to know the name of the component representing the black king.
Just as you've seen when designing **Lazarus** and **Delphi** forms, every component has a name which corresponds to how this component can be accessed from code.
You can edit the component name in **Castle Game Engine** by either editing the **Name row** in the **Object Inspector** (*on the right*) or editing the name in the hierarchy (*on the left,* next page)

3.1. HANDLE A KEY PRESS TO CHANGE POSITION OF AN OBJECT (CONTINUATION 1)

Simply click on the component name in hierarchy or press **F2** to go into name editing. On the screenshot below, you can see that black king is named **SceneBlackKing1**. I can use **Ctrl+C** to copy this to the clipboard.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Files Output

SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackQueen1 (TCastleScene)
SceneBlackKing1 (TCastleScene)
SceneBlackKnight1 (TCastleScene)
SceneBlackKnight2 (TCastleScene)
SceneBlackKnight3 (TCastleScene)
SceneBlackBishop1 (TCastleScene)
SceneBlackBishop2 (TCastleScene)
SceneBlackPawn1 (TCastleScene)
SceneBlackPawn2 (TCastleScene)
SceneBlackPawn3 (TCastleScene)
SceneBlackPawn4 (TCastleScene)
SceneBlackPawn5 (TCastleScene)
SceneBlackPawn6 (TCastleScene)
SceneBlackPawn7 (TCastleScene)
SceneBlackPawn8 (TCastleScene)

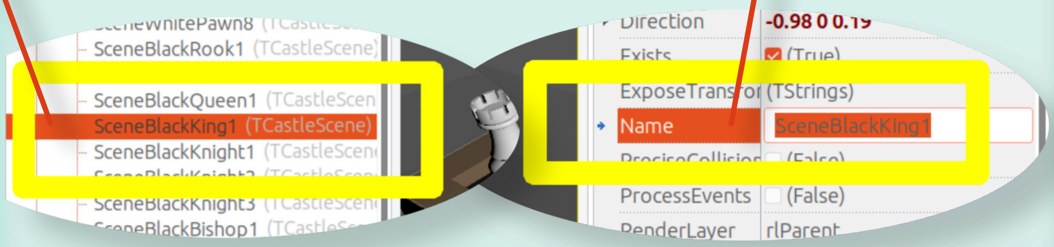
Selected:
SceneBlackKing1 (TCastleScene)

Basic Layout Events All

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLight	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	-0.98 0 0.19
Exists	✓ (True)
ExposeTransfor	(TStrings)
Name	SceneBlackKing1
PreciseCollision	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneBlackKing1.RenderOp
Rotation	0 -1 0 deg(79.33)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSpe	1
Translation	-8.45 0.7 -1.38

version_2_with_code
code
data
chess

Name	Size	Type
chess		
AUTHORS.md	135 bytes	.md
CastleSettings.xml	356 bytes	.xml
gameviewmain.castle-user-interface	26 kB	.castle-user-interface
README.txt	132 bytes	.txt


SceneBlackRook1 (TCastleScene)
SceneBlackQueen1 (TCastleScene)
SceneBlackKing1 (TCastleScene)
SceneBlackKnight1 (TCastleScene)
SceneBlackKnight3 (TCastleScene)
SceneBlackBishop1 (TCastleScene)
Direction -0.98 0 0.19
Exists (True)
ExposeTransfor (TStrings)
Name SceneBlackKing1
PreciseCollision (False)
ProcessEvents (False)
RenderLayer rlParent

NOTE that, for this first code exercise, we assume that the chess piece (**SceneBlackKing1**) does not have any physics components.
If you have added `TCastleRigidBody` or `TCastleXxxCollider` components as behaviors of `SceneBlackKing1`, please remove them for now. We will restore them in the next exercise.

Now we have to declare the variable with the exact same name in the view.
It will be automatically initialized to point to the component when we start the view.
Do this in the `published` section of the class `TViewMain`.

3.1. HANDLE A KEY PRESS TO CHANGE POSITION OF AN OBJECT (CONTINUATION 2)

This is how the end result should look like:

```
uses Classes,
 CastleVectors, CastleComponentSerialize,
 CastleUIControls, CastleControls, CastleKeysMouse, CastleScene;

type
  { Main view, where most of the application logic takes place. }
  TViewMain = class(TCastleView)
  published
    { Components designed using CGE editor.
     These fields will be automatically initialized at Start. }
    LabelFps: TCastleLabel;
    SceneBlackKing1: TCastleScene; //< new line
  public
  ... uses Classes,
  CastleVectors, CastleComponentSerialize,
  CastleUIControls, CastleControls, CastleKeysMouse, CastleScene;

type
  { Main view, where most of the application logic takes place. }
  TViewMain = class(TCastleView)
  published
    { Components designed using CGE editor.
     These fields will be automatically initialized at Start. }
    LabelFps: TCastleLabel;
    SceneBlackKing1: TCastleScene; //< new line
  public
  ...
```

NOTE: Right now, the **Castle Game Engine** editor doesn't do this automatically for you.

That is, we don't automatically update your **Pascal** sources to declare all the components.
We have a plan to do this soon.

The user experience will have to be a bit different than on **Delphi** and **Lazarus** forms, because the game visual designs can easily have hundredths of components that are not supposed to be used from code, so synchronizing them all with **Pascal** code would create unnecessary noise in your **Pascal** unit.
We will instead make a button to only expose a subset of designed components for code.

Once you have declared the published field, we can access the **SceneBlackKing1** from code, getting and setting its properties, calling its methods anywhere we like.
For this exercise, let's modify the `Translation` property of our chess piece, which changes the position of the object.

It is a property of type `TVector3`. `TVector3` is an advanced record in **Castle Game Engine** that represents **3D vector** - in this case a position, but we use it in many other cases too, e.g. to represent a direction or even **RGB color**.
There are a number of useful things defined to help you work with `TVector3`, in particular:

- `Vector3(...) function` returns a new `TVector3` value with given coordinates.
- The arithmetic operators like + work with `TVector3` values.

This means that we can easily move object by writing a code like this:

```
SceneBlackKing1.Translation := SceneBlackKing1.Translation + Vector3(0, 1, 0);
```

3.1. HANDLE A KEY PRESS TO CHANGE POSITION OF AN OBJECT (CONTINUATION 3)

Where to put this statement? In general, you can use this code anywhere in your `view` (*as long as it executes only after the view has been started*).
In this case, we want to react to user pressing a `key x`. To achieve this, we can edit the `TViewMain.Press` method in the view. The empty implementation of this method is already present, with some helpful comments, so we can just fill it with our code:

```pascal
function TViewMain.Press(const Event: TInputPressRelease): Boolean;
begin
  Result := inherited;
  if Result then Exit; // allow the ancestor to handle keys

  if Event.IsKey(keyX) then
  begin
    SceneBlackKing1.Translation :=
      SceneBlackKing1.Translation + Vector3(0, 1, 0);
    Exit(true); // key was handled
  end;
end;
```

Build and run the game (*e.g. by pressing* **F9** in **Castle Game Engine** editor, or in **Delphi**, or in **Lazarus**) and press X to see how it works.

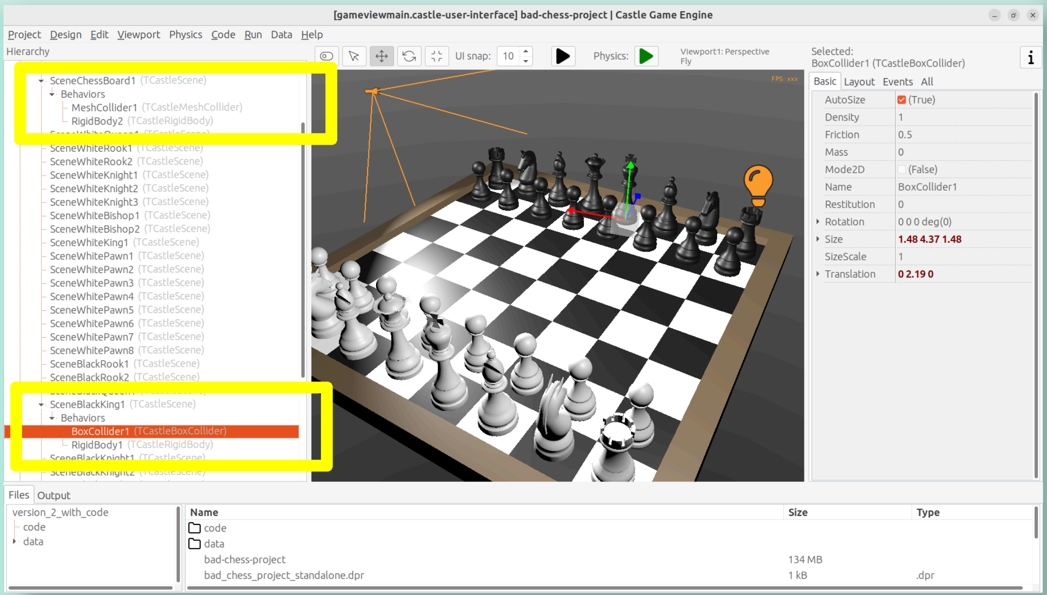
3.2. PUSH THE CHESS PIECE USING PHYSICS

Let's do one more exercise.

Let's make sure we can use code to push (*flick, throw*) a chess piece using physics.
The chess piece we push, and the direction in which we push it, will be hardcoded in this exercise.
But we will get confidence that we can use physics from **Pascal** code.

Let's use the black king again.
To do this, make sure to add the physics components to the relevant chess piece.
We described how to do this in 1st article part, the quick recap is to right-click on the component (**SceneBlackKing1** *in this case*) and from the context menu choose
"**Add Behavior → Physics → Collider → Box (TCastleBoxCollider)**".
Make sure you also have `physics` (*with* `TCastleMeshCollider`) active on the chess board, otherwise the chess piece would fall down due to `gravity` as soon as you run the game.

This is how it should look like:


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Basic Layout Events All
Files Output

```
SceneChessBoard1 (TCastleScene)
  Behaviors
    MeshCollider1 (TCastleMeshCollider)
    RigidBody2 (TCastleRigidBody)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
SceneBlackKing1 (TCastleScene)
  Behaviors
    BoxCollider1 (TCastleBoxCollider)
    RigidBody1 (TCastleRigidBody)
SceneBlackKnight1 (TCastleScene)
```

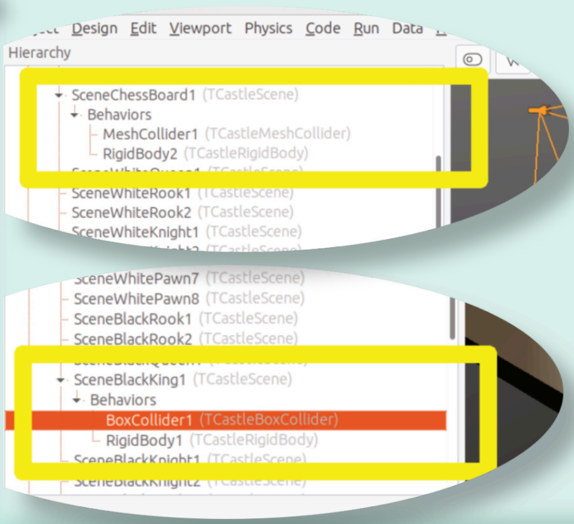
Selected:
BoxCollider1 (TCastleBoxCollider)

AutoSize	✓ (True)
Density	1
Friction	0.5
Mass	0
Mode2D	(False)
Name	BoxCollider1
Restitution	0
Rotation	0 0 0 deg(0)
Size	1.48 4.37 1.48
SizeScale	1
Translation	0 2.19 0

version_2_with_code
- code
- data

Name	Size	Type
code		
data		
bad-chess-project	134 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr

3.2. PUSH THE CHESS PIECE USING PHYSICS (CONTINUATION 1)



SceneChessBoard1 (TCastleScene)
Behaviors
MeshCollider1 (TCastleMeshCollider)
RigidBody2 (TCastleRigidBody)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
SceneBlackKing1 (TCastleScene)
Behaviors
BoxCollider1 (TCastleBoxCollider)
RigidBody1 (TCastleRigidBody)
SceneBlackKnight1 (TCastleScene)

To push it using physics, we want to use the `ApplyImpulse` method of the `TCastleRigidBody` component associated with the chess piece.

- You can get the `TCastleRigidBody` component using the `SceneBlackKing1.` FindBehavior(`TCastleRigidBody`) method, as shown below.

  Alternatively, you could also declare and access `RigidBody1: TCastleRigidBody` reference in the published section of your view. We don't show this approach here, just because using the `FindBehavior` seems more educational at this point, i.e. you will find the `FindBehavior` useful in more situations.

- The ApplyImpulse method takes two parameters: the direction of the impulse (*as* `TVector3`; *length of this vector determines the impulse strength*) and the position from which the impulse comes (it is simplest to just use the chess piece position here).

In the end, this is the modified version of `TViewMain`. Press that you should use:

```
function TViewMain.Press(const Event: TInputPressRelease): Boolean;
var
 MyBody: TCastleRigidBody;
begin
 Result := inherited;
 if Result then Exit; // allow the ancestor to handle keys

 if Event.IsKey(keyX) then
 begin
  MyBody := SceneBlackKing1.FindBehavior(TCastleRigidBody) as
TCastleRigidBody;
  MyBody.ApplyImpulse(Vector3(0, 10, 0), SceneBlackKing1.
WorldTranslation);
  Exit(true); // key was handled
 end;
end;
```

3.2. PUSH THE CHESS PIECE USING PHYSICS (CONTINUATION 1)

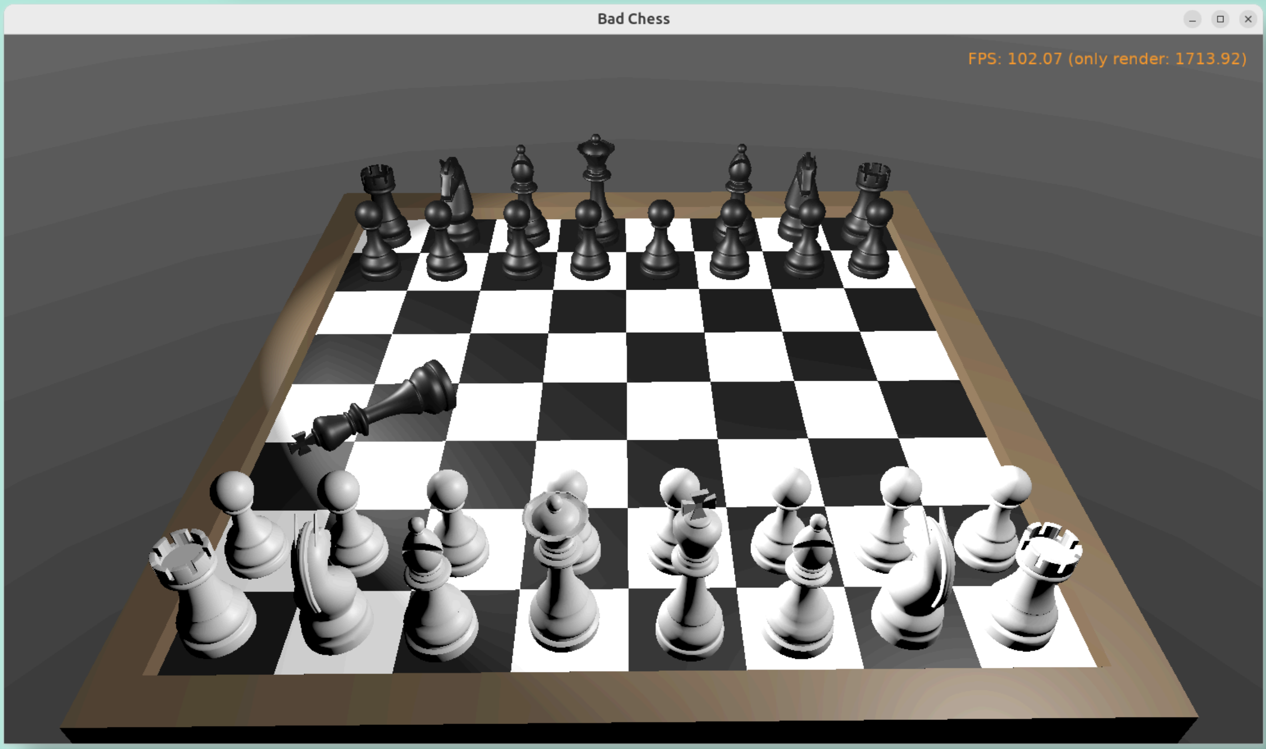
On the last page (*See page 6 or page 24 of this article*)
we use the direction `Vector3(0, 10, 0)` which means "**up, with strength 10**".

You can experiment with different directions and strengths. If we'd like to push the chess piece horizontally we would use a direction with `non-zero X and/or Z values`, and `leave Y axis zero.`

To the uses clause, add also **CastleTransform unit**, to have `TCastleRigidBody` class defined.

As usual, run the game and test.
Pressing X should now bump the chess piece up.

You can press X repeatedly, even when the chess piece is already in the air.
As you can see in the code - we don't secure from it, so we allow to push an object that is already flying.
We will not cover it in this exercise, but you could use `MyBody.PhysicsRayCast` to cast a ray with `direction Vector3(0, -1, 0)` and see whether the chess piece is already in the air.



❹ MAKE CODE AWARE "WHAT IS A CHESS PIECE" USING BEHAVIORS

To implement our desired logic, the code has to somehow know "what is a chess piece". So far, our 3D world is a collection of `TCastleScene` components, but it does not give us enough information to distinguish between chess pieces and other objects (`like a chessboard`).
We want to do something crazy, but we don't want to flip the chessboard! At least not this time

To "mark" that the given `TCastleScene` component is a chess pieces we will invent a new class called `TChessPieceBehavior` descending from the `TCastleBehavior` class.
We will then attach instances of this class to the `TCastleScene` components that represent chess pieces. In the future this class can have more fields (***holding information specific to this chess piece***) and methods. For start, the mere existence of `TCastleBehavior` instance attached to a scene indicates "this is a chess piece".

To know more about how our behaviors work, see `https://castle-engine.io/behaviors` for documentation and examples. You can also create a new project from the "3D FPS Game" template and see how the `TEnemy` class (***descendant of*** `TCastleBehavior`) is defined and used. The behaviors are a very flexible concept to add information and mechanics to your world and we advise to use them in many situations.

There's really nothing difficult about our initial `TChessPieceBehavior` definition.
It is almost an empty class. I decided to only add there a `Boolean` field that says whether the chess piece is white or black:

```
type
 TChessPieceBehavior = class(TCastleBehavior)
 public
  Black: Boolean;
 end;
```

You can declare it at the beginning of the interface section of `unit GameViewMain`.
Though larger behavior classes may deserve to be placed in their own units.

How to attach the behavior instances to the scenes?

❶ You could do this visually, by registering the `TChessPieceBehavior` class in the Castle Game Engine editor.

This is a very powerful method as it allows to visually add and configure the behavior properties.
See the `https://castle-engine.io/custom_components` for description how to use this.

❷ Or you can do it from code. In this article, I decided to go with this approach.

This is a bit easier if you have to effectively attach the behavior 32 times, to all the chess pieces, and there's no need to specifically configure the initial state of the behavior.
Clicking 32 times "Add Behavior" would be a bit tiresome and also unnecessary in our simple case (***for this demo, all chess pieces really work the same***), so let's instead utilize code to easily initialize the chess pieces.

To attach a behavior to our **SceneBlackKing1**, we would just create the instance of `TChessPieceBehavior` in our view's `Start` method, and add using `SceneBlackKing1.AddBehavior`. Like this:

```
procedure TViewMain.Start;
var
 ChessPiece: TChessPieceBehavior;
begin
 inherited;
 ChessPiece := TChessPieceBehavior.Create(FreeAtStop);
 ChessPiece.Black := true;
 SceneBlackKing1.AddBehavior(ChessPiece);
end;
```

❹ MAKE CODE AWARE "WHAT IS A CHESS PIECE" USING BEHAVIORS (CONTINUATION 1)

But this is not good enough for our application. Above we added `TChessPieceBehavior` to only one chess piece. We want to add it to all 32 the chess pieces.
How to do it easily?

We need to somehow iterate over all the chess pieces.
And to set the `Black boolean` field, we also should somehow know whether this is `black` or `white` piece. There are multiple solutions:

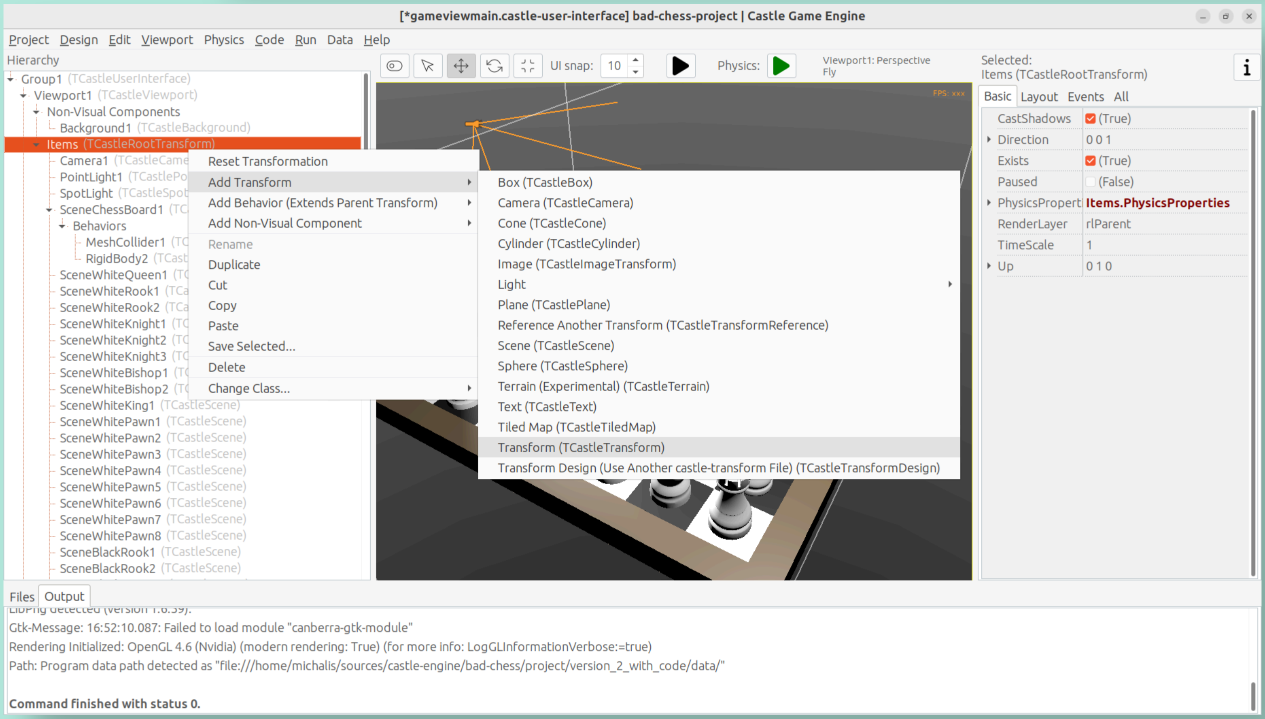
❶ We could assume that all chess pieces have names like `SceneWhiteXxx` or `SceneBlackXxx`. Then we can iterate over `Viewport1.Items` children, and check if their Name starts with given prefix.

❷ Or we could look at `Tag` value of scenes, and have a convention e.g. that `Tag = 1` means black chess piece, `Tag = 2` means white chess piece, and other tags (`Tag = 0` *is default, in particular*) means that this is not a chess piece.

❸ We could also introduce additional transformation components that group black chess pieces separately from white chess pieces and separately from other stuff (*like a chessboard*).

I decided to go with the latter approach, as introduction of "additional `TCastleTransform` components to group existing ones" is a powerful mechanism in many other situations. E.g. you can then easily hide or show a given group (*using* `TCastleTransform.Exists`) property.

To make this happen, right-click on `Viewport1.Items`, and choose from the context menu "**Add Transform → Transform (TCastleTransform)**".

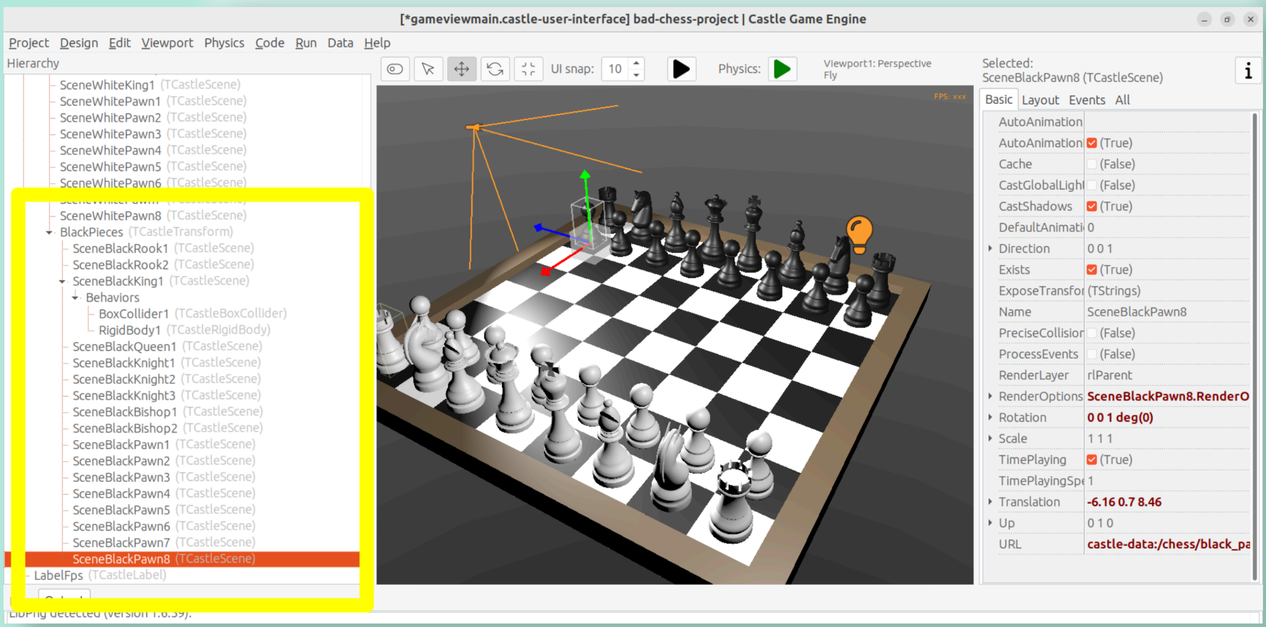

[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCame
PointLight1 (TCastlePo
SpotLight (TCastleSpot
SceneChessBoard1 (TC
Behaviors
MeshCollider1 (TC
RigidBody2 (TCast
SceneWhiteQueen1 (TC
SceneWhiteRook1 (TCa
SceneWhiteRook2 (TCa
SceneWhiteKnight1 (TC
SceneWhiteKnight2 (TC
SceneWhiteKnight3 (TC
SceneWhiteBishop1 (TC
SceneWhiteBishop2 (TC
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
UI snap: 10
Physics:
Viewport1: Perspective
Fly
Reset Transformation
Add Transform
Add Behavior (Extends Parent Transform)
Add Non-Visual Component
Rename
Duplicate
Cut
Copy
Paste
Save Selected...
Delete
Change Class...
Box (TCastleBox)
Camera (TCastleCamera)
Cone (TCastleCone)
Cylinder (TCastleCylinder)
Image (TCastleImageTransform)
Light
Plane (TCastlePlane)
Reference Another Transform (TCastleTransformReference)
Scene (TCastleScene)
Sphere (TCastleSphere)
Terrain (Experimental) (TCastleTerrain)
Text (TCastleText)
Tiled Map (TCastleTiledMap)
Transform (TCastleTransform)
Transform Design (Use Another castle-transform File) (TCastleTransformDesign)
Selected:
Items (TCastleRootTransform)
Basic Layout Events All
Files Output

CastShadows	✓ (True)
Direction	0 0 1
Exists	✓ (True)
Paused	(False)
PhysicsPropert	Items.PhysicsProperties
RenderLayer	rlParent
TimeScale	1
Up	0 1 0

```
LibPng detected (version 1.6.39).
Gtk-Message: 16:52:10.087: Failed to load module "canberra-gtk-module"
Rendering Initialized: OpenGL 4.6 (Nvidia) (modern rendering: True) (for more info: LogGLInformationVerbose:=true)
Path: Program data path detected as "file:///home/michalis/sources/castle-engine/bad-chess/project/version_2_with_code/data/"

Command finished with status 0.
```

❹ MAKE CODE AWARE "WHAT IS A CHESS PIECE" USING BEHAVIORS (CONTINUATION 2)

Name this new component `BlackPieces`.
Then drag-and-drop in the editor hierarchy all the black chess pieces (`SceneBlackXxx components`) to be children of `BlackPieces`. You can easily select all 16 scenes representing black pieces in the hierarchy by holding the **Shift** key and then drag-and-drop them all at once into `BlackPieces`.
The end result should look like this in the hierarchy:


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
Basic Layout Events All

```
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
BlackPieces (TCastleTransform)
  SceneBlackRook1 (TCastleScene)
  SceneBlackRook2 (TCastleScene)
  SceneBlackKing1 (TCastleScene)
    Behaviors
      BoxCollider1 (TCastleBoxCollider)
      RigidBody1 (TCastleRigidBody)
  SceneBlackQueen1 (TCastleScene)
  SceneBlackKnight1 (TCastleScene)
  SceneBlackKnight2 (TCastleScene)
  SceneBlackKnight3 (TCastleScene)
  SceneBlackBishop1 (TCastleScene)
  SceneBlackBishop2 (TCastleScene)
  SceneBlackPawn1 (TCastleScene)
  SceneBlackPawn2 (TCastleScene)
  SceneBlackPawn3 (TCastleScene)
  SceneBlackPawn4 (TCastleScene)
  SceneBlackPawn5 (TCastleScene)
  SceneBlackPawn6 (TCastleScene)
  SceneBlackPawn7 (TCastleScene)
  SceneBlackPawn8 (TCastleScene)
LabelFps (TCastleLabel)
```

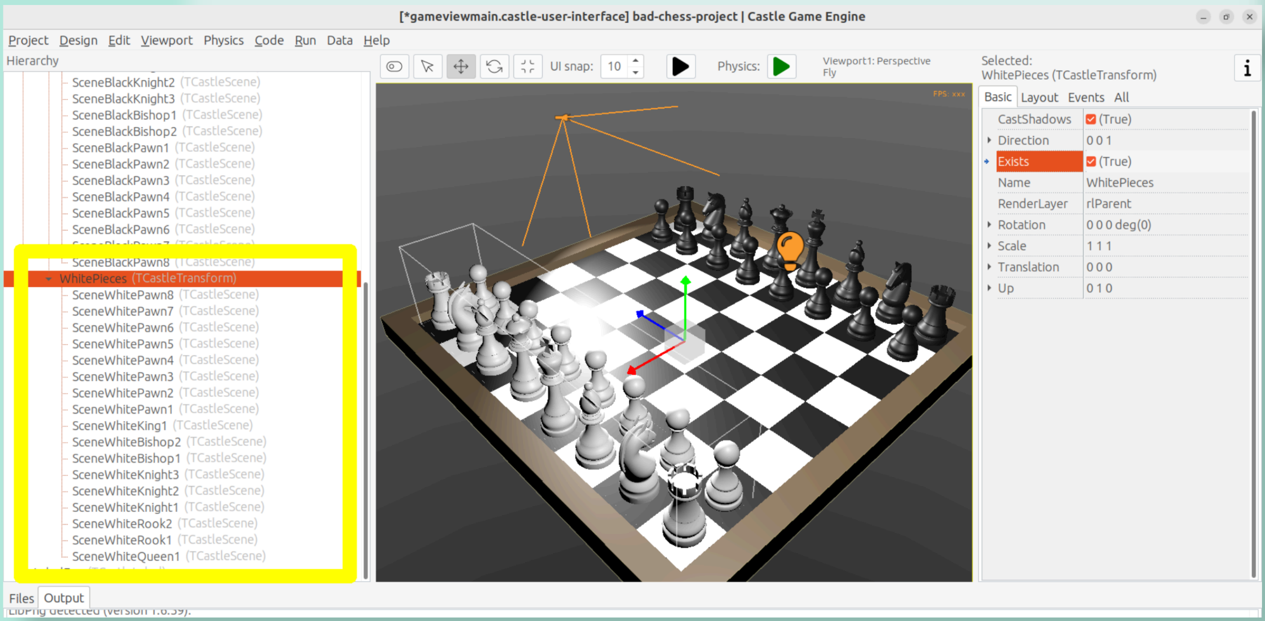
Selected:
SceneBlackPawn8 (TCastleScene)

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLight	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	0 0 1
Exists	✓ (True)
ExposeTransfo	(TStrings)
Name	SceneBlackPawn8
PreciseCollisior	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneBlackPawn8.RenderO
Rotation	0 0 1 deg(0)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSpe	1
Translation	-6.16 0.7 8.46
Up	0 1 0
URL	castle-data:/chess/black_pa

```
LibPng detected (version 1.6.39).
```

Don't worry that only the `SceneBlackKing1` has the physics components.
We will set the physics components using code soon too.

Now repeat the process to add a `WhitePieces` group.


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
Basic Layout Events All
Files Output

```
  SceneBlackKnight2 (TCastleScene)
  SceneBlackKnight3 (TCastleScene)
  SceneBlackBishop1 (TCastleScene)
  SceneBlackBishop2 (TCastleScene)
  SceneBlackPawn1 (TCastleScene)
  SceneBlackPawn2 (TCastleScene)
  SceneBlackPawn3 (TCastleScene)
  SceneBlackPawn4 (TCastleScene)
  SceneBlackPawn5 (TCastleScene)
  SceneBlackPawn6 (TCastleScene)
  SceneBlackPawn8 (TCastleScene)
WhitePieces (TCastleTransform)
  SceneWhitePawn8 (TCastleScene)
  SceneWhitePawn7 (TCastleScene)
  SceneWhitePawn6 (TCastleScene)
  SceneWhitePawn5 (TCastleScene)
  SceneWhitePawn4 (TCastleScene)
  SceneWhitePawn3 (TCastleScene)
  SceneWhitePawn2 (TCastleScene)
  SceneWhitePawn1 (TCastleScene)
  SceneWhiteKing1 (TCastleScene)
  SceneWhiteBishop2 (TCastleScene)
  SceneWhiteBishop1 (TCastleScene)
  SceneWhiteKnight3 (TCastleScene)
  SceneWhiteKnight2 (TCastleScene)
  SceneWhiteKnight1 (TCastleScene)
  SceneWhiteRook2 (TCastleScene)
  SceneWhiteRook1 (TCastleScene)
  SceneWhiteQueen1 (TCastleScene)
```

Selected:
WhitePieces (TCastleTransform)

CastShadows	✓ (True)
Direction	0 0 1
Exists	✓ (True)
Name	WhitePieces
RenderLayer	rlParent
Rotation	0 0 0 deg(0)
Scale	1 1 1
Translation	0 0 0
Up	0 1 0

```
LibPng detected (version 1.6.39).
```

❹ MAKE CODE AWARE "WHAT IS A CHESS PIECE" USING BEHAVIORS (CONTINUATION 3)

This preparation in the editor makes our code task easier. Add to the published section of `TViewMain` declaration of `BlackPieces` and `WhitePieces` fields, of type `TCastleTransform`:

```
 TViewMain = class(TCastleView)
published
  ... // keep other fields too
  BlackPieces, WhitePieces: TCastleTransform;
```

Now iterate over the 2 chess pieces' groups in the `Start` method:

```
procedure TViewMain.Start;

  procedure ConfigureChessPiece(const Child: TCastleTransform; const Black: Boolean);
  var
    ChessPiece: TChessPieceBehavior;
  begin
    ChessPiece := TChessPieceBehavior.Create(FreeAtStop);
    ChessPiece.Black := true;
    Child.AddBehavior(ChessPiece);
  end;

var
  Child: TCastleTransform;
begin
  inherited;
  for Child in BlackPieces do
    ConfigureChessPiece(Child, true);
  for Child in WhitePieces do
    ConfigureChessPiece(Child, false);
end;
```

It seems prudent to add basic "sanity check" at this point. Let's log the number of chess pieces each side has. Add the following code and the end of the `Start` method:

```
WritelnLog('Configured %d black and %d white chess pieces', [
  BlackPieces.Count,
  WhitePieces.Count
]);
```

To make `WritelnLog` available, add `CastleLog` unit to the uses clause.
Now when you run the game, you should see a log

```
Configured 16 black and 16 white chess pieces
```

On my first run, I actually saw that I have 17 chess pieces on each side by accident.
I mistakenly added 3 knights instead of 2 (*one knight was at exactly the same position as another, so it wasn't obvious*).
I have removed the excessive knight pieces thanks to this log. Detecting such mistakes is exactly the reason why we add logs and test - so I encourage you to do it too.

While we're at it, we can also use this opportunity to make sure all chess pieces have physics components (`TCastleRigidBody` and `TCastleBoxCollider`).
So you don't need to manually add them all. This is a reasonable approach if the components don't need any manual adjustment per-chess-piece.

❹ MAKE CODE AWARE "WHAT IS A CHESS PIECE" USING BEHAVIORS (CONTINUATION 4)

To do this, extend our `ConfigureChessPiece` method:

```
procedure ConfigureChessPiece(const Child: TCastleTransform; const Black: Boolean);
 begin
  ... // keep previous code too
  if Child.FindBehavior(TCastleRigidBody) = nil then
   Child.AddBehavior(TCastleRigidBody.Create(FreeAtStop));
  if Child.FindBehavior(TCastleCollider) = nil then
   Child.AddBehavior(TCastleBoxCollider.Create(FreeAtStop));
 end;
```

As you see above, this approach is quite direct: if you don't have the necessary component, just add it. We don't bother to configure any property on the new `TCastleRigidBody` and `TCastleBoxCollider` instances, as their defaults are good for our purpose.

This was all a good "ground work" for the remaining article part. Nothing functionally new has actually happened in our game, you should run it and see that... nothing changed. All 32 chess pieces just stand still, at the beginning.

❺ SELECTING 3D OBJECT USING THE MOUSE

5.1. HIGHLIGHT THE CHESS PIECE UNDER MOUSE AND ALLOW SELECTING IT

To implement the real interaction, we want to allow user to choose which chess piece to flick using the mouse. **Castle Game Engine** provides a ready function that tells you what is being indicated by the current mouse (*or last touch, on mobile*) position.
This is the `TCastleViewport.TransformUnderMouse` function.

For start, make sure to declare the viewport instance in the published section of class `TViewMain`, like this:

```
MainViewport: TCastleViewport;
```

Match the name of your viewport in the design.
Add unit `CastleViewport` to the uses clause to make type `TCastleViewport` known.

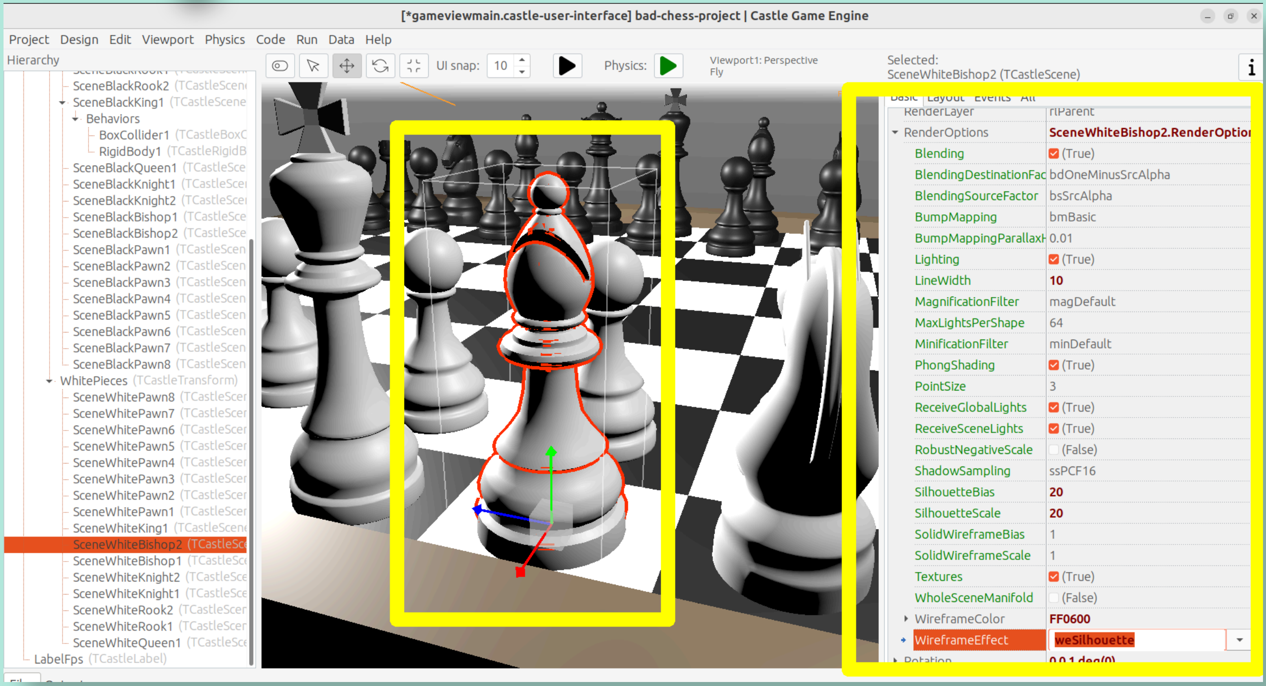
Let's utilize it to highlight the current chess piece at the mouse position.
We can just keep checking the MainViewport.TransformUnderMouse value in each Update call.

NOTE: Alternatively, we could check `MainViewport.TransformUnderMouse` in each Motion call, that occurs only when mouse (*or touch*) position changes. But doing it in Update is a bit better: as we use physics, some chess pieces may still be moving due to physics, so the chess piece under the mouse may change even if the mouse position doesn't change.

To actually show the highlight, we will use a ready effect available for every `TCastleScene` that can be activated by setting `MyScene.RenderOptions.WireframeEffect` to something else than `weNormal`.
This is the simplest way to show the highlight (*we discuss other ways in later section*).

Before we jump into code, I encourage to experiment with perfect settings of **RenderOptions** for highlight in the editor.
Just edit any chosen chess piece, until it seems to have a pretty highlight, and remember the chosen options.
The most useful properties to adjust are `WireframeEffect, WireframeColor, LineWidth, SilhouetteBias, SilhouetteScale`.
You can see them emphasized on the next page - editor shows properties which have non-default values using the bold font.

5.1. HIGHLIGHT THE CHESS PIECE UNDER MOUSE AND ALLOW SELECTING IT (CONTINUATION 1)


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
Selected:
SceneWhiteBishop2 (TCastleScene)
Basic Layout Events All

SceneBlackRook2 (TCastleScene)
SceneBlackKing1 (TCastleScene)
Behaviors
BoxCollider1 (TCastleBoxC
RigidBody1 (TCastleRigidB
SceneBlackQueen1 (TCastleSce
SceneBlackKnight1 (TCastleSce
SceneBlackKnight2 (TCastleSce
SceneBlackBishop1 (TCastleSce
SceneBlackBishop2 (TCastleSce
SceneBlackPawn1 (TCastleScen
SceneBlackPawn2 (TCastleScen
SceneBlackPawn3 (TCastleScen
SceneBlackPawn4 (TCastleScen
SceneBlackPawn5 (TCastleScen
SceneBlackPawn6 (TCastleScen
SceneBlackPawn7 (TCastleScen
SceneBlackPawn8 (TCastleScen
WhitePieces (TCastleTransform)
SceneWhitePawn8 (TCastleScer
SceneWhitePawn7 (TCastleScer
SceneWhitePawn6 (TCastleScer
SceneWhitePawn5 (TCastleScer
SceneWhitePawn4 (TCastleScer
SceneWhitePawn3 (TCastleScer
SceneWhitePawn2 (TCastleScer
SceneWhitePawn1 (TCastleScer
SceneWhiteKing1 (TCastleScene
SceneWhiteBishop2 (TCastleSc
SceneWhiteBishop1 (TCastleSc
SceneWhiteKnight2 (TCastleSce
SceneWhiteKnight1 (TCastleSce
SceneWhiteRook2 (TCastleScer
SceneWhiteRook1 (TCastleScer
SceneWhiteQueen1 (TCastleSce
LabelFps (TCastleLabel)

RenderLayer	rlParent
RenderOptions	SceneWhiteBishop2.RenderOptio
Blending	✓ (True)
BlendingDestinationFac	bdOneMinusSrcAlpha
BlendingSourceFactor	bsSrcAlpha
BumpMapping	bmBasic
BumpMappingParallaxH	0.01
Lighting	✓ (True)
LineWidth	10
MagnificationFilter	magDefault
MaxLightsPerShape	64
MinificationFilter	minDefault
PhongShading	✓ (True)
PointSize	3
ReceiveGlobalLights	✓ (True)
ReceiveSceneLights	✓ (True)
RobustNegativeScale	(False)
ShadowSampling	ssPCF16
SilhouetteBias	20
SilhouetteScale	20
SolidWireframeBias	1
SolidWireframeScale	1
Textures	✓ (True)
WholeSceneManifold	(False)
WireframeColor	FF0600
WireframeEffect	weSilhouette
Rotation	0 0 1 deg(0)

I decided to show the currently highlighted (*at mouse position*) chess piece with a light-blue wireframe. This chess piece is also set as the value of private field `ChessPieceHover`.

Moreover, once user clicks with mouse (*we can detect it in* `Press`) the chess piece is considered selected and gets a yellow highlight.
This chess piece is set as `ChessPieceSelected` value.

Remembering the `ChessPieceHover` and `ChessPieceSelected` values is useful for a few things. For one thing, we can later disable the effect (*when the piece is no longer highlighted or selected*). And it will allow to flick the `ChessPieceSelected` in the next sections.

We could store them as references to `TCastleScene` or `TChessPieceBehavior`.
That is, we could declare:

Either `ChessPieceHover, ChessPieceSelected: TChessPieceBehavior;`...
...or `ChessPieceHover, ChessPieceSelected: TCastleScene;`

Both declarations would be good for our application.
That is, we have to choose one or the other as it will imply a bit different code, but the differences are really minor. In the end, we can always get `TChessPieceBehavior` instance from a corresponding `TCastleScene` (*if we know it is a chess piece*) and we can get `TCastleScene` from a `TChessPieceBehavior`.

To get `TChessPieceBehavior` from the corresponding `TCastleScene` you would do:

```
 var
  MyBehavior: TChessPieceBehavior;
  MyScene: TCastleScene;
begin
  ...
  MyBehavior := MyScene.FindBehavior(TChessPieceBehavior) as
                                                TChessPieceBehavior;
```

To get `TCastleScene` from corresponding `TChessPieceBehavior` you would do:

5.1. HIGHLIGHT THE CHESS PIECE UNDER MOUSE AND ALLOW SELECTING IT (CONTINUATION 2)

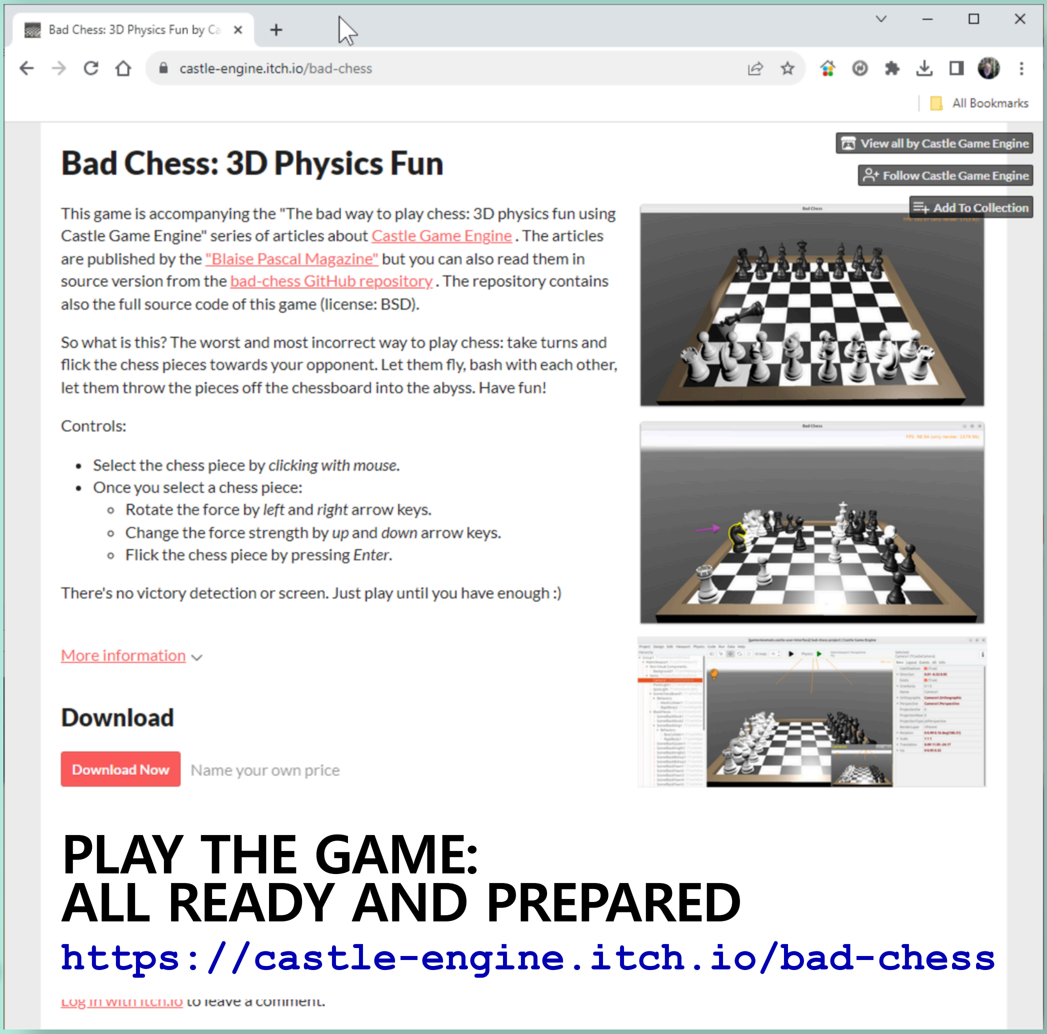
```
var
 MyBehavior: TChessPieceBehavior;
 MyScene: TCastleScene;
begin
 ...
 MyScene := MyBehavior.Parent as TCastleScene;
```

I decided to declare them as `TChessPieceBehavior`. If you want to follow my approach exactly, add this to the private section of class `TViewMain`:

```
 ChessPieceHover, ChessPieceSelected: TChessPieceBehavior;
{ Turn on / off the highlight effect, depending on whether
  Behavior equals ChessPieceHover, ChessPieceSelected or none of them.
  This accepts (and ignores) Behavior = nil value. }
procedure ConfigureEffect(const Behavior: TChessPieceBehavior);
```

Then add CastleColors unit to the uses clause (*of interface or implementation of unit* `GameViewMain`*, doesn't matter in this case*) to define **HexToColorRGB** utility.

Finally this is the code of new `Update`, `Press` and helper `ConfigureEffect` methods:


Bad Chess: 3D Physics Fun by Ca
castle-engine.itch.io/bad-chess
All Bookmarks
View all by Castle Game Engine
Follow Castle Game Engine
Add To Collection

Bad Chess: 3D Physics Fun

This game is accompanying the "The bad way to play chess: 3D physics fun using Castle Game Engine" series of articles about Castle Game Engine. The articles are published by the "Blaise Pascal Magazine" but you can also read them in source version from the bad-chess GitHub repository. The repository contains also the full source code of this game (license: BSD).

So what is this? The worst and most incorrect way to play chess: take turns and flick the chess pieces towards your opponent. Let them fly, bash with each other, let them throw the pieces off the chessboard into the abyss. Have fun!

Controls:

- Select the chess piece by *clicking with mouse*.
- Once you select a chess piece:
  - Rotate the force by *left* and *right* arrow keys.
  - Change the force strength by *up* and *down* arrow keys.
  - Flick the chess piece by pressing *Enter*.

There's no victory detection or screen. Just play until you have enough :)

More information

Download

Download Now   Name your own price

PLAY THE GAME:
ALL READY AND PREPARED
https://castle-engine.itch.io/bad-chess

Log in with itch.io to leave a comment.

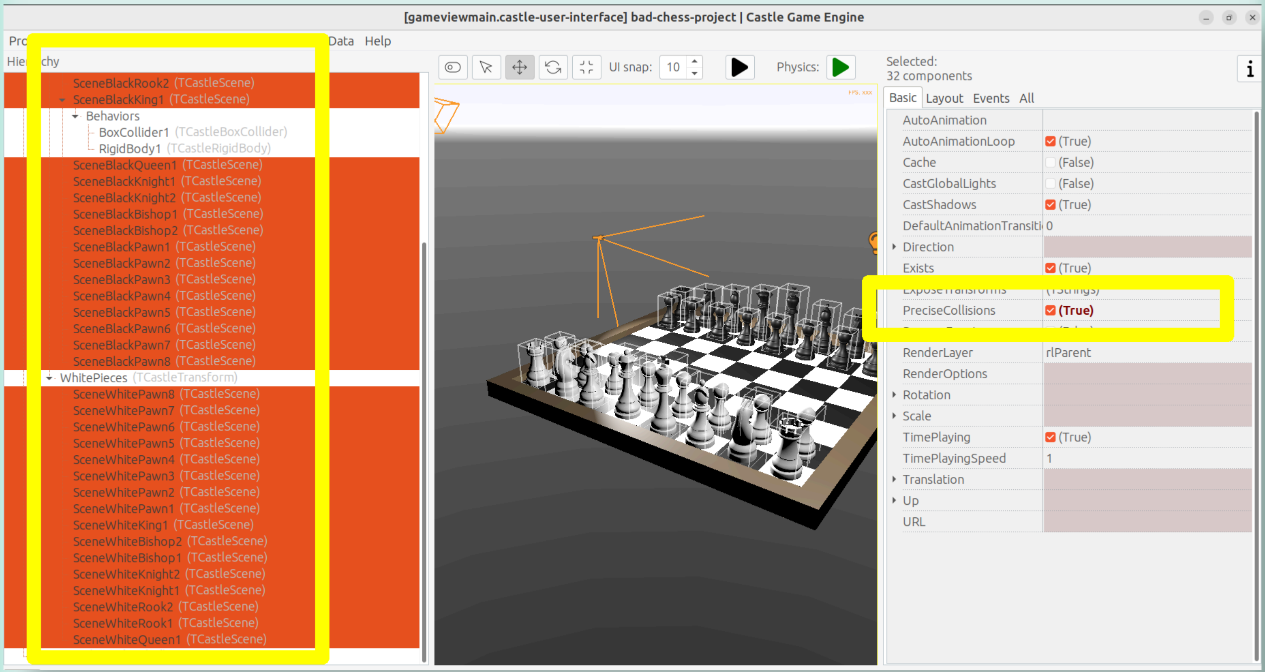
5.1. HIGHLIGHT THE CHESS PIECE UNDER MOUSE AND ALLOW SELECTING IT (CONTINUATION 3)

```
procedure TViewMain.ConfigureEffect(const Behavior: TChessPieceBehavior);
var   Scene: TCastleScene;
begin
  if Behavior = nil then Exit;
  { Behavior can be attached to any TCastleTransform.
   But in our case, we know TChessPieceBehavior is attached to TCastleScene. }
  Scene := Behavior.Parent as TCastleScene;
  if (Behavior = ChessPieceHover) or
     (Behavior = ChessPieceSelected) then
  begin
    Scene.RenderOptions.WireframeEffect := weSilhouette;
    if Behavior = ChessPieceSelected then
      Scene.RenderOptions.WireframeColor := HexToColorRGB('FFEB00')
    else
      Scene.RenderOptions.WireframeColor := HexToColorRGB('5455FF');
    Scene.RenderOptions.LineWidth := 10;
    Scene.RenderOptions.SilhouetteBias := 20;
    Scene.RenderOptions.SilhouetteScale := 20;
  end else
  begin
    Scene.RenderOptions.WireframeEffect := weNormal;
  end;
end;

procedure TViewMain.Update(const SecondsPassed: Single; var HandleInput: Boolean);
var OldHover: TChessPieceBehavior;
begin
  inherited;

  LabelFps.Caption := 'FPS: ' + Container.Fps.ToString;
  OldHover := ChessPieceHover;

  if MainViewport.TransformUnderMouse <> nil then
  begin
    ChessPieceHover := MainViewport.TransformUnderMouse. FindBehavior(TChessPieceBehavior)
                                                       as TChessPieceBehavior;
  end else  ChessPieceHover := nil;

  if OldHover <> ChessPieceHover then
  begin
    ConfigureEffect(OldHover);
    ConfigureEffect(ChessPieceHover);
  end;
end;

function TViewMain.Press(const Event: TInputPressRelease): Boolean;
var  MyBody: TCastleRigidBody; OldSelected: TChessPieceBehavior;
begin
  Result := inherited;
  if Result then Exit; // allow the ancestor to handle keys

  // ... if you want, keep here the handling of keyX from previous exercise

  if Event.IsMouseButton(buttonLeft) then
  begin
    OldSelected := ChessPieceSelected;
    if (ChessPieceHover <> nil) and
       (ChessPieceHover <> ChessPieceSelected) then
    begin
      ChessPieceSelected := ChessPieceHover;
      ConfigureEffect(OldSelected);
      ConfigureEffect(ChessPieceSelected);
    end;
    Exit(true); // mouse click was handled
  end;
end;
```

5.1. HIGHLIGHT THE CHESS PIECE UNDER MOUSE AND ALLOW SELECTING IT (CONTINUATION 4)

As always, remember to compile and run the code to make sure it works OK!

You will notice that `MainViewport.TransformUnderMouse` detects what is under the mouse, but treating each chess piece as a box. So the detection is visibly not accurate. To fix this, set `PreciseCollisions` to `true` on all the chess pieces. You can do this easily by selecting all chess pieces in editor using **Shift** or **Ctrl** and then toggling `PreciseCollisions` in the **Object Inspector.**



[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine

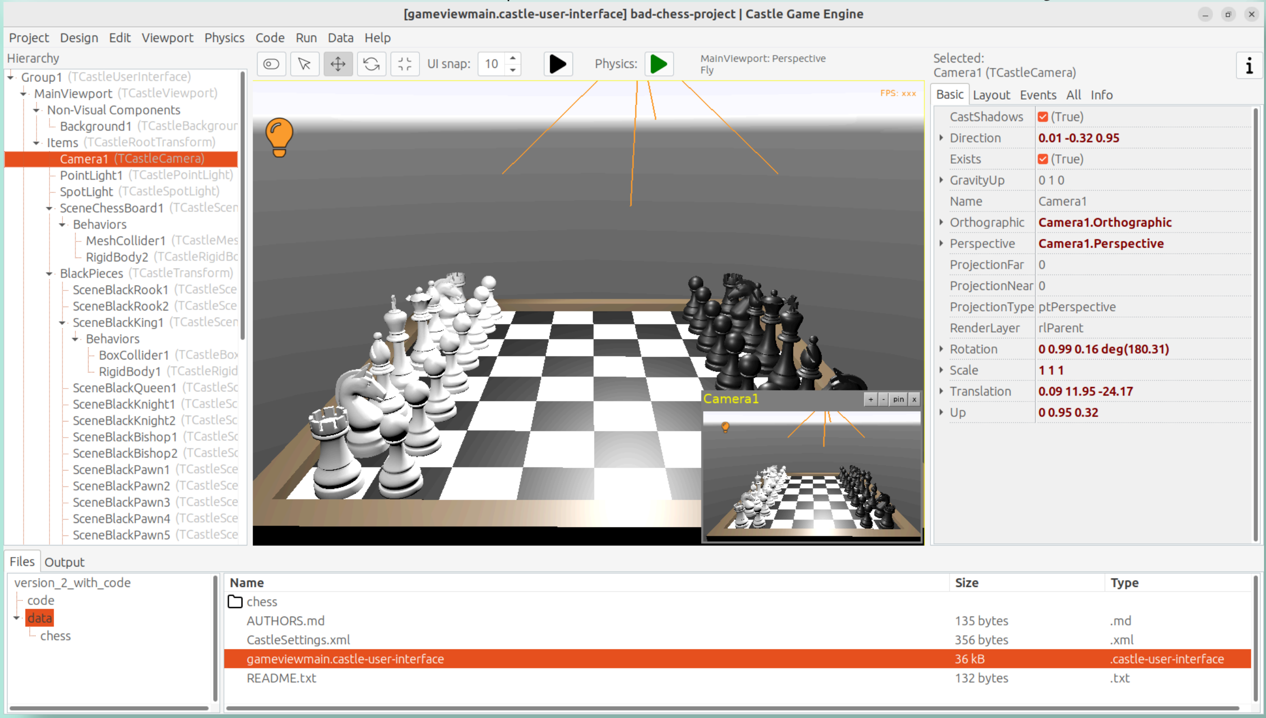
Hierarchy:
- SceneBlackRook2 (TCastleScene)
- SceneBlackKing1 (TCastleScene)
  - Behaviors
    - BoxCollider1 (TCastleBoxCollider)
    - RigidBody1 (TCastleRigidBody)
- SceneBlackQueen1 (TCastleScene)
- SceneBlackKnight1 (TCastleScene)
- SceneBlackKnight2 (TCastleScene)
- SceneBlackBishop1 (TCastleScene)
- SceneBlackBishop2 (TCastleScene)
- SceneBlackPawn1 (TCastleScene)
- SceneBlackPawn2 (TCastleScene)
- SceneBlackPawn3 (TCastleScene)
- SceneBlackPawn4 (TCastleScene)
- SceneBlackPawn5 (TCastleScene)
- SceneBlackPawn6 (TCastleScene)
- SceneBlackPawn7 (TCastleScene)
- SceneBlackPawn8 (TCastleScene)
- WhitePieces (TCastleTransform)
  - SceneWhitePawn8 (TCastleScene)
  - SceneWhitePawn7 (TCastleScene)
  - SceneWhitePawn6 (TCastleScene)
  - SceneWhitePawn5 (TCastleScene)
  - SceneWhitePawn4 (TCastleScene)
  - SceneWhitePawn3 (TCastleScene)
  - SceneWhitePawn2 (TCastleScene)
  - SceneWhitePawn1 (TCastleScene)
  - SceneWhiteKing1 (TCastleScene)
  - SceneWhiteBishop2 (TCastleScene)
  - SceneWhiteBishop1 (TCastleScene)
  - SceneWhiteKnight2 (TCastleScene)
  - SceneWhiteKnight1 (TCastleScene)
  - SceneWhiteRook2 (TCastleScene)
  - SceneWhiteRook1 (TCastleScene)
  - SceneWhiteQueen1 (TCastleScene)

UI snap: 10   Physics:

Selected:
32 components

Basic	
AutoAnimation	
AutoAnimationLoop	☑ (True)
Cache	☐ (False)
CastGlobalLights	☐ (False)
CastShadows	☑ (True)
DefaultAnimationTransiti	0
Direction	
Exists	☑ (True)
ExposeTransforms	(TStrings)
PreciseCollisions	☑ **(True)**
RenderLayer	rlParent
RenderOptions	
Rotation	
Scale	
TimePlaying	☑ (True)
TimePlayingSpeed	1
Translation	
Up	
URL	

I decided to move the camera at this point too (*to show both sides, black and white, from a side view*).



[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine

Project Design Edit Viewport Physics Code Run Data Help

Hierarchy:
- Group1 (TCastleUserInterface)
  - MainViewport (TCastleViewport)
    - Non-Visual Components
      - Background1 (TCastleBackgrour
    - Items (TCastleRootTransform)
      - Camera1 (TCastleCamera)
      - PointLight1 (TCastlePointLight)
      - SpotLight (TCastleSpotLight)
      - SceneChessBoard1 (TCastleScen
        - Behaviors
          - MeshCollider1 (TCastleMes
          - RigidBody2 (TCastleRigidBc
      - BlackPieces (TCastleTransform)
        - SceneBlackRook1 (TCastleSce
        - SceneBlackRook2 (TCastleSce
        - SceneBlackKing1 (TCastleScen
          - Behaviors
            - BoxCollider1 (TCastleBox
            - RigidBody1 (TCastleRigid
        - SceneBlackQueen1 (TCastleSc
        - SceneBlackKnight1 (TCastleSc
        - SceneBlackKnight2 (TCastleSc
        - SceneBlackBishop1 (TCastleSc
        - SceneBlackBishop2 (TCastleSc
        - SceneBlackPawn1 (TCastleSce
        - SceneBlackPawn2 (TCastleSce
        - SceneBlackPawn3 (TCastleSce
        - SceneBlackPawn4 (TCastleSce
        - SceneBlackPawn5 (TCastleSce

UI snap: 10   Physics: MainViewport: Perspective Fly

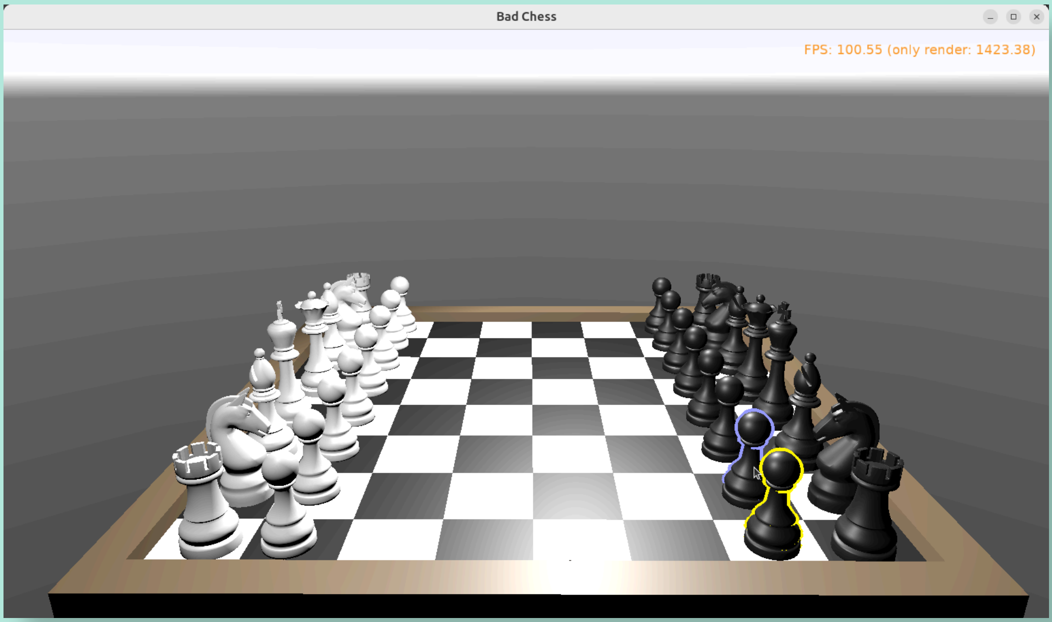
Selected:
Camera1 (TCastleCamera)

Basic	
CastShadows	☑ (True)
Direction	0.01 -0.32 0.95
Exists	☑ (True)
GravityUp	0 1 0
Name	Camera1
Orthographic	Camera1.Orthographic
Perspective	Camera1.Perspective
ProjectionFar	0
ProjectionNear	0
ProjectionType	ptPerspective
RenderLayer	rlParent
Rotation	0 0.99 0.16 deg(180.31)
Scale	1 1 1
Translation	0.09 11.95 -24.17
Up	0 0.95 0.32

Files Output

- version_2_with_code
  - code
  - data
    - chess

Name	Size	Type
chess		
AUTHORS.md	135 bytes	.md
CastleSettings.xml	356 bytes	.xml
gameviewmain.castle-user-interface	36 kB	.castle-user-interface
README.txt	132 bytes	.txt



5.2. SIDENOTE: OTHER WAYS TO SHOW A HIGHLIGHT

There are other ways to show the **highlighted** (**or selected**) chess piece.

Dynamically changing the material color. Do this by accessing an instance of `TPhysicalMaterialNode` within the scene's nodes (`TCastleScene.RootNode`) and changing the `TPhysicalMaterialNode.BaseColor`.
See e.g. engine example **`examples/viewport_and_scenes/collisions/`** that uses this.

Dynamically adding/removing a shader effect. This means adding `TEffectNode` and `TEffectPartNode` nodes to the scene and implementing the effect using **GLSL (OpenGL Shading Language)**. See e.g. engine example **`examples/viewport_and_scenes/shader_effects/`** that demonstrates this.

Adding a additional box that surrounds chosen object. The **CGE** editor itself uses this technique to show highlighted / selected 3D objects. Use `TDebugTransformBox` class to implement this easily.

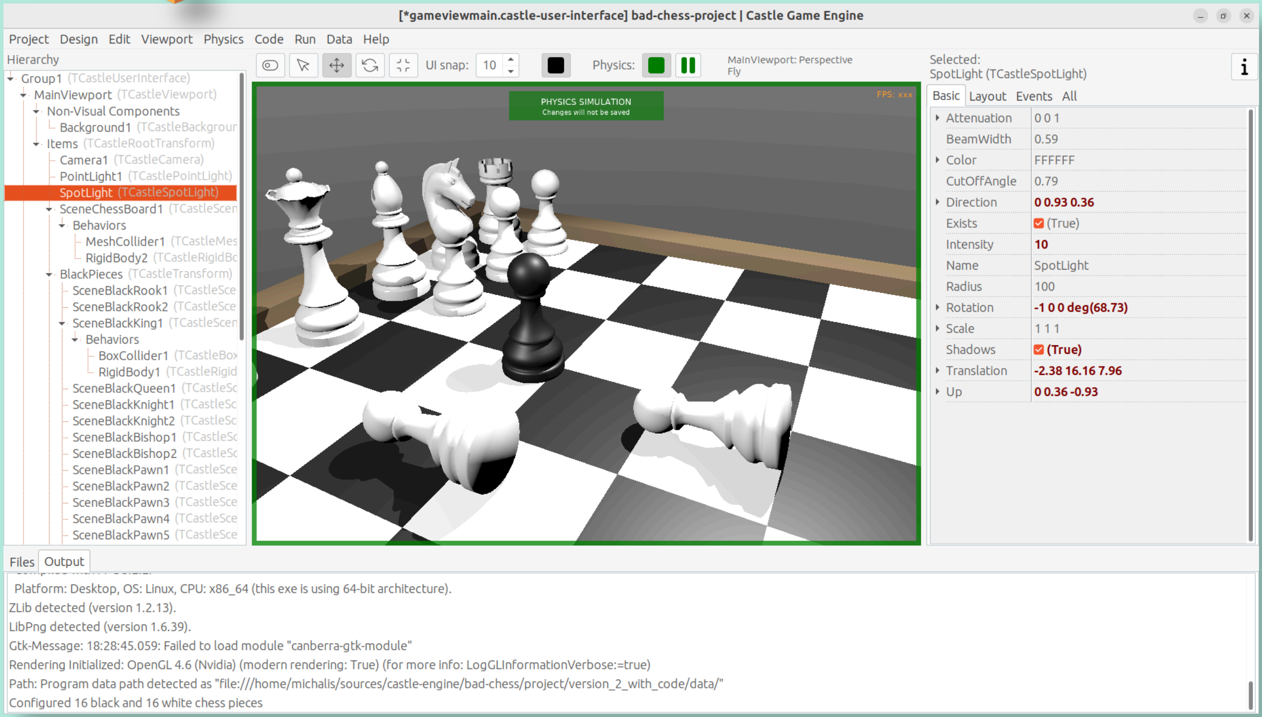
If you are curious, hopefully the above information and examples will point you in the right direction.

5.3. SIDENOTE: SHADOWS

I decided to activate shadows at this point. Just set `Shadows` to true on the main light source.
Moreover, set `RenderOptions.WholeSceneManifold` to true at the chess pieces.
This should make everything cast nice shadows. The shadows are dynamic which means that they will properly change when we will move the chess pieces.

See **`https://castle-engine.io/shadow_volumes`** for more information about shadows in **Castle Game Engine.**

5.3. SIDENOTE: SHADOWS (CONTINUATION)


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
UI snap: 10
Physics:
MainViewport: Perspective
Fly
PHYSICS SIMULATION
Changes will not be saved
FPS: xxx

Hierarchy

```
Group1 (TCastleUserInterface)
  MainViewport (TCastleViewport)
    Non-Visual Components
      Background1 (TCastleBackgrour
    Items (TCastleRootTransform)
      Camera1 (TCastleCamera)
      PointLight1 (TCastlePointLight)
      SpotLight (TCastleSpotLight)
      SceneChessBoard1 (TCastleScen
        Behaviors
          MeshCollider1 (TCastleMes
          RigidBody2 (TCastleRigidBc
      BlackPieces (TCastleTransform)
        SceneBlackRook1 (TCastleSce
        SceneBlackRook2 (TCastleSce
        SceneBlackKing1 (TCastleScen
          Behaviors
            BoxCollider1 (TCastleBox
            RigidBody1 (TCastleRigid
        SceneBlackQueen1 (TCastleSc
        SceneBlackKnight1 (TCastleSc
        SceneBlackKnight2 (TCastleSc
        SceneBlackBishop1 (TCastleSc
        SceneBlackBishop2 (TCastleSc
        SceneBlackPawn1 (TCastleSce
        SceneBlackPawn2 (TCastleSce
        SceneBlackPawn3 (TCastleSce
        SceneBlackPawn4 (TCastleSce
        SceneBlackPawn5 (TCastleSce
```

Selected:
SpotLight (TCastleSpotLight)

Basic Layout Events All

Attenuation	0 0 1
BeamWidth	0.59
Color	FFFFFF
CutOffAngle	0.79
Direction	**0 0.93 0.36**
Exists	✓ (True)
Intensity	**10**
Name	SpotLight
Radius	100
Rotation	**-1 0 0 deg(68.73)**
Scale	1 1 1
Shadows	✓ **(True)**
Translation	**-2.38 16.16 7.96**
Up	**0 0.36 -0.93**

Files Output

```
 Platform: Desktop, OS: Linux, CPU: x86_64 (this exe is using 64-bit architecture).
ZLib detected (version 1.2.13).
LibPng detected (version 1.6.39).
Gtk-Message: 18:28:45.059: Failed to load module "canberra-gtk-module"
Rendering Initialized: OpenGL 4.6 (Nvidia) (modern rendering: True) (for more info: LogGLInformationVerbose:=true)
Path: Program data path detected as "file:///home/michalis/sources/castle-engine/bad-chess/project/version_2_with_code/data/"
Configured 16 black and 16 white chess pieces
```

❻ LET USER CHOOSE THE ANGLE AND STRENGTH TO FLICK THE CHESS PIECE

Once the user has picked a chess piece, we want to allow configuring the direction and strength with which to flick the chosen object.
We already know that "flicking" the chess piece technically means "applying a physics force to the rigid body of a chosen chess piece". We have almost everything we need, but we need to allow user to choose the direction and strength of this force.

6.1. DESIGNING A 3D ARROW

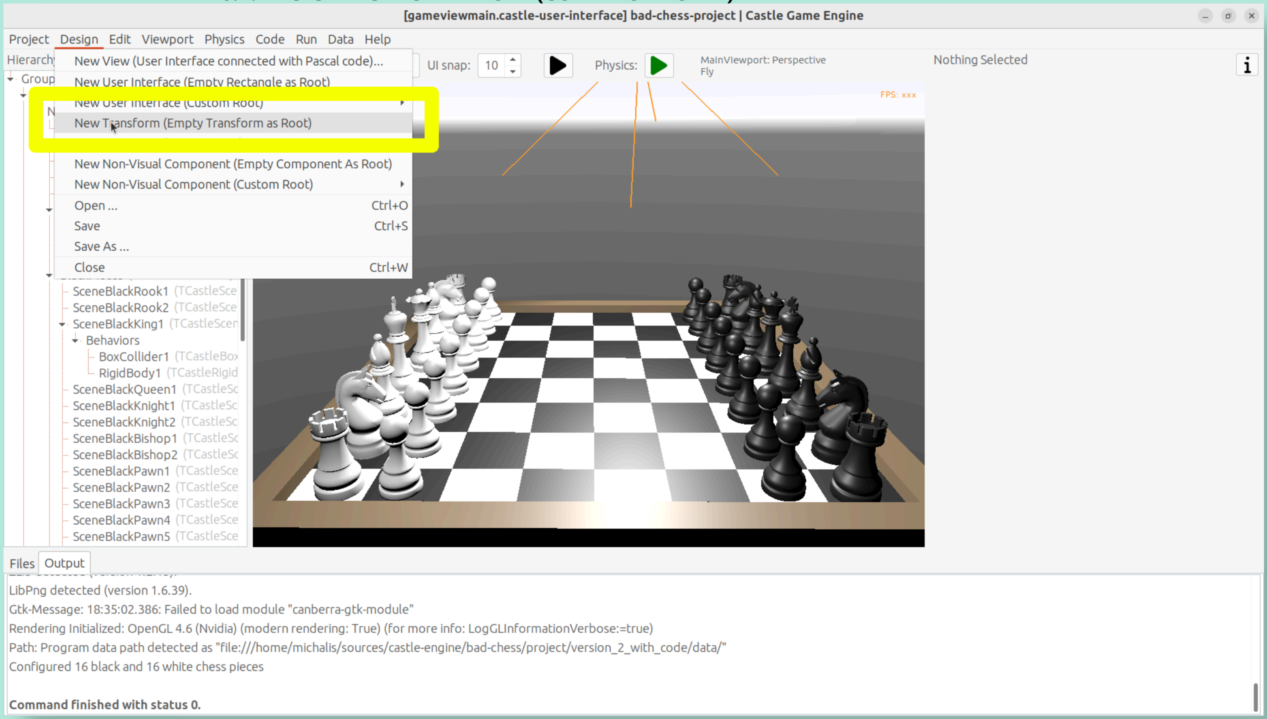
To visualize the desired force we will use a simple **3D arrow** model, that will be rotated and scaled accordingly.
While we could design such model in **Blender** or other **3D authoring** software, in this case it's easiest to just do it completely in the **Castle Game Engine** editor.
The arrow is a composition of two simple shapes: cone (*for the arrow tip*) and a cylinder.

Moreover let's design the arrow independently, as a separate design.
The new design will contain a hierarchy of components, with the root being `TCastleTransform`.
We will save it as a file `force_gizmo.castle-transform` in the project data subdirectory.
Then we will add it to the main design (*gameviewmain.castle-user-interface*), and toggle the existence, rotation and scale of the visualized force.

Using a separate design file for the **3D arrow**, while not strictly necessary in this case, is a powerful technique. When something is saved as a separate design file, you can reuse it freely, and instantiate it many times (*at* **design-time**, *or by* **dynamically spawning** *during the* **game run-time**).
This is e.g. how to have creatures in your game: 3D objects that share common logic and that can be spawned whenever needed.

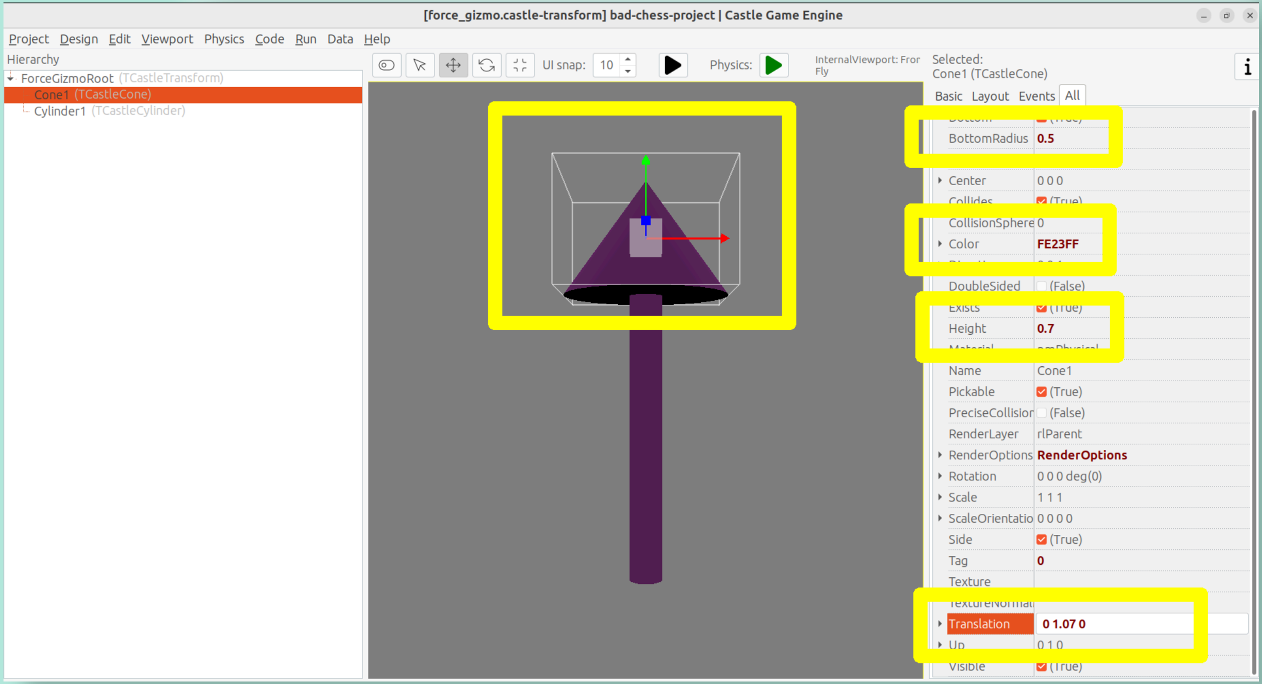
To start designing the arrow, choose editor menu item
 "**Design → New Transform (Empty Transform as Root)**".

6.1. DESIGNING A 3D ARROW (CONTINUATION 1)


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
New View (User Interface connected with Pascal code)...
New User Interface (Empty Rectangle as Root)
New User Interface (Custom Root)
New Transform (Empty Transform as Root)
New Non-Visual Component (Empty Component As Root)
New Non-Visual Component (Custom Root)
Open ... Ctrl+O
Save Ctrl+S
Save As ...
Close Ctrl+W
UI snap: 10
Physics:
MainViewport: Perspective Fly
Nothing Selected
FPS: xxx
SceneBlackRook1 (TCastleSce
SceneBlackRook2 (TCastleSce
SceneBlackKing1 (TCastleScen
Behaviors
BoxCollider1 (TCastleBox
RigidBody1 (TCastleRigid
SceneBlackQueen1 (TCastleSc
SceneBlackKnight1 (TCastleSc
SceneBlackKnight2 (TCastleSc
SceneBlackBishop1 (TCastleSc
SceneBlackBishop2 (TCastleSc
SceneBlackPawn1 (TCastleSce
SceneBlackPawn2 (TCastleSce
SceneBlackPawn3 (TCastleSce
SceneBlackPawn4 (TCastleSce
SceneBlackPawn5 (TCastleSce
Files Output

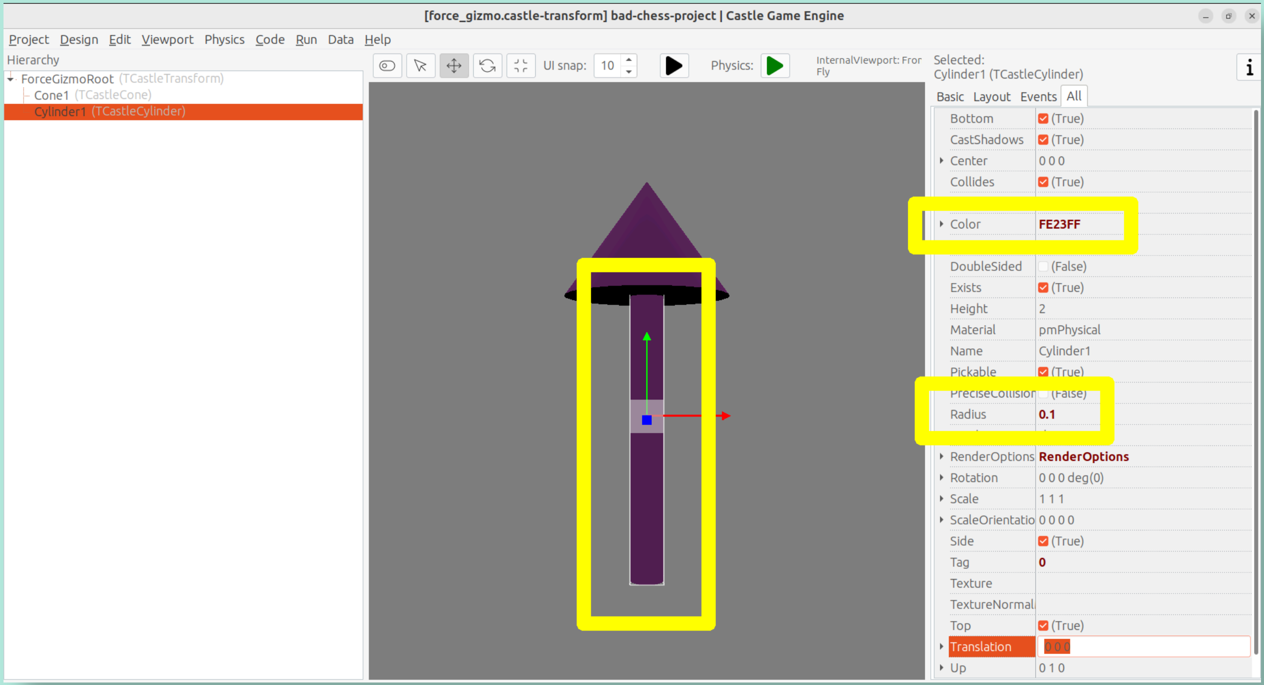
```
LibPng detected (version 1.6.39).
Gtk-Message: 18:35:02.386: Failed to load module "canberra-gtk-module"
Rendering Initialized: OpenGL 4.6 (Nvidia) (modern rendering: True) (for more info: LogGLInformationVerbose:=true)
Path: Program data path detected as "file:///home/michalis/sources/castle-engine/bad-chess/project/version_2_with_code/data/"
Configured 16 black and 16 white chess pieces

Command finished with status 0.
```

Underneath, add two components: `TCastleCylinder` and `TCastleCone`.
Adjust their `Height`, `Radius` (*on cylinder*), `BottomRadius` (on cone) and `Translation` to form a nice **3D arrow**.
Adjust their Color to something non-default to make things prettier. Remember that the arrow with later be lit by the lights we have set up in the main design (*gameviewmain.castle-user-interface*), so it will probably be brighter than what you observe now.
You can follow the values I have chosen on the screenshots below, but really these are just examples. Go ahead and create your own **3D arrow** as you please.


[force_gizmo.castle-transform] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
ForceGizmoRoot (TCastleTransform)
Cone1 (TCastleCone)
Cylinder1 (TCastleCylinder)
UI snap: 10
Physics:
InternalViewport: Fror Fly
Selected:
Cone1 (TCastleCone)
Basic Layout Events All

```
BottomRadius      0.5
Center            0 0 0
Collides          (True)
CollisionSphere   0
Color             FE23FF
DoubleSided       (False)
Exists            (True)
Height            0.7
Name              Cone1
Pickable          (True)
PreciseCollision  (False)
RenderLayer       rlParent
RenderOptions     RenderOptions
Rotation          0 0 0 deg(0)
Scale             1 1 1
ScaleOrientatio   0 0 0 0
Side              (True)
Tag               0
Texture
TextureNormal
Translation       0 1.07 0
Up                0 1 0
Visible           (True)
```

6.1. DESIGNING A 3D ARROW (CONTINUATION 2)


[force_gizmo.castle-transform] bad-chess-project | Castle Game Engine

Hierarchy
- ForceGizmoRoot (TCastleTransform)
  - Cone1 (TCastleCone)
  - Cylinder1 (TCastleCylinder)

Selected: Cylinder1 (TCastleCylinder)

Bottom	(True)
CastShadows	(True)
Center	0 0 0
Collides	(True)
Color	FE23FF
DoubleSided	(False)
Exists	(True)
Height	2
Material	pmPhysical
Name	Cylinder1
Pickable	(True)
PreciseCollision	(False)
Radius	0.1
RenderOptions	RenderOptions
Rotation	0 0 0 deg(0)
Scale	1 1 1
ScaleOrientatio	0 0 0 0
Side	(True)
Tag	0
Texture	
TextureNormal	
Top	(True)
Translation	0 0 0
Up	0 1 0

Project Design Edit Viewport Physics Code Run Data Help
UI snap: 10
Physics:
InternalViewport: Fron Fly
Basic Layout Events All

Now comes a bit difficult part.
We want to have an arrow that can easily rotate around a dummy box (in the actual game, it will rotate around a chess piece). Ideally, an arrow should also easily scale to visualize the force strength. I use the words easily to emphasize that we don't want to only rotate it in the editor, but we will also have to allow user to rotate it during the game. So the rotation and scale that are interesting to us must be very easy to get and set from code.

To do this, first add a dummy box representing a chess piece. I called it DebugBoxToBeHidden and set Size of the box to 2 3 2 to account for tall (large Y axis) chess pieces. Later we will make the box hidden by setting its Exists property to false.

Once you have a box, you want to add intermediate TCastleTransform components to

rotate the arrow (cone and cylinder) to be horizontal

move the arrow away from the box

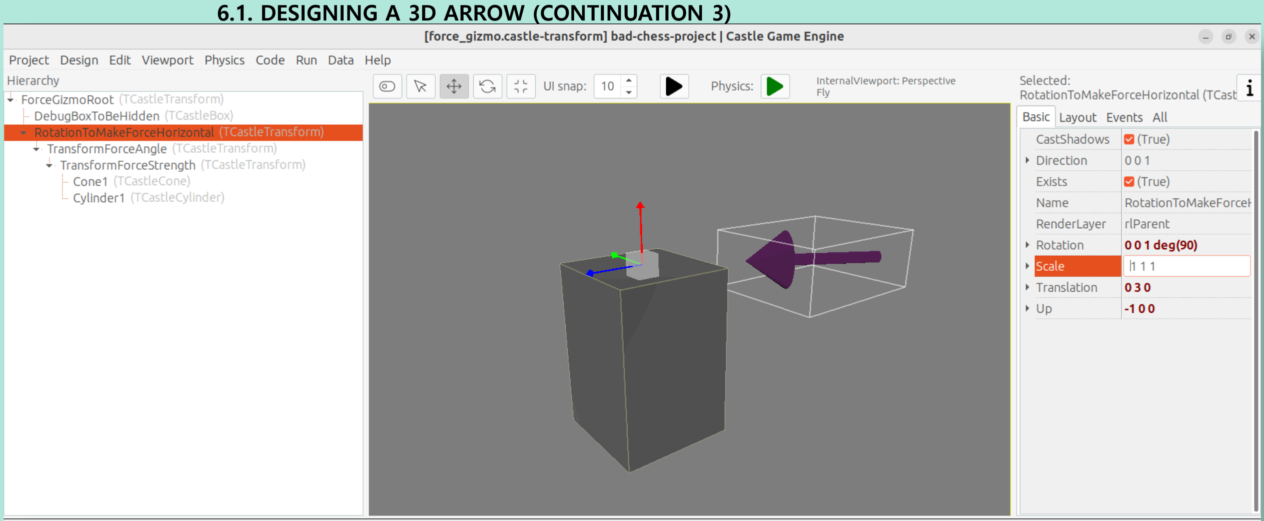
rotate the arrow around the box

scale the arrow.

There are multiple valid ways of achieving this. The key advise is to not hesitate to make a nested composition, that is place TCastleTransform within another TCastleTransform within another TCastleTransform and so on. Let each TCastleTransform perform a single function. Take it step by step and you will get to a valid solution (and there are really a number of possible ways to arrange this).

See my arrangement on the screenshots below. If you get stuck, just use the design from our resulting project in https://github.com/castle-engine/bad-chess/ (in project/version_2_with_code subdirectory).

6.1. DESIGNING A 3D ARROW (CONTINUATION 3)

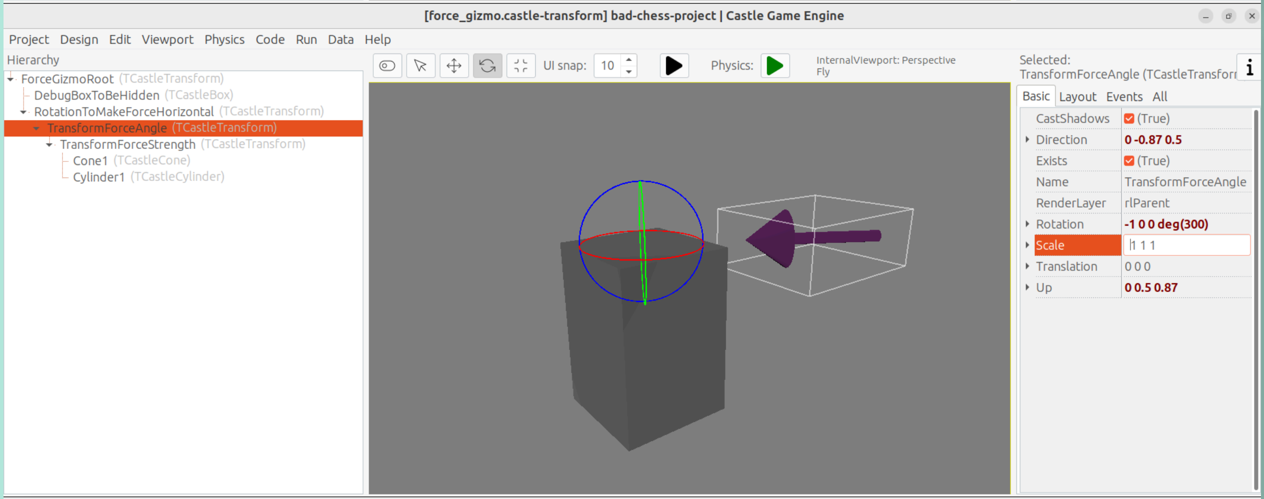

Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
InternalViewport: Perspective
Fly
Basic Layout Events All

[force_gizmo.castle-transform] bad-chess-project | Castle Game Engine

- ForceGizmoRoot (TCastleTransform)
  - DebugBoxToBeHidden (TCastleBox)
  - RotationToMakeForceHorizontal (TCastleTransform)
    - TransformForceAngle (TCastleTransform)
      - TransformForceStrength (TCastleTransform)
        - Cone1 (TCastleCone)
        - Cylinder1 (TCastleCylinder)

Selected:
RotationToMakeForceHorizontal (TCast

CastShadows	(True)
Direction	0 0 1
Exists	(True)
Name	RotationToMakeForceH
RenderLayer	rlParent
Rotation	**0 0 1 deg(90)**
Scale	1 1 1
Translation	**0 3 0**
Up	**-1 0 0**

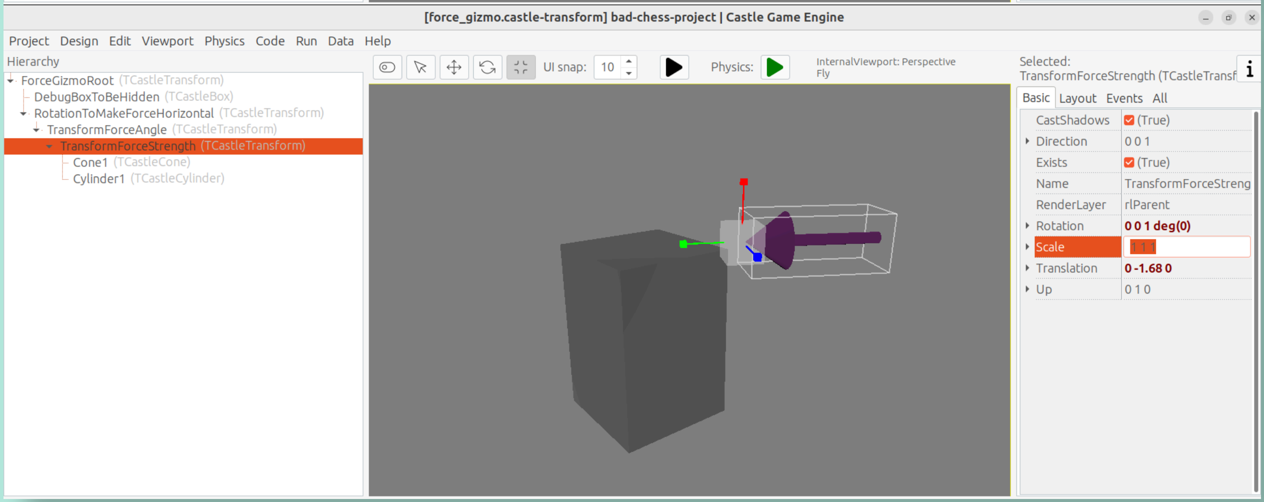

Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
InternalViewport: Perspective
Fly
Basic Layout Events All

[force_gizmo.castle-transform] bad-chess-project | Castle Game Engine

- ForceGizmoRoot (TCastleTransform)
  - DebugBoxToBeHidden (TCastleBox)
  - RotationToMakeForceHorizontal (TCastleTransform)
    - TransformForceAngle (TCastleTransform)
      - TransformForceStrength (TCastleTransform)
        - Cone1 (TCastleCone)
        - Cylinder1 (TCastleCylinder)

Selected:
TransformForceAngle (TCastleTransforr

CastShadows	(True)
Direction	**0 -0.87 0.5**
Exists	(True)
Name	TransformForceAngle
RenderLayer	rlParent
Rotation	**-1 0 0 deg(300)**
Scale	1 1 1
Translation	0 0 0
Up	**0 0.5 0.87**


Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
InternalViewport: Perspective
Fly
Basic Layout Events All

[force_gizmo.castle-transform] bad-chess-project | Castle Game Engine

- ForceGizmoRoot (TCastleTransform)
  - DebugBoxToBeHidden (TCastleBox)
  - RotationToMakeForceHorizontal (TCastleTransform)
    - TransformForceAngle (TCastleTransform)
      - TransformForceStrength (TCastleTransform)
        - Cone1 (TCastleCone)
        - Cylinder1 (TCastleCylinder)

Selected:
TransformForceStrength (TCastleTransf

CastShadows	(True)
Direction	0 0 1
Exists	(True)
Name	TransformForceStreng
RenderLayer	rlParent
Rotation	**0 0 1 deg(0)**
Scale	1 1 1
Translation	**0 -1.68 0**
Up	0 1 0

6.1 DESIGNINGA3DARROW (CONTINUATION 4)

The outcome of my design is that I know that from code, I can:

Adjust Rotation property of the `TransformForceAngle` component to be a simple rotation around the X axis. The angle of this rotation can be chosen by user and effectively the arrow will orbit around the debug box (*chess piece*).

Adjust Y of the Scale property of the `TransformForceStrength` component. The amount of this scale can be chosen by user to visualize the strength.

Remember to set Exists of the DebugBoxToBeHidden component to false once done.

6.2. ADD THE ARROW TO THE MAIN DESIGN

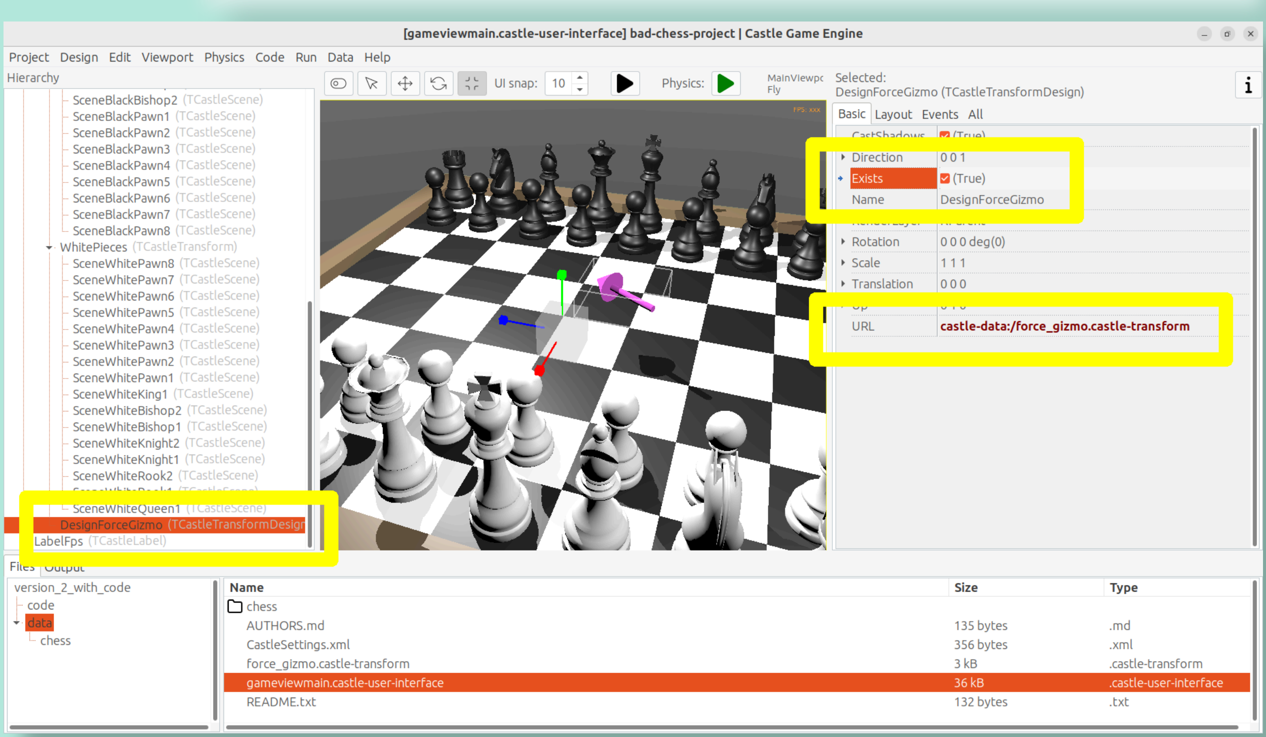
To test that it works, add the arrow design to the main design using the editor.

Save the design `force_gizmo.castle-transform`, open our main design in `gameviewmain`.castle-user-interface, select the Items component inside **MainViewport** and drag-and-drop the file `force_gizmo.castle-transform` (*from the "Files" panel below*) on the hierarchy.

The result should be that a new component called `DesignForceGizmo1` is created and placed as a child of Items. The component class is `TCastleTransformDesign`, which means that it's an instance of `TCastleTransform` loaded from another file with `.castle-transform` extension. The URL property of this component should automatically be set to indicate our `force_gizmo.castle-transform` file.

Rename this component to just `DesignForceGizmo` (*up to you, but I think it makes things clearer — we will only ever need one such gizmo*).
Moreover, change the `Exists` property of this component to false because initially, we don't want this component to be visible or pickable by the mouse.

The screenshot below shows the state right before I set `Exists` to false.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
MainViewpc
Fly
Basic Layout Events All
Files Output

SceneBlackBishop2 (TCastleScene)
SceneBlackPawn1 (TCastleScene)
SceneBlackPawn2 (TCastleScene)
SceneBlackPawn3 (TCastleScene)
SceneBlackPawn4 (TCastleScene)
SceneBlackPawn5 (TCastleScene)
SceneBlackPawn6 (TCastleScene)
SceneBlackPawn7 (TCastleScene)
SceneBlackPawn8 (TCastleScene)
WhitePieces (TCastleTransform)
SceneWhitePawn8 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteQueen1 (TCastleScene)
DesignForceGizmo (TCastleTransformDesign)
LabelFps (TCastleLabel)

Selected:
DesignForceGizmo (TCastleTransformDesign)

CastShadows	✓ (True)
Direction	0 0 1
Exists	✓ (True)
Name	DesignForceGizmo
RenderLayer	
Rotation	0 0 0 deg(0)
Scale	1 1 1
Translation	0 0 0
Up	0 1 0
URL	castle-data:/force_gizmo.castle-transform

version_2_with_code
code
data
chess

Name	Size	Type
chess		
AUTHORS.md	135 bytes	.md
CastleSettings.xml	356 bytes	.xml
force_gizmo.castle-transform	3 kB	.castle-transform
gameviewmain.castle-user-interface	36 kB	.castle-user-interface
README.txt	132 bytes	.txt

6.3. LETTING USER CONTROL THE ARROW

We need to declare and initialize the fields that describe current angle and strength.

Add this to the private section of the `TViewMain` class:

```
TransformForceAngle, TransformForceStrength: TCastleTransform;
ForceAngle: Single;
ForceStrength: Single;
```

Then let's set some constants. You can declare them at the beginning of unit GameViewMain implementation:

```
 const
  MinStrength = 1;
  MaxStrength = 1000;

  MinStrengthScale = 1;
  MaxStrengthScale = 3;

  StrengthChangeSpeed = 30;
  AngleAChangeSpeed = 10;
```

Add to the uses clause new necessary units: `Math, CastleUtils.`

Finally add to the `TViewMain.Start` additional piece of code to initialize everything:

```
TransformForceAngle :=
    DesignForceGizmo.DesignedComponent('TransformForceAngle') as TCastleTransform;
TransformForceStrength :=
    DesignForceGizmo.DesignedComponent('TransformForceStrength') as TCastleTransform;
ForceAngle := 0; // 0 is default value of Single field anyway
TransformForceAngle.Rotation := Vector4(1, 0, 0, ForceAngle);
ForceStrength := 10; // set some sensible initial value
TransformForceStrength.Scale := Vector3(1,
   MapRange(ForceStrength, MinStrength, MaxStrength, MinStrengthScale,MaxStrengthScale), 1);
```

NOTE that we initialize the components within our **DesignForceGizmo** design using the `DesignForceGizmo.DesignedComponent(...)` call.
This is necessary, as in general you can have multiple instances of the design `force_gizmo.castle-transform` placed in your view. So the published fields of the view cannot be automatically associated with components in nested designs.

Moreover we synchronize Single fields `ForceStrength` and `ForceAngle` with their counterpart `TCastleTransform` instances. Single in **Pascal** is a simple **floating-point number**, which is super-easy to manipulate. We treat two `TCastleTransform` instances above as just a fancy way to visualize these numbers as 3D rotation and scale.

You may want to lookup what the `MapRange` function does in **Castle Game Engine API reference.** In short, it's a comfortable way of doing a linear interpolation, converting from one range to another.

Now that we have initialized everything, let's actually show the **DesignForceGizmo** when user selects a chess piece. We already have a code to select chess piece on mouse click. Just extend it to show the **DesignForceGizmo** and reposition it at the selected chess piece.

6.3. LETTING USER CONTROL THE ARROW (CONTINUATION)

```
 if Event.IsMouseButton(buttonLeft) then
begin
  OldSelected := ChessPieceSelected;
  if (ChessPieceHover <> nil) and
    (ChessPieceHover <> ChessPieceSelected) then
  begin
    ... // keep existing code

    // new lines:
    DesignForceGizmo.Exists := true;
    DesignForceGizmo.Translation := ChessPieceSelected.Parent.
WorldTranslation;
  end;
  Exit(true); // mouse click was handled
end;
```

NOTE: You may wonder about an alternative approach, where we don't reposition **DesignForceGizmo**, but instead dynamically change it's parent, like `DesignForceGizmo.Parent := ChessPieceSelected.Parent.`

This would work too, alas with some additional complications: the rotation of the selected object, once we flick it, would rotate also the gizmo. This would make the calculation of "desired flick direction" later more complicated.

So I decided to go with the simpler approach of just repositioning the `DesignForceGizmo`.
If you want to experiment with the alternative complicated approach, go ahead.
One solution would be to design `DesignForceGizmo` such that you can later do `TransformForceAngle.GetWorldView(WorldPos, WorldDir, WorldUp)` and use resulting WorldDir as a force direction.

But since we keep things simple... we're almost done.
You can run the game and see that selecting a chess piece shows the arrow gizmo properly.
It remains to allow user to change direction and strength.
We can do this by observing the keys user presses in the `Update` method.

The code below allows to rotate the arrow (***make it orbit around the chess piece***) using left and right arrow keys, and change force strength (***scaling the arrow***) using up and down arrow keys. Add this code to your existing Update method:

```
procedure TViewMain.Update(const SecondsPassed: Single; var HandleInput: Boolean);
begin
  ... // keep existing code
  if Container.Pressed[keyArrowLeft] then
    ForceAngle := ForceAngle - SecondsPassed * AngleAChangeSpeed;
  if Container.Pressed[keyArrowRight] then
    ForceAngle := ForceAngle + SecondsPassed * AngleAChangeSpeed;
  if Container.Pressed[keyArrowUp] then
    ForceStrength := Min(MaxStrength, ForceStrength + SecondsPassed * StrengthChangeSpeed);
  if Container.Pressed[keyArrowDown] then
    ForceStrength := Max(MinStrength, ForceStrength - SecondsPassed * StrengthChangeSpeed);

  TransformForceAngle.Rotation := Vector4(1, 0, 0, ForceAngle);
  TransformForceStrength.Scale := Vector3(1,
    MapRange(ForceStrength, MinStrength, MaxStrength, MinStrengthScale, MaxStrengthScale),1);
end;
```

❼ FLICK THAT CHESS PIECE!

Looks like we have all the knowledge we need.

- We know how to flick the chess piece,
- we know which chess piece to flick,
- we know the direction and strength of the flick.

You can consult the code we did a few sections before, in the exercise "Push the chess piece using physics".
Our new code will be similar.

Add it to the `Press` method implementation:

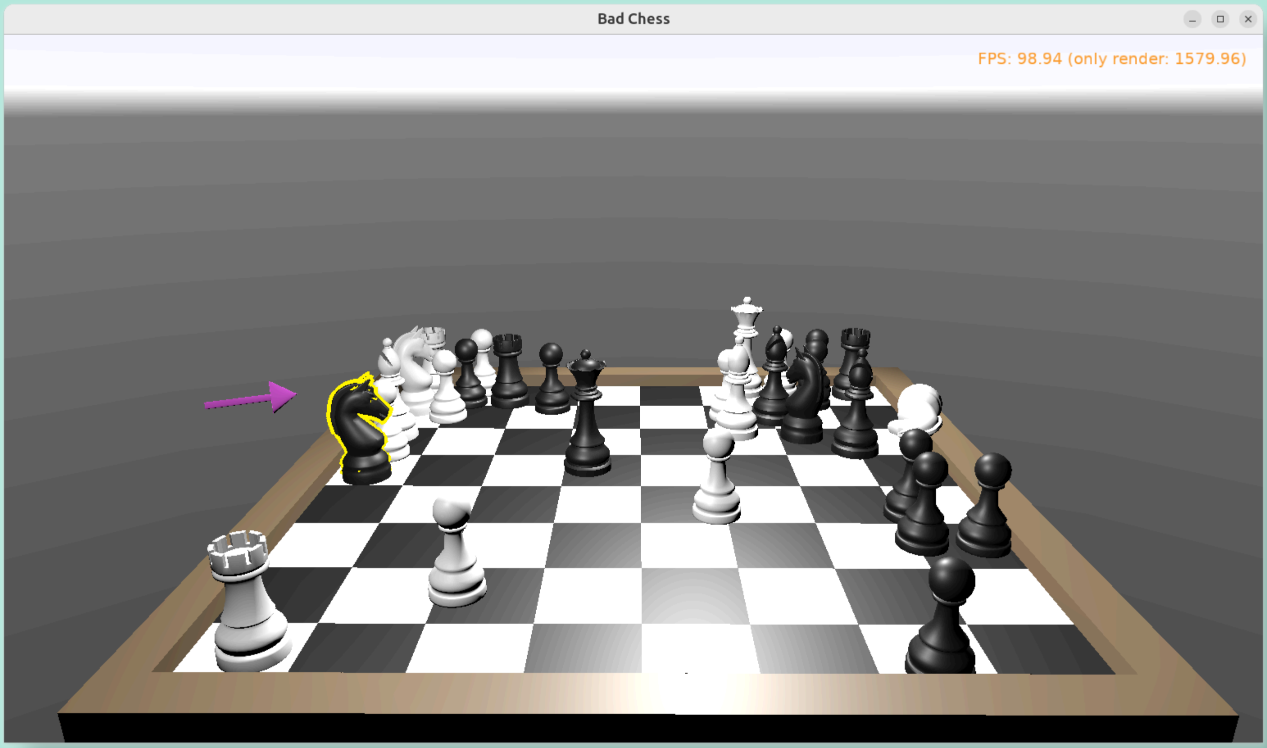
```
function TViewMain.Press(const Event: TInputPressRelease): Boolean;
var
  ... // keep existing variables used by other inputs
  ChessPieceSelectedScene: TCastleScene;
  ForceDirection: TVector3;
begin
  Result := inherited;
  if Result then Exit; // allow the ancestor to handle keys

  ... // keep existing code handling other inputs

  if Event.IsKey(keyEnter) and (ChessPieceSelected <> nil) then
  begin
    ChessPieceSelectedScene := ChessPieceSelected.Parent as TCastleScene;
    MyBody := ChessPieceSelectedScene.FindBehavior(TCastleRigidBody) as
                                                          TCastleRigidBody;
   ForceDirection := RotatePointAroundAxis(
     Vector4(0, 1, 0, ForceAngle), Vector3(-1, 0, 0));

   MyBody.ApplyImpulse(
     ForceDirection * ForceStrength,
     ChessPieceSelectedScene.WorldTranslation);
   // unselect after flicking; not strictly necessary, but looks better
   ChessPieceSelected := nil;
   DesignForceGizmo.Exists := false;
   Exit(true); // input was handled
  end;
end;
```

Depending on how you designed the `force_gizmo.castle-transform` design, you may need to adjust the `ForceDirection` calculation, in particular the 2nd parameter to `RotatePointAroundAxis` which is a direction used when angle is zero.

There's nothing magic about our value `Vector3(-1, 0, 0)`, it just follows our `force_gizmo.`castle-transform design.

Run the game and see that you can now flick the chess pieces!

- **Select** the chess piece by clicking with mouse.
- **Rotate** the force by left and right arrow keys.
- **Change** the force strength by up and down arrow keys.
- **Flick** the chess piece by pressing Enter.
- **Repeat** 🙂



❽ CONCLUSION AND FUTURE IDEAS

Invite a friend to play with you. Just take turns using the mouse to flick your chess pieces and have fun

I am sure you can invent now multiple ways to make this better.

- Maybe each player should be able to flick only its own chess pieces?
  We already know which chess piece is black or white (*the Black boolean field in* `TChessPieceBehavior`), though we didn't use it for anything above.
  You should track which player flicked the object last (black or white), and only allow to choose the opposite side next time.

- Maybe you want to display some user interface, like a label, to indicate whose turn is it?
  Just drop a `TCastleLabel` component on view, and change the label's Caption whenever you want.

- Maybe you want to show the current force angle and strength - either as numbers, or as some colorful bars? Use `TCastleRectangleColor` for a trivial rectangle with optional border and optionally filled with a color.

- Maybe you want to implement a proper chess game? Sure, just track in code all the chess pieces and the chessboard tiles — what is where. Then add a logic that allows player to select which piece and where should move. Add some validation. Add playing with a computer opponent if you wish — there are standardized protocols to communicate with "chess engines" so you don't need to implement your own chess AI from scratch.

- Maybe you want to use networking? You can use a number of networking solutions (*any Pascal library*) together with **Castle Game Engine**.
  See `https://castle-engine.io/manual_network.php` .
  We have used the engine with Indy and RNL (Realtime Network Library).
  In the future we plan to integrate the engine with Nakama, an open-source server and client framework for multi-player games.

❽ CONCLUSION AND FUTURE IDEAS (CONTINUATION)

- Maybe you want to use networking?
  You can use a number of networking solutions
  (*any Pascal library*) together with **Castle Game Engine.**
  See **`https://castle-engine.io/manual_network.php`** .

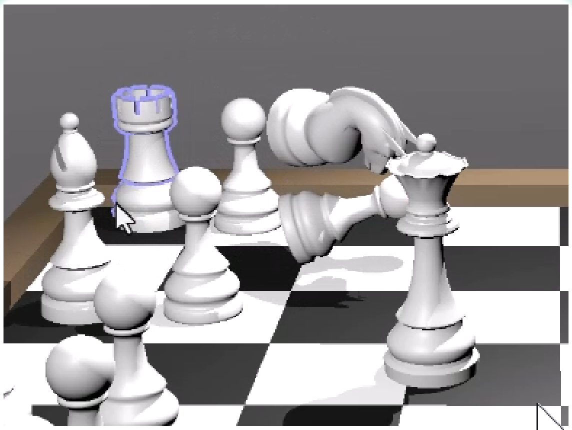
We have used the engine with Indy and **RNL** (**Realtime Network Library**).
In the future we plan to integrate the engine with **Nakama**, an open-source server and client framework for multi-player games.

- Maybe you want to deploy this game to other platforms, in particular mobile? Go ahead.
  The code we wrote above is already cross-platform and can be compiled using
  Castle Game Engine to any Android or iOS.
  Our build tool does everything for you, you get a ready **APK, AAB** or **IPA** file to install on your phone. See the engine documentation on
  `https://castle-engine.io/manual_cross_platform.php` .

  Although keyboard inputs will not work on mobile.
  You need to invent and implement a new user interface to rotate the `force`,
  change the `strength`, and actually throw the chess piece.
  It is simplest to just show clickable buttons to perform the relevant actions.
  The `TCastleButton` class of the engine is a button with a freely customizable look.

If you want to learn more about the engine, read the documentation on
`https://castle-engine.io/` and join our community on forum and Discord:
`https://castle-engine.io/talk.php`
Last but not least, if you like this article and the engine, we will appreciate if you support us on **Patreon**
`https://www.patreon.com/castleengine`. We really count on your support.

Finally, above all, have fun! Creating games is a wild process and experimenting with "what feels good" feeling is the right way to do it. I hope you will enjoy it.



ADVERTISEMENT

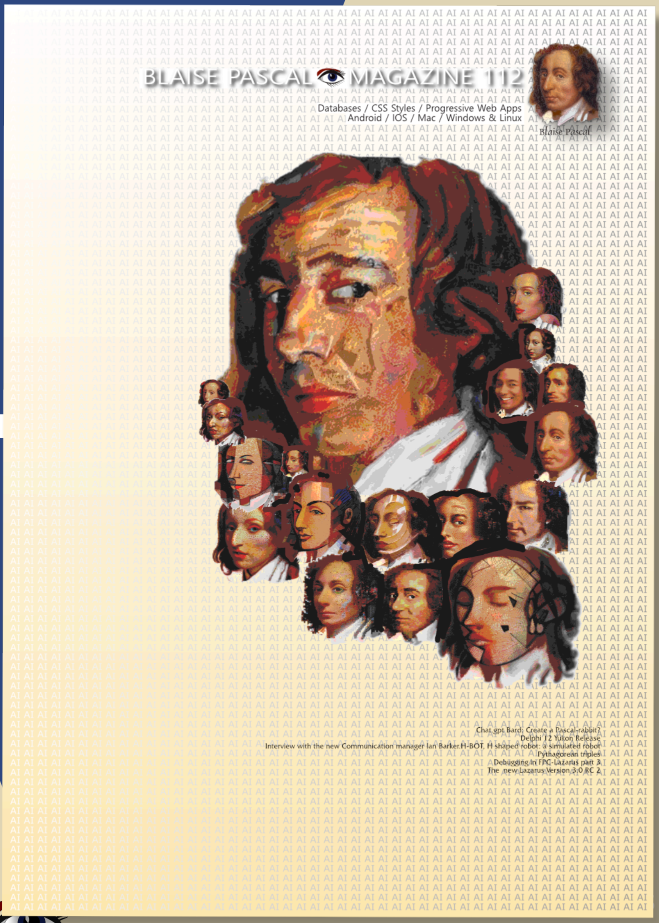
LAZARUS HANDBOOK (PDF)
+SUBSCRIPTION 1 YEAR

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

SPECIAL OFFER € 75

Ex Shipping

+


BLAISE PASCAL MAGAZINE 112

Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Blaise Pascal

Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

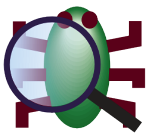
UNIQUEMENT CHEZ BARNSTEN JUSQU'À 30 % DE RÉDUCTION JUSQU'AU 10 NOVEMBRE

MISE À JOUR VERS LA VERSION SUIVANTE INCLUSE

barnsten

Tel.: +31 23 542 22 27 Web: www.barnsten.com
Info: info@barnsten.com

THE LAZARUS DEBUGGER PART 4: TAKING A LOOK – WATCHES



BY MARTIN FRIEBE

Starter Expert

IT'S ALL ABOUT PRESENTATION

We already used the `Watches` and `Locals` window to look at data while debugging. However, not all data is equal and there are many ways to look at it.

NOTE: This article is based on the **FpDebug** debugger in **Lazarus 2.2.6** and the project using **DWARF 3.** Using any other of the debugger backends or settings may result in a different display of the values.
Some of the features shown in this article require **Lazarus 3.0**

OUR SAMPLE CODE FOR THIS ARTICLE:

```pascal
program project1;

uses Classes, StrUtils;

const
  FORMAT_HEX = 0;
  FORMAT_OCT = 1;
  FORMAT_BIN = 2;

type

  TApiData = LongWord;

  (* This API expects a LongWord made up from
     1 Byte (Bits 24..31): A Digit Count (Size)
     1 Byte (Bits 16..23): A Format
     1 Word (Bits  0..15): A Number
  *)

  { TMyAPIBase }

  TMyAPIBase = class
  private
    FList: TStringList;
  public
    constructor Create;
    destructor Destroy; override;
    procedure Add(ASize: Byte; AFormat: Byte; ANumber: Word);
    function GetText: String;
  end;

  TMyAPI = class(TMyAPIBase)
  type
    TApiDataStruct = packed record
      Size: Byte;
      Format: Byte;
      Number: Word;
    end;
  public
    procedure ApiStore(AData: TApiData);
    procedure Print;
  end;
```

```

constructor TMyAPIBase.Create;
begin
  inherited Create;
  FList := TStringList.Create;
end;

destructor TMyAPIBase.Destroy;
begin
  FList.Destroy;
  inherited Destroy;
end;

procedure TMyAPIBase.Add(ASize: Byte; AFormat: Byte; ANumber: Word);
begin
  case AFormat of
    FORMAT_HEX:
      FList.Add(Dec2Numb(ANumber, ASize, 16));
    FORMAT_OCT:
      FList.Add(Dec2Numb(ANumber, ASize, 8));
    FORMAT_BIN:
      FList.Add(Dec2Numb(ANumber, ASize, 2));
    else
      FList.Add(Dec2Numb(ANumber, ASize, AFormat));
  end;
end;

function TMyAPIBase.GetText: String;
begin
  Result := FList.Text;
end;

procedure TMyAPI.ApiStore(AData: TApiData);
var
  d: TApiDataStruct;
begin
  d := TApiDataStruct(AData);
  with d do
    Add(Size, Format, Number);
end;

procedure TMyAPI.Print;
begin
  WriteLn(GetText);
end;

function GetApiValue(S, F, N: Integer): TApiData;
begin
  Result := S << 24 + F << 16 + N;
end;

var
  Api: TMyAPI;
  Val: TApiData;

begin
  Api := TMyAPI.Create;

  Val := GetApiValue(4, FORMAT_HEX, 42);
  Api.ApiStore(Val);
```

```
103.
104.   Val := GetApiValue(8, FORMAT_OCT, 42);
105.   Api.ApiStore(Val);
106.
107.   Val := GetApiValue(16, FORMAT_BIN, 42);
108.   Api.ApiStore(Val);
109.
110.   Val := GetApiValue(3, 10, 42);
111.   Api.ApiStore(Val);
112.
113.   writeln(Api.GetText);
114.   readln;
115.   Api.Free;
116. end.
```

We have an Object with some custom **API** that requires several values bit-packed into a single parameter. We call it to format the value **42** as **Hex, Oct, Bin** and **Decimal**, and we expect it to print:

```
002A
00000052
0000000000101010
042
```

But instead we get:

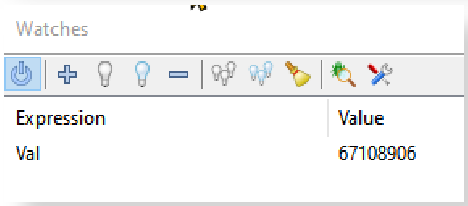
```
000000000000000000000000000000000000000400
000000000000000000000000000000000000000801
000000000000000000000000000000000000001002
00000000000000000000000000000000000000030A
```

To debug it, we run **(F9)** to a breakpoint that we put on **line** `102`, which is the first call to

```
  Api.ApiStore(Val);
```

THE DISPLAY FORMAT

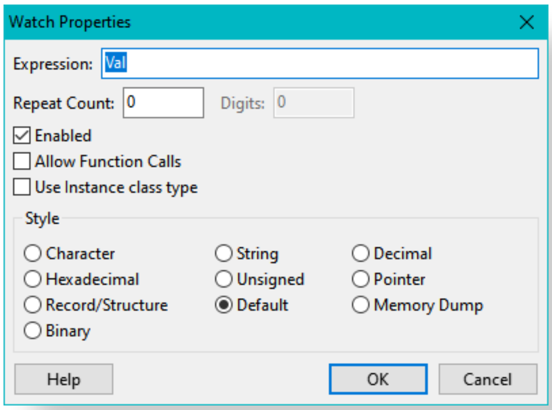
When we look at "**Val**" either in the **Locals** or **Watches** window we get:



Watches

Expression	Value
Val	67108906

Unfortunately, that doesn't tell us anything about the bytes in the value. Luckily, the debugger offers us some options, which we can find in the watches properties dialog. The quickest way to open it is to double click the watch. Alternatively, we can choose "properties" from the context menu of the watch, or press the 🛠 button.


Watch Properties

Expression: Val

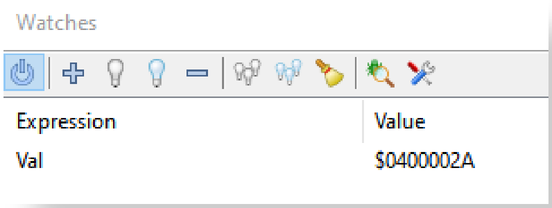
Repeat Count: 0 Digits: 0

[✓] Enabled
[ ] Allow Function Calls
[ ] Use Instance class type

Style

○ Character ○ String ○ Decimal
○ Hexadecimal ○ Unsigned ○ Pointer
○ Record/Structure ● Default ○ Memory Dump
○ Binary

For now, we are interested in the "Style" or more commonly referred to as "Display format".
As "Val" is a number, we are interested in styles that apply to numbers. And we need a format that allows us to see the boundaries of the contained bytes. A good choice for this is "Hexadecimal".
The output changes to:


Watches

Expression	Value
Val	$0400002A

Val was initialized with the values passed to

```
//function GetApiValue(S, F, N: Integer): TapiData;
Val := GetApiValue(4, FORMAT_HEX, 42);
```

Looking at the hex value we compare this with the description of "TApiData"

- Byte 1 = 04: This is the Size we wanted
- Byte 2 = 00: This is FORMAT_HEX
- Byte 3 and 4: = $002a = 42: The value we specified

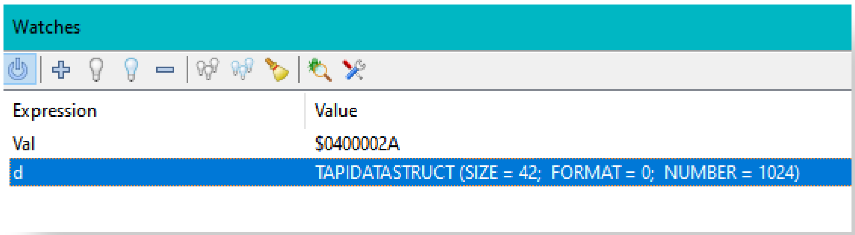
So that looks good.

THE DETAIL PANE

Next, we step in, and go to line 80 to pause after

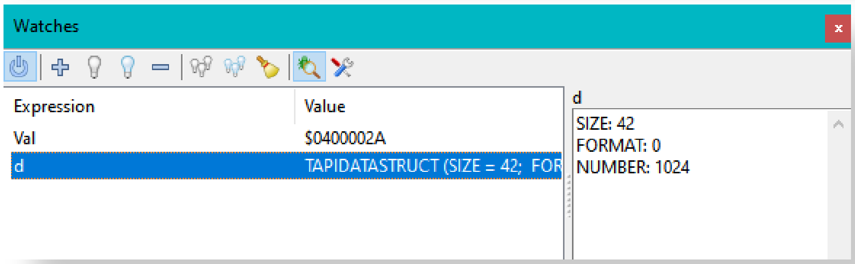
```
d := TApiDataStruct(AData);
```

has been executed. Now, we can have a look at "d".


Watches

Expression	Value
Val	$0400002A
d	TAPIDATASTRUCT (SIZE = 42; FORMAT = 0; NUMBER = 1024)

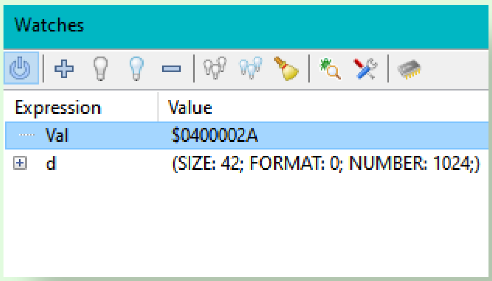
Before we continue to debug a little bit more on the watches window. The above shows all fields in one line, that may just work for the 3 fields we have, but if there are more fields they would be cut off. For this there is the "**Detail pane**". It can be toggled with the button and shows the content of the selected watch in a more spacious way.


Watches

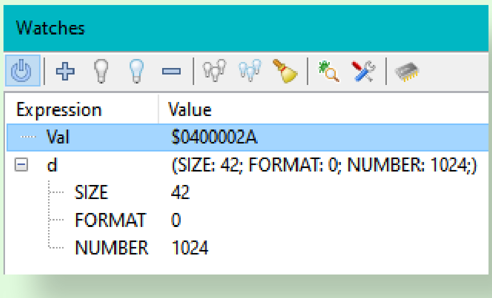
Expression	Value
Val	$0400002A
d	TAPIDATASTRUCT (SIZE = 42; FOR

d
SIZE: 42
FORMAT: 0
NUMBER: 1024

In **Lazarus 3.0** watches can also be expanded in the main view.
Structured value have a **+ symbol** to allow unfolding them.


Watches

Expression	Value
Val	$0400002A
⊞ d	(SIZE: 42; FORMAT: 0; NUMBER: 1024;)

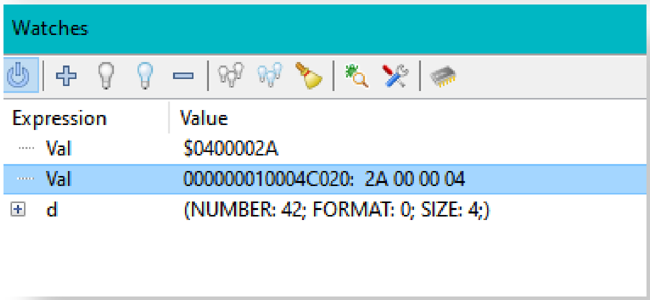

Watches

Expression	Value
Val	$0400002A
⊟ d	(SIZE: 42; FORMAT: 0; NUMBER: 1024;)
SIZE	42
FORMAT	0
NUMBER	1024

To continue our debugging, while "**Val**" still looks correct, the values in "`d: TApiDataStruct`" do not match our expectation. This is despite "`TApiDataStruct`" has the fields `Size, Format, Number` in the same order in which the bytes are present in "**Val**".

DUMPING MEMORY

Time to have another look at "**Val**". Going back to the watch properties dialog, lets pick "**Memory Dump**". Actually, we will add a 2nd watch for "**Val**" as memory dump, so we can compare it with the hexadecimal representation of "**Val**".
In **Lazarus 3.0** we can then drag and drop the new "**Val**" watch, to be shown right below the existing "**Val**". In **2.2.6** the new "**Val**" will be at the end of the list.


Watches

Expression	Value
Val	$0400002A
Val	000000010004C020: 2A 00 00 04
⊞ d	(NUMBER: 42; FORMAT: 0; SIZE: 4;)

Now, we have some interesting information. The bytes in memory start with `$2A`, the lowest byte of "**Val**". This is a little endian machine. So while looking at the data as a single `LongWord` number the Size is upfront, but in memory it is at the very end.
We need to change the order in which the record is defined to match the layout in memory.

```
type
  TApiDataStruct = packed record
   Number: Word;
   Format: Byte;
   Size: Byte;
  end;
```

Running the app again will now show us the correct result.

We are done with the debug session, but lets continue and have a look at some more features the watch windows has to offer.

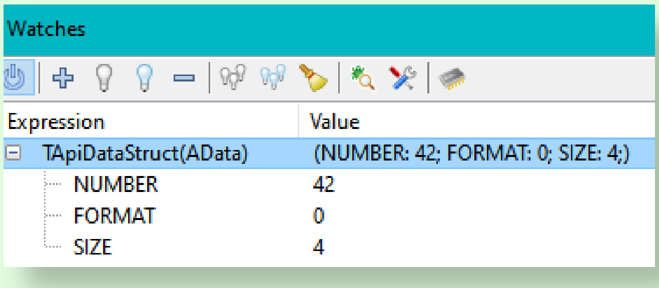
VARIABLE OR EXPRESSION

In the previous articles were already some examples of adding expressions instead of variables to the watches Window. **FpDebug** can interpret most expression that **Fpc** can. Accessing fields, array entries, working with pointers, doing arithmetic, type casts, all of it.
However it can not yet access properties, if the use a getter function.

With **Lazarus 3.0** in the above example we would not have needed

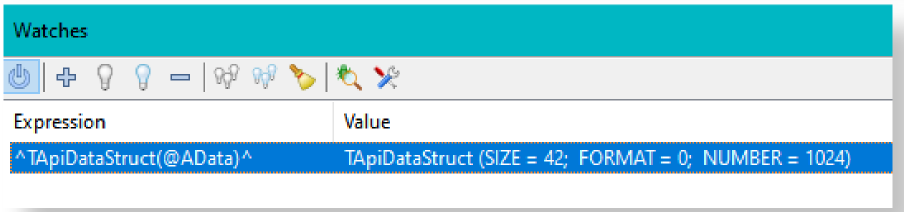
```
  d := TApiDataStruct(AData);
```

While **FpDebug** already knew about many typecasts (*and conversions*), now it can also cast arbitrary data to record, so long as the data has the same size as the record.



Expression	Value
TApiDataStruct(AData)	(NUMBER: 42; FORMAT: 0; SIZE: 4;)
NUMBER	42
FORMAT	0
SIZE	4

With **Lazarus 2.2.6** or for data that has a size different from the target type of the cast, its required to take the address and cast to a pointer. Luckily, it is possible to write "`^TApiDataStruct`" to use it as pointer to the type.



Expression	Value
^TApiDataStruct(@AData)^	TApiDataStruct (SIZE = 42; FORMAT = 0; NUMBER = 1024)

NOTE: When using debugger backends without **FpDebug**, like pure "**gdb**" or pure "**lldb**", some expressions may not work. Some operators may not exist, or the may have different precedence leading to a different result.

CALLING FUNCTIONS

Watch expressions can contain function calls. This feature must be enabled in the global **IDE** options under **Debugger > General.** And it must then be enabled in the properties of each watch that wants to make use of it.
The reason for this requiring such explicit enabling is that evaluating a function call may have unwanted side effects.
If the called function modifies a global variable or a field of an object, or any other non local data, then this change will persist after the evaluation. If you continue debugging, **your app will run with those changes, and may behave different.** (*Mind that even just temporarily allocating memory can change the state of your app, as it can change the future memory layout.*)

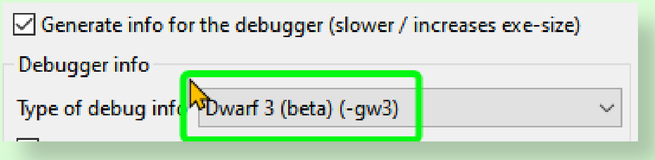
In **Lazarus 3.0**, the debugger can run the function call, with all other threads suspended.
This is the default, as your app is paused, and those other threads are not meant to run.
Other threads may however need to run, if the called function may wait for an event from one of the other threads.

For the debugger to call a function, it is mandatory to specify the "()" even if there are no parameters.

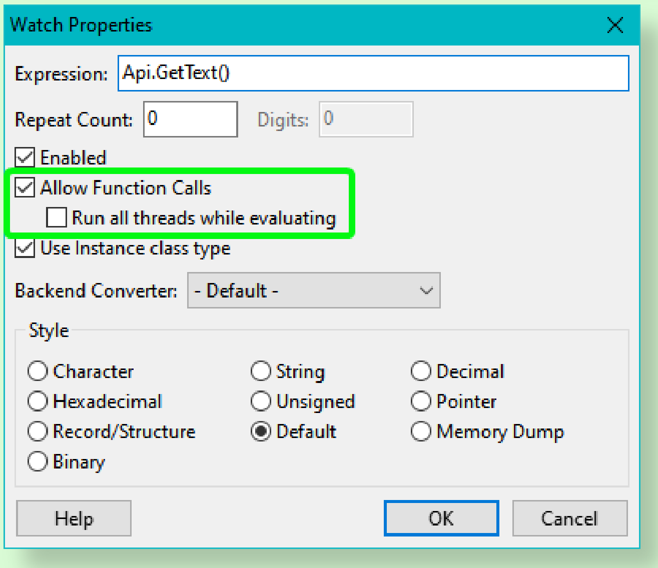
NOTE: Values of some data types can not be used as parameter of a function. Equally functions can only be called if they return a supported type.

Lets run to the line "`writeln(Api.GetText);`", and add the following watch.

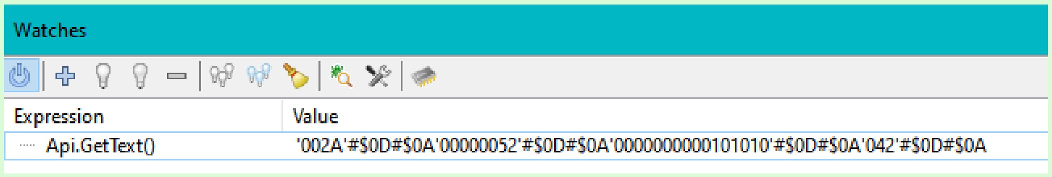
In "**Project Options" → "Debugger**":



☑ Generate info for the debugger (slower / increases exe-size)
Debugger info
Type of debug info: Dwarf 3 (beta) (-gw3)



Watch Properties
Expression: Api.GetText()
Repeat Count: 0 Digits: 0
☑ Enabled
☑ Allow Function Calls
☐ Run all threads while evaluating
☑ Use Instance class type
Backend Converter: - Default -
Style: ○ Character ○ String ○ Decimal ○ Hexadecimal ○ Unsigned ○ Pointer ○ Record/Structure ● Default ○ Memory Dump ○ Binary



Expression	Value
Api.GetText()	'002A'#$0D#$0A'00000052'#$0D#$0A'0000000000101010'#$0D#$0A'042'#$0D#$0A

Note: Functions returning Strings (Long/Ansistring) or have parameters of this type only work with Lazarus 3.0. They also require DWARF 3 (or up) as debug info.
The function call may otherwise crash, and leave your debug session damaged, i.e. your app will be unable to continue as expected.

CLASSES: DECLARED OR INSTANTIATED.

```
procedure ButtonClick(Sender: TObject);
```

By default, the debugger displays any data according to its declaration. So if you watch "`Sender`" you will see only a `TObject`, which gives you very little information.

You can of course type cast the variable yourself and add a watch like "`TButton(Sender)`. But if the callback is shared by different controls (or if you just want something more convenient) the debugger can do that for you.

The debugger can find the actual instantiated class of the object and display the watch according to that class. This feature is on by default. You can disable it globally in the **Options > Debugger > General:** "Automatically set '**Use instance class type**' **for new watches**".
Or you can toggle it in the properties of each watch changing "**Use instance class type**".

This feature only works, if the declared type is a class (`TObject` or descendant). If you have a variable of different type like "**Data: PtrUInt**", then you first need to typecast it to "**TObject(Data)**". And then, the debugger will further cast it to the instance's class.

ARRAYS AND "REPEAT COUNT"

If an array is declared (*or set via* "**SetLength()**") to have a certain amount of entries, the debugger will show the declared amount of entries (*or the limit set in the global debugger options*).
Sometimes you may expect more data after the end of the array.
In that case you can set the "**repeat count**" in the watches property, and specify how many elements to show. The "**repeat count**" can also limit the amount of shown values to less than the default.
In order for "**repeat count**" to work, your watch must be for the entire array.
You can not apply a "**repeat count**" for an array element "`MyArray[11]`".
This means you can't get a slice of e.g. 10 elements starting at 11.

In **Lazarus 3.0** you can get array slices by using the [11..20] syntax.

The "**repeat count**" can also be applied to memory dumps. By default, they return the memory for the size of the watched value (*e.g. 4 bytes for a LongInt*). But with "**repeat count**" you can get a bigger amounts of memory.

ENABLED, DISABLED OR POWER OFF

You can disable all or some of the watches. This will prevent the debugger from evaluating them. It may be useful, if a watch is calling a function and you don't want it to be called, except under certain conditions.

It can also be useful if you have some watches that return tremendously large amount of data (*huge structures with long arrays included*). Such data may take a moment to retrieve, and especially if you have several such watches, they may add up to take noticeable time.
The "**power**" button can preserve the current state of the window. All watches will keep their currently displayed value for as long as power is off.

Note: Power was originally introduced when the debugger was using **gdb** and **watches** could take long time to process. Turning watches off (*and relying on locals instead*) did improve the time between steps **(F7/F8**) under gdb.

TO THE CLIPBOARD

In the context menu are several options to copy the selected value to the clipboard.

- "**Copy Name**" copies the expression (or name of the local variable in the locals window)
- "**Copy Value**" copies the shown value
- "**Copy raw value**" (*locals windows*) is for strings.
  It copies an non-escaped version of the string to the clipboard.

In **Lazarus 3.0** this is also available in the watch window.

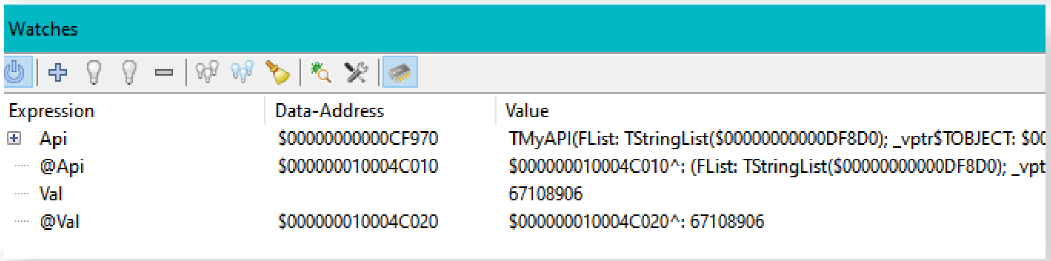
- "**Copy data address**" (**Lazarus 3.0**) copies the data address (*see section below*).
- "**Copy entire entry**" (**Lazarus 3.0**) copies name, value and address.

TAKING A HINT

Variables and expressions can also be evaluated directly from the Source editor by hovering the mouse, until the **IDE** displays a hint.
The **IDE** will try to establish the correct bounds of the expression. Usually the word under the mouse, but if that is a field the **IDE** will find the dot and use the full "**object.field**". Equally, if the mouse is at the end of a bracket or parenthesis the **IDE** will find its begin and even include any type cast if present.
The **IDE** will not call functions for hint evaluation. But it can (*and does by default*) cast any object to the instance's class. That can be toggled in the global options (**Editor > Completion and Hints**).

DATA ADDRESS (**Lazarus 3.0**)

The watches window has a column "**Data Address**".
This column is used for pointer types, including types that have an internal pointer.
`(T)Objects, AnsiStrings, dynamic arrays` all have an internal pointer. That pointer is stored in the variable. Using `@variable` will get the address of the variable, that is the address where the pointer is stored. The data itself is at a different address, and that address is shown in the "**Data address**" column. That address is the value you would get if you evaluated "**Pointer(variable)**"



```
Expression      Data-Address         Value
⊞ Api           $00000000000CF970    TMyAPI(FList: TStringList($00000000000DF8D0); _vptr$TOBJECT: $00
  @Api          $000000010004C010    $000000010004C010^: (FList: TStringList($00000000000DF8D0); _vpt
  Val                                67108906
  @Val          $000000010004C020    $000000010004C020^: 67108906
```

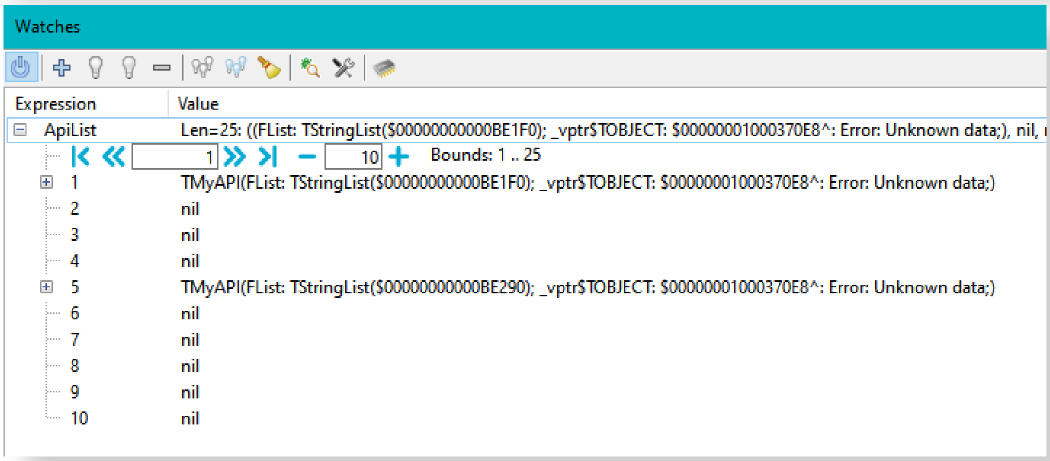
The variable for the object "**Api**" is at address `$10004C010`. The data for the object is at Address `$0000AE290`.
The "**Data address**" of a `Pointer` is the same as the value of the pointer. A `pointer` to Address 123 has its data at that address.
So for "`@Api`" the "**Data address**" is the same as it's value. The debugger additionally shows the dereferenced value. At the Address `$10004C010` the memory contains the data `$0000AE290`, but that is not shown, because the debugger knows the type and displays the array instead.
"**Val**" does not have a "**Data address**" (*that is, it is the same as the variables address*).
Its data is stored directly in the variable. "**Val**" does of course have an address, which "`@Val`" will show us. And here the "**Data address**" is the address in which the value
`67108906 (= $0400002A)` is stored.
The "**Data address**" can be used to determine if two **strings/arrays/objects** are the same, or merely have identical content.

STRUCTURES AND ARRAYS (LAZARUS 3.0)

Earlier in this article we saw how structured types can be expanded in the watches view.
The watch window also has the ability to expand the entries in an Array.
Lets say we had the following code:

```
var
 ApiList: Array [1..25] of TMyAPI;
begin
 ApiList[1] := TMyAPI.Create;
 ApiList[5] := TMyAPI.Create;
 ApiList[15] := TmyAPI.Create;
```

Then the watches could show us


Watches

Expression	Value
ApiList	Len=25: ((FList: TStringList($00000000000BE1F0); _vptr$TOBJECT: $00000001000370E8^: Error: Unknown data;), nil,
1 … 10	Bounds: 1 .. 25
1	TMyAPI(FList: TStringList($00000000000BE1F0); _vptr$TOBJECT: $00000001000370E8^: Error: Unknown data;)
2	nil
3	nil
4	nil
5	TMyAPI(FList: TStringList($00000000000BE290); _vptr$TOBJECT: $00000001000370E8^: Error: Unknown data;)
6	nil
7	nil
8	nil
9	nil
10	nil

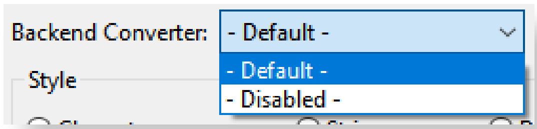
The content of the array will be paginated. So if the array had 1000 entries, it wouldn't force us to scroll through a long list. It allows us to go to the page we need, and see the correct selection of entries.

If we need more than 10 entries, we can specify a different page size in the 2nd edit field.

Additionally, the window allows us to expand nested values. Since the array contains objects, we can unfold the structure of those objects. We can explore to any depth that we need.
If we are interested in a particular value in the list, we can drag and drop that value to a position in the main list of watches (*outside the unfolded area*). This automatically creates a new watch like "ApiList[5]". This also works for nested entries, so after expanding "**ApiList[5]**", we could select and drag/drop "**ApiList[5].FList**".

CONVERTER (LAZARUS 3.0)

There is one more addition in the watches properties: Converter.



Backend Converter: - Default -

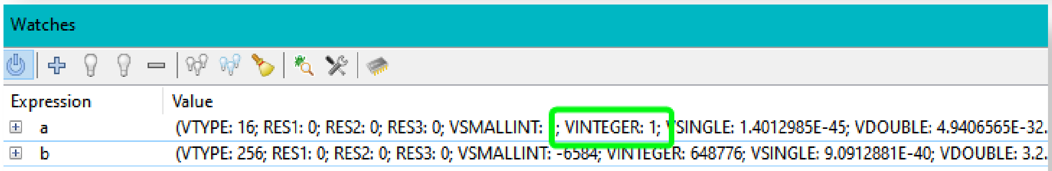
- Default -
- Disabled -

Style

By default, it does not offer much. Converters must first be configured, either in the global options, or in the project options.
We will look at one converter as example, which is a helper to display "variants". Before we do this, we will have a look at how the debugger shows us a variable of type variant.
Considering the code:

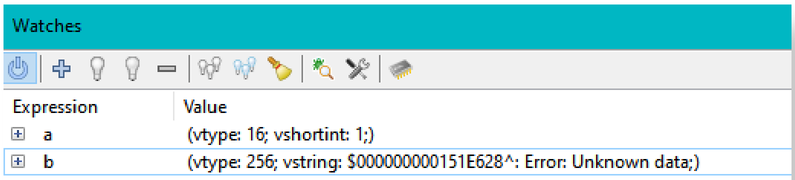
```
var
  a, b: variant;
begin
  a := 1;
  b := 'abc';
```

With **DWARF 2** the debugger shows



```
Expression   Value
a            (VTYPE: 16; RES1: 0; RES2: 0; RES3: 0; VSMALLINT: ; VINTEGER: 1; VSINGLE: 1.4012985E-45; VDOUBLE: 4.9406565E-32.
b            (VTYPE: 256; RES1: 0; RES2: 0; RES3: 0; VSMALLINT: -6584; VINTEGER: 648776; VSINGLE: 9.0912881E-40; VDOUBLE: 3.2.
```

Using **DWARF 3** we get



```
Expression   Value
a            (vtype: 16; vshortint: 1;)
b            (vtype: 256; vstring: $000000000151E628^: Error: Unknown data;)
```

In both cases the string value for "**b**" is not shown. This is because a variant is declared as

```
        vtAnsiString   : (VAnsiString: Pointer);
```

And the compiler does only include "**Pointer**" in the debug info.
To remedy this we use a converter.
This can be set up in either the global options or for a single project in the project options.
In each dialog, there is a page **Debugger** → **Backend Converte**r.
On this page, we press "**Add**" and give a name of our choosing (**e.g.** "**Variant**").
This will add an entry to the checklist box on the top. This entry must be checked.

In the field “Match types by name” we must enter a line with “variant”. This means that this will apply to variables of the type named “variant”.

And we need to select what the converter should do. This is the “Action” drop-down, and we choose “Convert variant to value type”.

> **What does the converter do?** Well, in general a converter can perform any translation on the result value (or even parts of it, like fields in an object). A converter works in the debugger backend, so it can also retrieve additional data.

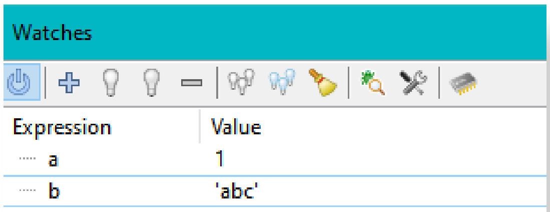
The “**Convert variant to value type**” specially deals with the **FPC** type “`variant`”.
It has additional knowledge on that type, like it knows that “`vtAnsiString`” is a string.

It only works on the fpc variant type. But since an application could define aliases, it is still necessary to specify the type name(s) to be matched, as we did above.
There is also a converter “`CallSysVarToLStr`” which deals with variants, but includes custom variants and executes code in the debugged app. So specifying type name(s) to match can also be used to configure different converters.

Now, that we configured a converter it will by default be applied to all watches that return (*or contain as field/entry with*) variant data. Individual watches can be excluded by disabling the converter in their properties.

If we had unchecked the converter in the checklist box, then it would not be used by default. It would only be used by watches for which it is selected in the drop-down of their properties. (*It will still check the type name*).

Now (*with either* **DWARF** *version*) the debugger will show us:



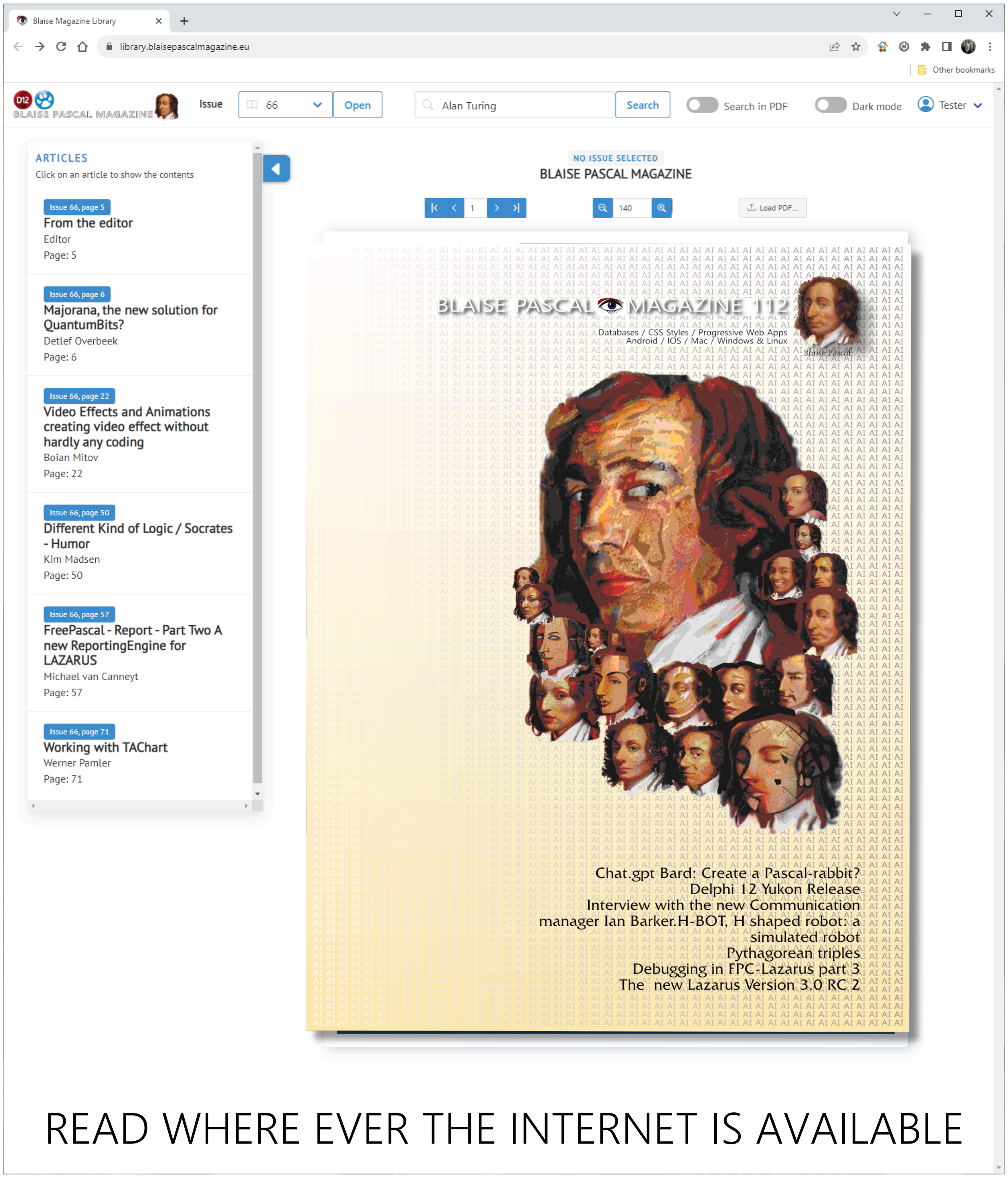
Expression	Value
a	1
b	'abc'

The converter will also apply in an array to each entry. Or in objects to each field of variant type.

THE NEW INTERNET BLAISE PASCAL LIBRARY 2023

https://library.blaisepascalmagazine.eu/

JUST OPEN ANY BROWSER (CHROME, SAFARI, EDGE, FIREFOX, OPERA, DUCKDUCKGO)
LOGIN: YOU WILL HAVE ALL ISSUES AVAILABLE - 6500 PAGES.
FOR ALL ISSUES STARTING AT NR1 UP TO THE LATEST ITEM.
YOU NEED A VALID SUBSCRIPTION: FREE - VALID THROUGH ONE YEAR



ARTICLES
Click on an article to show the contents

Issue 66, page 5
From the editor
Editor
Page: 5

Issue 66, page 6
Majorana, the new solution for QuantumBits?
Detlef Overbeek
Page: 6

Issue 66, page 22
Video Effects and Animations creating video effect without hardly any coding
Boian Mitov
Page: 22

Issue 66, page 50
Different Kind of Logic / Socrates - Humor
Kim Madsen
Page: 50

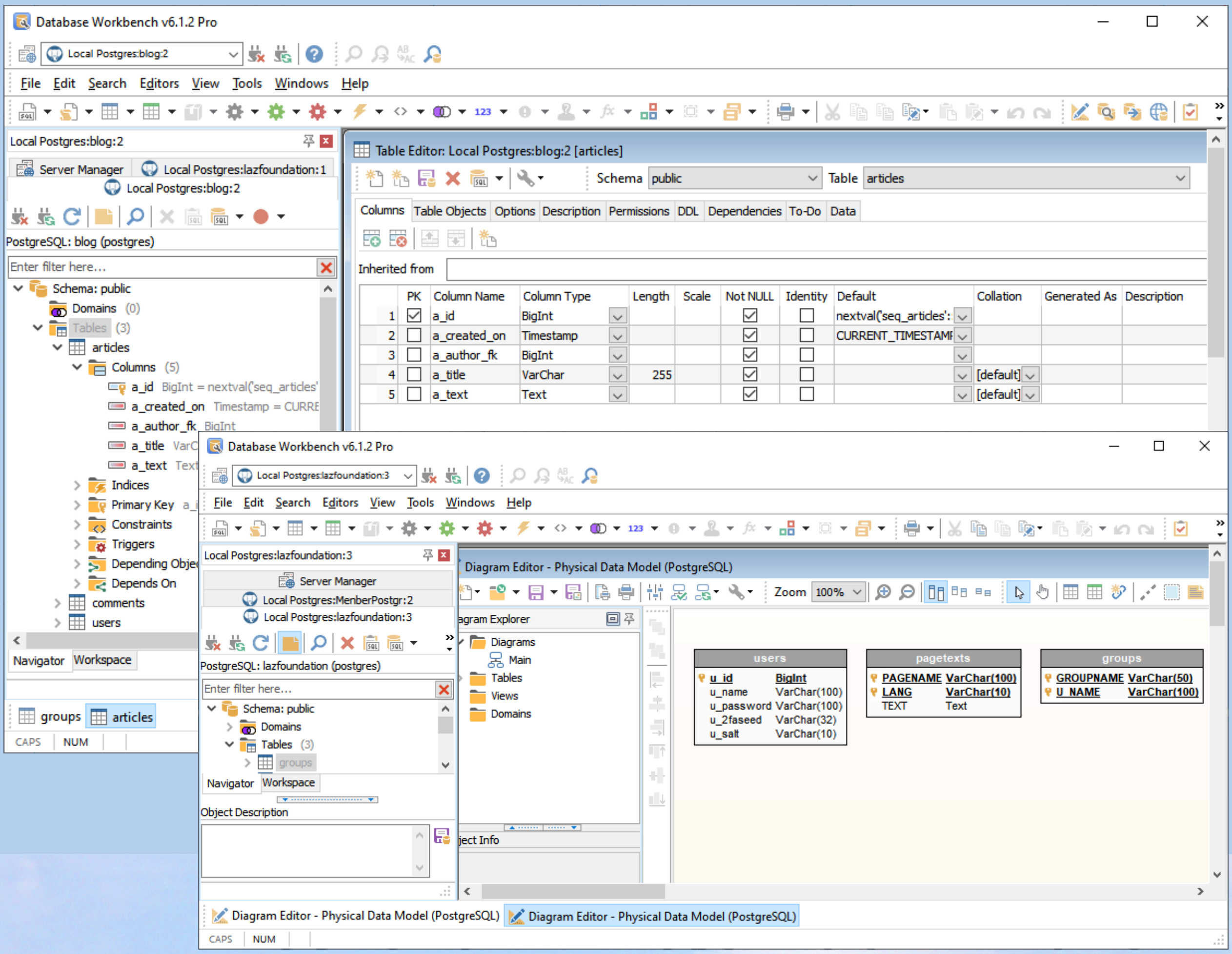
Issue 66, page 57
FreePascal - Report - Part Two A new ReportingEngine for LAZARUS
Michael van Canneyt
Page: 57

Issue 66, page 71
Working with TAChart
Werner Pamler
Page: 71

BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

READ WHERE EVER THE INTERNET IS AVAILABLE


Database Workbench v6.1.2 Pro
Local Postgres:blog:2
File Edit Search Editors View Tools Windows Help
Server Manager
Local Postgres:lazfoundation:1
PostgreSQL: blog (postgres)
Enter filter here...
Schema: public
Domains (0)
Tables (3)
articles
Columns (5)
a_id BigInt = nextval('seq_articles'
a_created_on Timestamp = CURRE
a_author_fk BigInt
a_title VarC
a_text Text
Indices
Primary Key
Constraints
Triggers
Depending Objec
Depends On
comments
users
Navigator Workspace
groups articles
CAPS NUM
Table Editor: Local Postgres:blog:2 [articles]
Schema public Table articles
Columns Table Objects Options Description Permissions DDL Dependencies To-Do Data
Inherited from
Local Postgres:lazfoundation:3
Local Postgres:MenberPostgr:2
PostgreSQL: lazfoundation (postgres)
Object Description
Diagram Editor - Physical Data Model (PostgreSQL)
Zoom 100%
agram Explorer
Diagrams
Main
Tables
Views
Domains
ject Info

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

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

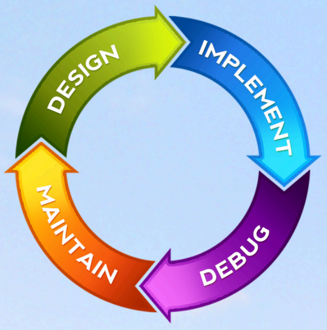
pagetexts	
PAGENAME	VarChar(100)
LANG	VarChar(10)
TEXT	Text

groups	
GROUPNAME	VarChar(50)
U_NAME	VarChar(100)

Introducing

Database Workbench 6

database development environment



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

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

Upscene *Database tools for developers*

www.upscene.com

SENDING DEBUG LOGS TO THE SERVER IN PAS2JS

BY MICHAEL VAN CANNEYT

Starter Expert

● ABSTRACT
In this article we show how to use a ready-to-use mechanism for sending debug logs from a **Pas2js** program to a HTTP server application written in **Free Pascal**

a better solution to gather debug info is to send it directly to the webserver

❶ INTRODUCTION

In an ideal world, the application runs smoothly, and all eventualities during execution of a program are handled gracefully. If this were so, debug logging and unexpected error handling can be stripped once the program is ready for shipment.
In reality, programs or their execution environment are not perfect, and users do unexpected and unforeseen things: for these two reasons, often you must still have debugging logs in shipped applications.
In the browser, the all-time **Pascal** '`WriteLn()`' statement can be used to write to the browser console. If so desired, the result can be shown in the HTML. But as soon as the user closes the browser window, this information is lost. For most users, finding and transmitting the information in the browser console at the request of a support team is a difficult, not to say impossible task.
Therefor a better solution to gather debug info is to send it directly to the webserver, where the logs can be examined at once or saved to be examined later on.
In this article, we demonstrate a mechanism for transmitting such debug information.
This mechanism is included by default in **Free Pascal** and **Pas2js**: debugcapture.

❷ ARCHITECTURE

The debug capture functionality - naturally - consists of 2 parts: one part is included in fcl-web, the other is part of **Pas2js**, and is used in a **Pas2js** client program.
The `fpDebugCaptureSvc` unit is part of **Free Pascal**'s `fpweb` package for making HTTP server applications: You can include it in a HTTP server program and with a single line of code activate it.
It is included by the simpleserver application by default. The compileserver program included in **Pas2js** also provides this functionality.
The functionality is disabled by default, the -u command-line switch must be provided to enable the debug capture: if no extra argument is given the captured info is printed on the console.
If an extra argument is given to the -u option , it is interpreted as a filename in which to save the output. The URL for the service is /debugcapture/ by default.
The client part is contained in the debugcapture unit, part of **Pas2JS**.
It contains a simple client component that sends the output to a configurable URL.

We'll demonstrate the use of both sides in the rest of this article.

❸ THE SERVER PART: TDEBUGCAPTURESERVICE

The `fpDebugCaptureSvc` unit contains a `TDebugCaptureService` component. It can be used to handle one or more HTTP routes. It can log to console or file by default, but additional backends can be registered. This component has the following declaration:

```
TDebugCaptureHandler = Procedure (aSender : TObject; aCapture : TJSONData) of object;
TDebugCaptureLogHandler =Procedure (EventType : TEventType; const Msg : String) of object;

TDebugCaptureService = class(TComponent)
  class Property Instance : TDebugCaptureService;
  class function JSONDataToString(aJSON: TJSONData): TJSONStringType;
  Procedure HandleRequest(ARequest: TRequest; AResponse: TResponse);
  Procedure RegisterHandler(const aName : String;aHandler: TDebugCaptureHandler);
  Procedure UnregisterHandler(const aName : String);
  Property LogFileName : string;
  Property LogToConsole : Boolean;
  Property CaptureToErrorLog : Boolean;
  Property OnLog : TDebugCaptureLogHandler;
  Property CORS : TCORSSupport;
end;
```

The following methods exist:

- **`HandleRequest`**
  This is the entry point of the service: the signature of this method is such that it can be used as the handler of a route in the `fpWeb` server's **HTTP** router.
- **`RegisterHandler`**
  You can add as many handlers for a debug capture request as you want.
  You register a callback `aHandler` with a (*unique*) name `aName`. The name is used in log messages when appropriate, and can be used to unregister the handler.
- **`UnregisterHandler`**
  can be used to unregister a handler with given name from the list of debug capture handlers.
- **`JSONDataToString`**
  this class method can be used to convert the **JSON** payload to a string. It will take case of special cases such as null or objects.

The following properties can be used:

- **`Instance`**
  This is a global instance of the component. This can be used for quickly setting up an instance of the debug capture service.
- **`LogFileName`**
  When set to a non-empty, logging captured debug output to file is enabled.
- **`LogToConsole`**
  When set to True a non-empty, captured debug output is sent to the console.
- **`CaptureToErrorLog`**
  When set to True, output is sent to the `OnLog` log handler together with error messages from the component.
- **`OnLog`**
  This event is used to log error messages from the component: when an error happens during writing of debug output to one of the handlers, it is logged using this event.
  If `CaptureToErrorLog` is set to `true`, all captured debug output is also sent to this event.
- **`CORS`**
  This can be configured to handle **CORS** preflight requests, enabling you to run the debug capture service on a different URL from where your application is served. Make sure you configure CORS correctly if you enable it, it is a bad idea to allow all possible domains to use this service.

The 3 standard logging mechanisms (`file, console, errorlog`) use the `RegisterHandler` and `UnregisterHandler` calls, so they are called in the same manner as your own handlers.
Any errors when writing to file or console will therefor also be reported using the standard log mechanism.
The use of this component is very simple. The following little program is a complete webserver that also has the `debugcapture` output.
It overrides 2 methods of the standard `TCustomHTTPApplication` class to provide logging and to configure the server:

```
program demosvr;
uses
  custhttpapp, sysutils, Classes, jsonparser, fpjson, httproute,
  httpdefs, fpmimetypes, fpwebfile, fpwebproxy, fpdebugcapturesvc;

Type
  { THTTPApplication }

  THTTPApplication = Class(TCustomHTTPApplication)
  private
    procedure HandleCaptureOutput(aSender: TObject; aCapture: TJSONData);
  published
    procedure DoLog(EventType: TEventType; const Msg: String); override;
    Procedure Initialize; override;
  end;

procedure THTTPApplication.DoLog(EventType: TEventType; const Msg: String);
begin
 Writeln(FormatDateTime('yyyy-mm-dd hh:nn:ss.zzz',Now),' [',EventType,'] ',Msg)
end;
```

Here we have done nothing yet except define our class and implement logging.
The override of the `DoRun` method is where the magic happens: the standard instance of the `TDebugCaptureService` is used to provide the debug capture functionality.
It is configured to send the debug output to the console and to a file called `debug.log` by setting the `LogToConsole` and `LogFileName` properties:

```
procedure THTTPApplication.Initialize;
var
  aBaseDir : String;
  Svc : TDebugCaptureService;
begin
  Port:=8080;
  Svc:=TDebugCaptureService.Instance;
  Svc.OnLog:=@DoLog;
  Svc.LogFileName:='debug.log';
  Svc.RegisterHandler('log',@HandleCaptureOutput);
  HTTPRouter.RegisterRoute('/debugcapture',rmPost,@Svc.HandleRequest,False);
  aBaseDir:=IncludeTrailingPathDelimiter(GetCurrentDir);
  TSimpleFileModule.RegisterDefaultRoute;
  TSimpleFileModule.BaseDir:=aBaseDir;
  TSimpleFileModule.OnLog:=@Log;
  TSimpleFileModule.IndexPageName:='index.html';
  MimeTypes.LoadKnownTypes;
  inherited;
end;
```

After registering the `/debugcapture` route, the standard `TSimpleFileModule` component is used to provide standard **HTTP** file serving from the current directory.

NOTE that we will not use the standard mechanism to log to console, instead, we implement our own handler: `HandleCaptureOutput`, which we register with the name Log.
(*the names for the internal logging mechanisms all start with* `$`, *do not use this character in your own handlers*)

The `HandleCaptureOutput` method uses the `JSONDataToString` class method to create a `string` and logs it using the standard `DoLog` method of the application class.

```
procedure THTTPApplication.HandleCaptureOutput(aSender: TObject; aCapture: TJSONData);
begin
  DoLog(etDebug,TDebugCaptureService.JSONDataToString(aCapture));
end;
```

As a result, the debug info and the info about served pages is displayed in the same uniform manner.
With this, the application class is ready, all that needs to be done is to start it:

```
Var
  Application : THTTPApplication;
begin
  Application:=THTTPApplication.Create(Nil);
  Application.Initialize;
  Application.Run;
  Application.Free;
end.
```

And so, with 20 lines of code, we have created a HTTP server that also acts as a receiver of debug log info.

❹ THE CLIENT PART: TDEBUGCAPTURECLIENT

In **Pas2JS**, the debugcapture unit provides the `TDebugCaptureClient` component.

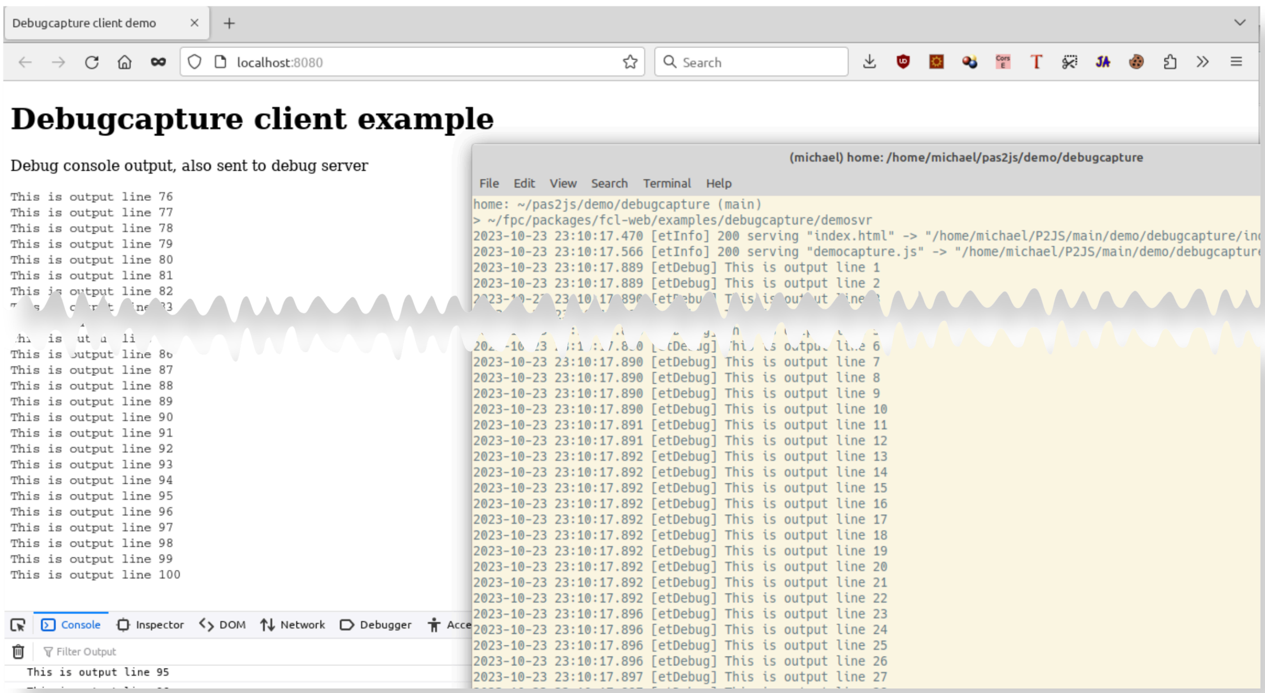
```
TDebugCaptureClient = class(TComponent)
Public
  Class property Instance : TDebugCaptureClient Read _Instance;
  Procedure Capture(const aLine : String; NewLine : Boolean = True); virtual;
  Procedure Flush;
  Procedure SetConsoleHook;
  Procedure ClearConsoleHook;
  Property URL : String;
  Property BufferTimeout : Integer;
  Property HookConsole : Boolean;
end;
```

The following methods exist:

- **`Capture`**
  This is the central call: the `string aLine` is sent to the server. If `NewLine` is set to `True`, a `newline` character is added.
- **`Flush`**
  if the `BufferTimeout` is set to a positive number, lines will be buffered till the indicated timeout is reached. `Flush` will empty the buffer and send the contents to the server.
- **`SetConsoleHook`**
  When calling this, the console hook will be installed, which means that all `Write(Ln)` statements will be written to the debug capture output as well. if a previous console hook was present, it will also be called.
- **`ClearConsoleHook`**
  Resets the console hook to the state previous to calling `SetConsoleHook`
- **`SetExceptionsHook`**
  When calling this, the `OnShowException` hook in `SysUtils` will be installed, which means that all calls to `Show-exceptions` will write to the debug capture output as well. If a previous console hook was present, it will also be called.
- **`ClearExceptionsHook`**
  Resets the exceptions hook to the state previous to calling `SetExceptionsHook`

In addition, the following properties exist:

- **`Instance`**
  This class property provides a standard instance, which is ready for you to configure and use.
- **`URL`**
  The URL to which all debug output is sent. The default URL is '`/debugcapture`'.
- **`BufferTimeout`**
  A time (*in milliseconds*) during which log output is buffered locally before sending it to the server. If set to 0, then no buffering takes place, all logging is sent to the server immediately.
- **`HookConsole`**
  If set to `True`, then `SetConsoleHook` is called. If set to `False`, `ClearConsoleHook` is called.
- **`HookExceptins`**
  If set to `True`, then `SetExceptionsHook` is called. If set to `False`, `ClearExceptionsHook` is called.



Debugcapture client example

Debug console output, also sent to debug server

```
This is output line 76
This is output line 77
This is output line 78
This is output line 79
This is output line 80
This is output line 81
This is output line 82
This is output line 83
This is output line 85
This is output line 86
This is output line 87
This is output line 88
This is output line 89
This is output line 90
This is output line 91
This is output line 92
This is output line 93
This is output line 94
This is output line 95
This is output line 96
This is output line 97
This is output line 98
This is output line 99
This is output line 100
```

```
This is output line 95
```

(michael) home: /home/michael/pas2js/demo/debugcapture

```
home: ~/pas2js/demo/debugcapture (main)
> ~/fpc/packages/fcl-web/examples/debugcapture/demosvr
2023-10-23 23:10:17.470 [etInfo] 200 serving "index.html" -> "/home/michael/P2JS/main/demo/debugcapture/in
2023-10-23 23:10:17.566 [etInfo] 200 serving "democapture.js" -> "/home/michael/P2JS/main/demo/debugcapture
2023-10-23 23:10:17.889 [etDebug] This is output line 1
2023-10-23 23:10:17.889 [etDebug] This is output line 2
2023-10-23 23:10:17.890 [etDebug] This is output line 3
2023-10-23 23:10:17.890 [etDebug] This is output line 6
2023-10-23 23:10:17.890 [etDebug] This is output line 7
2023-10-23 23:10:17.890 [etDebug] This is output line 8
2023-10-23 23:10:17.890 [etDebug] This is output line 9
2023-10-23 23:10:17.890 [etDebug] This is output line 10
2023-10-23 23:10:17.891 [etDebug] This is output line 11
2023-10-23 23:10:17.891 [etDebug] This is output line 12
2023-10-23 23:10:17.892 [etDebug] This is output line 13
2023-10-23 23:10:17.892 [etDebug] This is output line 14
2023-10-23 23:10:17.892 [etDebug] This is output line 15
2023-10-23 23:10:17.892 [etDebug] This is output line 16
2023-10-23 23:10:17.892 [etDebug] This is output line 17
2023-10-23 23:10:17.892 [etDebug] This is output line 18
2023-10-23 23:10:17.892 [etDebug] This is output line 19
2023-10-23 23:10:17.892 [etDebug] This is output line 20
2023-10-23 23:10:17.892 [etDebug] This is output line 21
2023-10-23 23:10:17.892 [etDebug] This is output line 22
2023-10-23 23:10:17.896 [etDebug] This is output line 23
2023-10-23 23:10:17.896 [etDebug] This is output line 24
2023-10-23 23:10:17.896 [etDebug] This is output line 25
2023-10-23 23:10:17.896 [etDebug] This is output line 26
2023-10-23 23:10:17.897 [etDebug] This is output line 27
```

The use of this component is again quite straightforward, as shown by the following example program:

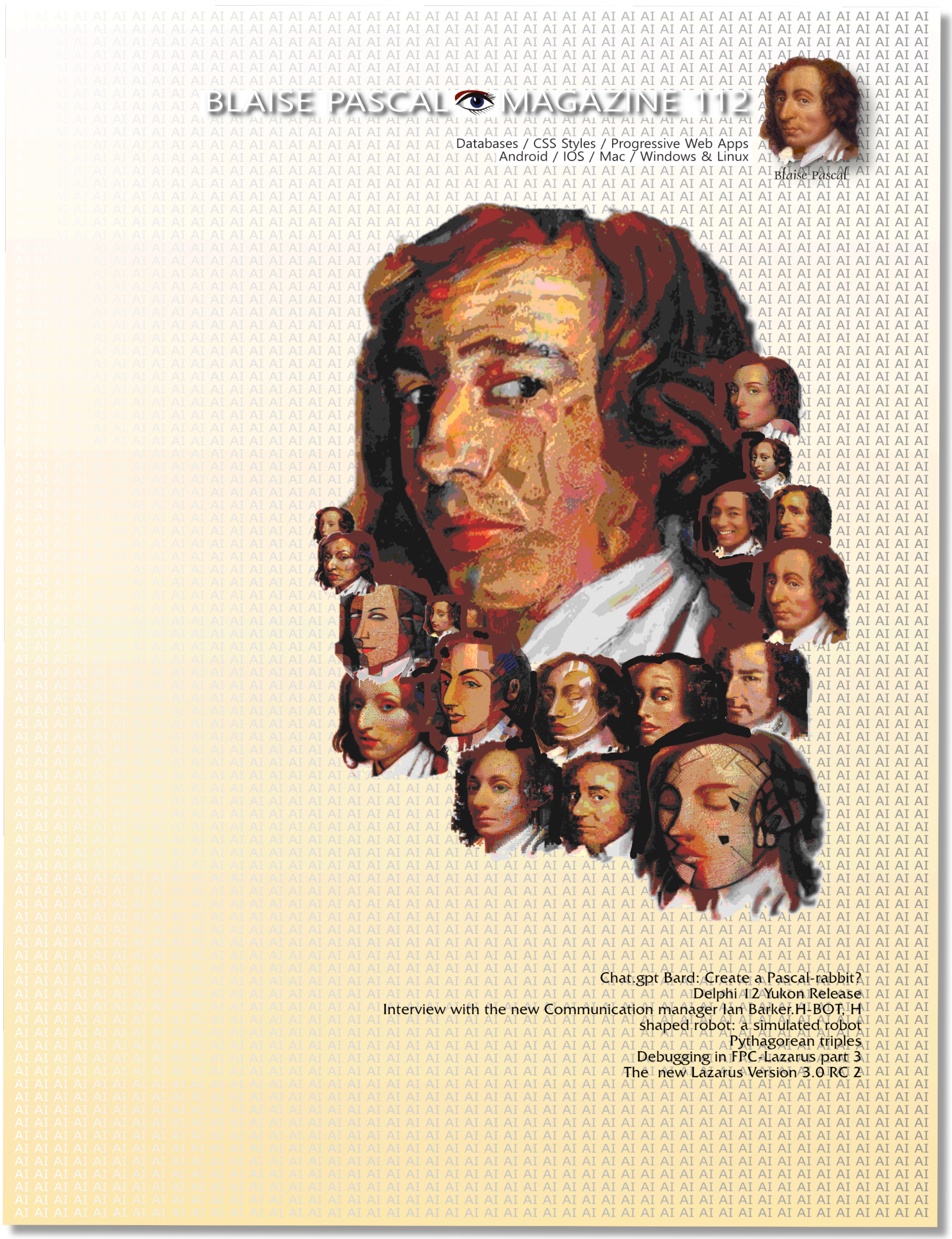
```pascal
program democapture;
{$mode objfpc}
{$h+}
uses
   sysutils, classes, browserconsole, debugcapture;
Var
    I : integer;
begin
   With TDebugCaptureClient.Instance do
      begin
        BufferTimeout:=100;
        HookConsole:=True;
      end;
   For I:=1 to 100 do
      Writeln('This is output line '+IntToStr(I))
end.
```

The result of the 2 programs combined is shown in figure 1 on page 6 of this article, page 75.
In the background, the browser is visible with the output of the `WriteLn` statements as **HTML** and in the browser debug console. In the foreground, the console on which the **HTTP** server program was started is visible. It shows the URLs that were loaded, and the debug capture output.

❺ CONCLUSION

Free Pascal and Pas2JS come equipped with simple tools to enable you to to collect debugging information from applications in production.
As shown here, the code to achieve this is really simple, and the classes used in the process are easily extended with extra functionalities: you can add a threaded mechanism on the server for
improved performance, you can store the logs in a database, send them to logstash, all with a single mechanism which also works out of the box without the need for extra code.

ADVERTISEMENT


BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
Blaise Pascal
Chat.gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2

SUBSCRIPTION FOR 2 YEAR
BLAISE PASCAL MAGAZINE
€128,44 ex Vat

ONLY AT BARNSTEN UPTO 30 % DISCOUNT ENDING 10 NOVEMBER

UPDATE TO NEXT VERSION INCLUDED

barnsten

Tel.: +31 23 542 22 27 Web: www.barnsten.com
Info: info@barnsten.com

BY MICHAEL VAN CANNEYT

Starter Expert

Webassembly was mentioned in other articles
issue 77 page 43 / issue 83 /17 and issue 101/82

IT HAS FINALLY ARRIVED: NOW WE CAN USE IT

ABSTRACT
Webassembly was designed to run in the browser. It's design is focused on simplicity and safety, making it ideal for **sandboxing**. As a result more and more it finds its way in applications that run outside the browser.
In this article we show how to embed a **Webassembly** Module in a **Free Pascal** module.

❶ INTRODUCTION

WebAssembly is a an open bytecode format similar in purpose to the **Java** and **C#** bytecode formats: it is designed to run in a sandboxed environment.

The initial target of this was the browser, where it allows computationally intensive tasks to be run in the browser at speeds that are vastly superior to the speed of plain **javascript**.

Today, you can compile from any programming language (Notably **C, C++, Rust** and of course **Pascal**) to the webassembly format: the **LLVM** compiler supports **WebAssembly** as an output format.

The specification of this bytecode format is open and managed by the **W3C** consortium:
`https://www.w3.org/TR/wasm-core-2/`

The specification is maintained on github:
`https://github.com/WebAssembly/design`

Beside the core specification, which describes the basic bytecode format and the supported assembly instructions, there are also various extensions.

A list of extensions and their various stages of implementation can be found on github as well:
`https://github.com/WebAssembly/proposals`

Some of the more interesting ones are threading and exception support.
The open character of the format means anyone can implement a runtime that loads and execute the format. In fact, outside the browser, several wasm execution environments exist:

WasmTime This engine is maintained by the **Bytecode Alliance**, which actively supports the development of **WebAssembly**. It is a conservative implementation, meaning that it only supports established proposals of the WebAssembly specification.

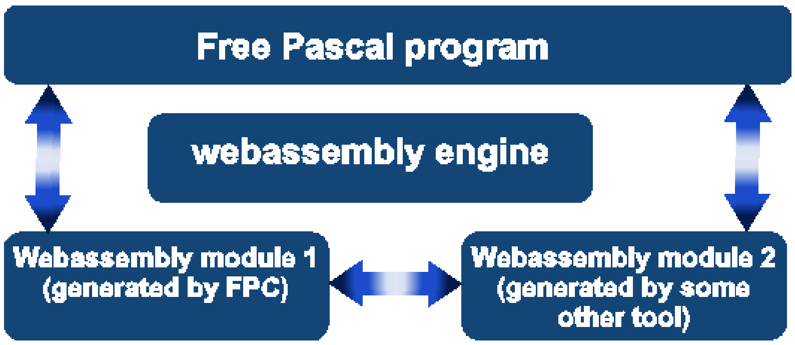

Free Pascal program
webassembly engine
Webassembly module 1 (generated by FPC)
Webassembly module 2 (generated by some other tool)

Figure 1: Using a webassembly engine to run a WebAssembly program in a FPC program

https://wasmtime.dev/

WasmEdge
This engine is maintained by an independent community of developers and was recently brought under the umbrella of the **Cloud** native computing foundation

https://cncf.io/, itself part of the **Linux foundation**.

This implementation is more cutting edge,
it supports many of the more experimental **WebAssembly** extensions.

https://wasmedge.org/

Wasmer is an independent implementation of an webassembly bytecode engine.
Like **wasmedge**, it is more accepting of new proposals:

https://wasmer.io/

It has adopted an approach similar to npm (*the node package manager*):
it has a package system with ready-to-run webassembly modules.

WAMR is another bytecode alliance implementation of the webassembly runtime, focused on small memory footprint and fast execution:
https://github.com/bytecodealliance/-micro-runtime

All these implementations have a library which you can use to embed the engine in your application: this means you can run a webassembly program (*which can consist of multiple webassembly modules linked together*) embedded in your **Free Pascal program**.

This embedded **webassembly** program can also be generated by **Free Pascal** or by some other programming tool or - most likely - a combination of both:

All engines support linking together various **webassembly** modules, regardless of the language they were originally programmed in. The format used by **webassembly** ensures that all modules use the same format to exchange data and code.

This is shown diagrammatically in figure 1 on page 1 of this article: a native **Free Pascal** host program loads 2 **webassembly** modules: one written in **Free Pascal** and one written in another language, and can execute functions in both modules. The modules themselves can also call functions in each other.

Of these, the **wasmtime** and **wasmedge** engines have at least some comprehensive documentation, and therefore import units for these libraries have been created for **Free Pascal**, which we will demonstrate here.

❷ THE WASI SPECIFICATION

The WebAssembly specification by itself does not specify how to interact with the environment: The format does not describe how to read and write files, get the time and so on.

It only describes a mechanism how to import functions from the runtime environment. Obviously, it also specifies how to execute functions in the **webassembly**.

This allows for modules to be chained together, just as dynamically loadable libraries.

Naturally, a bytecode format that cannot interact with the environment is of little use. Therefore a separate specification was developed which provides a minimal list of functions needed to interact with the outside world: **WASI**:
the **WebAssembly System Interface**.

`https://github.com/WebAssembly/WASI`

All engines specified above support this interface.
This means that when a **webassembly** module is loaded into one of these engines, the functions listed in **WASI** are available.
It serves as the basis for a **LibC** implementation that runs in a **webassembly** environment.

The **WASI** current specification has only basic **OS** interaction support:
only **basic file I/O** and getting the time and environment variables.
That means **no graphical environment, no TCP/IP or HTTP** environment etc.
(*The latter are however expected to appear in version 2 of the spec*)

In essence, the spec provides enough calls to implement the `SysUtils` unit in **Free Pascal**, and that is what has been used to develop the **Free Pascal WebAssembly** target.

Putting all this together, it means basic **Free Pascal** programs can be loaded into one of the engines mentioned above.

Does this mean you cannot run more advanced code (*requiring sockets, UI etc.*) in these runtimes?
No, of course not: the **engines support providing your own functions** to the **webassembly**.
That means that if you provide functions to execute a HTTP request, then these functions can be executed from inside the runtime to download HTML pages.
It should be noted that you must be careful with the functionality you provide to the **webassembly**: functions open doors to the environment which can potentially be exploited.

❸ USING WASMTIME

Wasmtime is available as a command-line tool with which you can start a webassembly module from the command-line. This command-line tool itself is simply a shell around the wasmtime dynamically loadable library.
Instructions for downloading and installing wasmtime can be found here

`https://docs.wasmtime.dev/cli-install.html`

Binaries of the releases for all major platforms can be found here:

`https://github.com/bytecodealliance/wasmtime/releases`

Free Pascal contains a unit called wasmtime which can be used to access the functionality of the wasmtime library. The library is loaded at runtime with the `LoadWasmTime` call:

```
Procedure LoadWasmTime(const Lib : string);
```

The argument is the name of the library file to load. The name of the library as distributed is available in the `libwasmtime`) constant. If the loading fails, an exception is raised.

The library exposes well over 100 types and 500 functions, this is clearly more than can be explained in the context of a single article. Therefore we'll describe a simple example which demonstrates how to load a library, make a host-provided function available to the **wasm** module, and show how this function is called.

The **webassembly** program which we will be loading and executing is quite simple:

```
(module
  (func $hello (import "" "hello"))
  (func (export "run") (call $hello))
)
```

Without going into the details of the webassembly text format, it is apparent from the text that this module imports a function called `hello` (*with no parameters and no return value*) and exports a function called `run` which simply calls the imported "hello" function.
The **run** function again has no parameters and return value.

Note that no **file IO** or other external functions are used: just one imported function and one exported function. There is also no initialization or finalization code. To execute this **webassembly** program, we need therefore to load this file, convert it to bytecode, provide it with a **hello** function and then call the **run** function. The expectation is that the **hello** function is called, and that the **run** function returns immediately afterwards. The main program starts by declaring a lot of variables:

```
Var
  engine : Pwasm_engine_t = Nil;
  store  : Pwasmtime_store_t = Nil;
  context : Pwasmtime_context_t = Nil;
  F    : TMemoryStream;
  wat : Twasm_byte_vec_t;
  wasm : twasm_byte_vec_t;
  module    : Pwasmtime_module_t = Nil;
  error     : Pwasmtime_error_t = Nil;
  hello_ty : Pwasm_functype_t = nil;
  hello     : Twasmtime_func_t;
  trap      : Pwasm_trap_t = Nil;
  instance : Twasmtime_instance_t;
  import    : Twasmtime_extern_t;
  run : Twasmtime_extern_t;
  OK   : Byte;
```

The meaning of these variables will be explained as we encounter them in the program code.
All types used in WasmTime are opaque record types: the exact details of the record are not exposed. Most of these records are created dynamically with a function that returns a pointer to such an opaque type, and as a rule the function name ends in (*or contains*) `new`. When you are done with a particular variable, you must release the memory occupied by the variable using a function whose name ends in `delete`.

The program of course starts by loading the wasmtime library. When this has succeeded, a webassembly engine is created using the wasm engine new function.

```
begin
  Writeln('Loading wasm library');
  Loadwasmtime('./'+libwasmtime);
  Writeln('Initializing...');
  engine := wasm_engine_new();
  store:=wasmtime_store_new(engine, nil,nil);
  context:=wasmtime_store_context(store);
```

The `store` is a general purpose memory area for the engine. It can be used to add user data, but is also used by the engine. The `context` is is a pointer used by the engine to add/remove data to the store.
The following piece of code will load a file containing a webassembly module using text representation (a kind of assembly language). It allocates a memory area (`wat`) using the `twasm_byte_vec_t` type (which represents a memory block) needed by the engine, and moves the contents of the file into it:

```
F:=TMemoryStream.Create;
try
  F.LoadFromFile('hello.wat');
  wasm_byte_vec_new_uninitialized(@wat, F.Size);
  Move(F.Memory^,wat.data^,F.Size);
finally
  F.Free;
end;
```

In the following step, the text representation of the webassembly module is converted to bytecode using `wasmtime_wat2wasm` and stored in a memory block wasm. (*again of type* `twasm_byte_vec_t`). The text representation of the module (**wat**) is disposed of.

```
Writeln('Compiling module...');
error:=wasmtime_wat2wasm(PAnsiChar(wat.data), wat.size, @wasm);
if (error<>Nil) then
  exit_with_error('failed to parse wat', error, Nil);
wasm_byte_vec_delete(@wat);
error:=wasmtime_module_new(engine, Puint8_t(wasm.data), wasm.size, @module);
wasm_byte_vec_delete(@wasm);

if (error <> nil) then
  exit_with_error('failed to compile module', error, nil);
```

After compiling the webassembly, the bytecode is loaded into a module (*module, of type* `Pwasmtime_module_t`), with the `wasmtime_module_new` function, and the bytecode representation is discarded.
The module is what will be used when executing the webassembly.

The `exit_with_error` function is an auxiliary function which will be used in several locations in the program. We'll come back to it later.

At this point we have a module, ready to be executed. We did not yet use the context which we created at the beginning. Now we get to the point where this context will be used: We will provide a pascal '**hello**' function to the webassembly module.

Functions provided to the webassembly module must have the appropriate function type

```
Twasmtime_func_callback_t = function (env:pointer;
                                      caller:Pwasmtime_caller_t;
                                      args:Pwasmtime_val_t;
                                      nargs:Tsize_t;
                                      results:Pwasmtime_val_t;
                                      nresults:Tsize_t):Pwasm_trap_t;cdecl;
```

The `env` argument can be used to pass information along, for example the `Self` pointer of an object. The `caller` contains information about the calling environment, and the `args` pointer points to the arguments passed during the call. The `nargs` parameter contains the number of arguments. Similarly, the `results` and `nresults` arguments are used to specify return values. The return value is a trap (*of type* `Pwasm_trap_t`): when non-nil, it signals an error condition to the **webassembly** engine.

Knowing this, our 'Hello' function looks like this:

```
function hello_callback(env : Pointer;
                 caller : Pwasmtime_caller_t;
                 args : pwasmtime_val_t;
                 nargs : size_t;
                 results : pwasmtime_val_t; nresults : size_t) : pwasm_trap
begin
  Writeln('Calling back...');
  Writeln(' Hello World!');
  Result:=Nil;
end;
```

The next part of our program is defining this function in the webassembly module, so it can be called. This starts by creating a function type (`hello_ty`, of type `Pwasm_functype_t`), which is then registered as a function using `wasmtime_func_new`.
A function type is represented by `Pwasm_functype_t`. This corresponds to a procedural type in pascal. The WebAssembly format defines a function type for all functions and procedures: For both internal and external functions, a function type must be defined.
For external (imported/exported) functions, this is logical: the runtime engine needs to know what data to provide or what date to extract whenever the boundary between webassembly and the host environment is crossed:
both when calling a webassembly function in a webassembly module and when an external function is called by the webassembly module.

To register a callable function, the `wasmtime_func_new` is used:

```
procedure wasmtime_func_new (
    store: Pwasmtime_context_t;
    _type: Pwasm_functype_t;
    callback: Twasmtime_func_callback_t;
    env: pointer;
    finalizer: TFinalizer;
    ret: Pwasmtime_func_t)
```

The second argument (`_type`) is the function type,
and the third (`callback`) is the actual function to call.

The `env` argument can be filled with anything you like, it will be passed as-is when the `callback` is called. This can be used for example to store an object pointer.
Finally a `finalizer` for `Env` can be specified, this is a function which is called typically to free the `env` object when the webassembly module is destroyed. The `ret` argument points to a location which will be filled with a function definition.
Since our function accepts no arguments and returns no results, the function type and the registration of the callback is quite simple:

```
Writeln('Creating callback...\n');
hello_ty:=wasm_functype_new_0_0();
wasmtime_func_new(context, hello_ty, @hello_callback, Nil, Nil, @hello);
```

NOTE the `context` argument and the `hello` variable which will contain the function definition as created by the wasm runtime.
What we did till now is define webassembly module, and the functions which we will be providing to it. It is ready to run.
To actually run a webassembly, we must create an instance of the module (it is possible to create multiple instances of a single module, and execute them in parallel).
From this instance we can then extract the address of the exported function ('run'), and call it.
To create an instance of a webassembly module, the wasmtime instance new function is used

```
function wasmtime_instance_new(
    store : Pwasmtime_context_t;
    module: Pwasmtime_module_t;
    imports : Pwasmtime_extern_t;
    nimports: Tsize_t;
    instance: Pwasmtime_instance_t;
    trap: PPwasm_trap_t):Pwasmtime_error_t
```

The store is the context we are using, the module is the module we just defined. As can be seen from the imports argument, we must provide it with all the functions that can be used.
If several instances can be created and run, it makes sense that the functions are provided to the instance, and not to the module: the env pointer for the callable functions will typically be different for each instance.
Upon successful return, instance will be filled with a runnable instance. trap will be filled with an error report if an error occurred.

Errors can happen for example when the module expects to be able to import functions foo and bar, but only bar is supplied.

In our case, we need to provide the hello function we just created:

```
Writeln('Instantiating module...');
import.kind:=WASMTIME_EXTERN_FUNC;
import.of_.func:=hello;
error:=wasmtime_instance_new(context, module, @import, 1, @instance, @trap);
if (error<>nil) or (trap <>Nil) then
    exit_with_error('failed to instantiate', error, trap);
```

The arguments to the wasmtime_instance_new function are the context, the module, 1 import definition, and 2 variables for return values error and trap, which we examine on return.

The function that is exported from the webassembly module is called 'run'.
We extract the function definition from the instance using the wasmtime_instance_export_get function:

```
function wasmtime_instance_export_get(
     store    : Pwasmtime_context_t;
     instance : Pwasmtime_instance_t;
     name     : PAnsiChar;
     name_len : Tsize_t;
     item     : Pwasmtime_extern_t):T_Bool;
```

The store and instance arguments are of course the context we are using and the instance we just created. The name and name_len functions are used to pass the name of the function you wish to have ('run' in our case) and the item is filled with the function definition on return: when the function exists, the function returns a nonzero return value.
So, our code to get the 'run' function definition is:

```
 Writeln('Extracting export...\n');
ok:=wasmtime_instance_export_get(context, @instance, PAnsiChar('run'), 3,
@run) ;
if OK=0 then
   exit_with_error('failed to get run export', nil, nil);
if run.kind<>WASMTIME_EXTERN_FUNC then
   exit_with_error('run is not a function', nil, nil);
```

The run variable holds a reference to the exported function.

Now we are ready to actually run the function. This is done with the wasmtime_func_call:

```
function wasmtime_func_call (
  store    : Pwasmtime_context_t;
  func     : Pwasmtime_func_t;
  args     : Pwasmtime_val_t;
  nargs    : Tsize_t;
  results  : Pwasmtime_val_t;
  nresults : Tsize_t;
  trap     : PPwasm_trap_t): Pwasmtime_error_t;
```

The first 2 arguments are the store context and the function definition we just extracted.
NOTE that the wasm module or instance do not need to be specified:
they are implicit in the function definition. The arguments to be provided to the called function and results returned by it, are specified in the next 4 arguments.
The last argument (`trap`) is used to hold an error condition when something goes wrong.
Since the 'run' function does not take arguments, and provides no results, we do not need to set up anything to specify them, so we are ready to call our function:

```
Writeln('Calling export...');
error:=wasmtime_func_call(context, @run.of_.func, nil, 0, nil, 0, @trap);
if (error<>nil) or (trap<>nil) then
  exit_with_error('failed to call function', error, trap);
```

The first thing to do is to check if an error was returned.

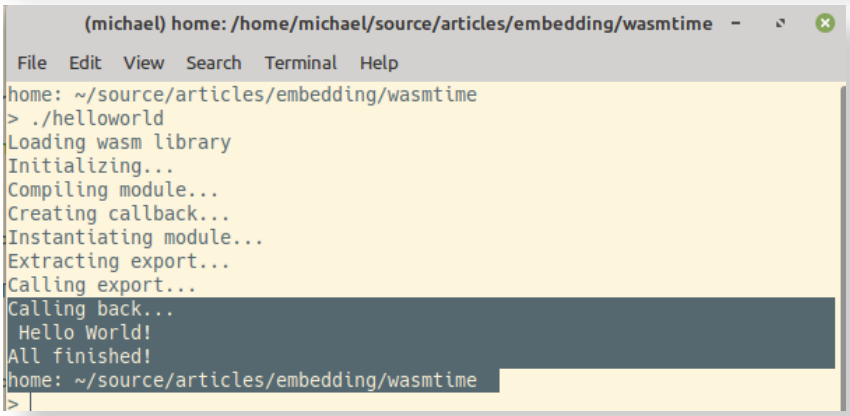
After all this, the 'run' function has been called, and the instance and module can be cleaned up.
To clean up, we clean up the module and the store context: everything connected to the store will also be cleaned up:

```
   Writeln('All finished!\n');
   wasmtime_module_delete(module);
   wasmtime_store_delete(store);
   wasm_engine_delete(engine);
end.
```

All that remains to be done is to show the `exit_with_error` procedure: This procedure shows the error information returned by the wasmtime engine, and demonstrates that you must release the `trap` and `error` runtime error reports when they occur.
Failure to do so will result in memory leaks:

```
procedure exit_with_error(message : PAnsiChar; error : Pwasmtime_error_t;
                                                        trap: Pwasm_trap_t);
var
  error_message : Twasm_byte_vec_t ;
  S : AnsiString;

begin
  Writeln(stderr, 'error: ', message);
  S:='';
  if (error <> Nil) then
    begin
      wasmtime_error_message(error, @error_message);
      wasmtime_error_delete(error)
    end
  else
    begin
      wasm_trap_message(trap, @error_message);
      wasm_trap_delete(trap);
    end;
  SetLength(S,error_message.size);
  Move(error_message.data^,S[1],error_message.size);
  Writeln(stderr, S);
  wasm_byte_vec_delete(@error_message);
  halt(1);
end;
```

With all this in place, we can now run the binary. If all goes well, you can see output similar to the one shown in figure 2 on page 10.


(michael) home: /home/michael/source/articles/embedding/wasmtime
File Edit View Search Terminal Help

```
home: ~/source/articles/embedding/wasmtime
> ./helloworld
Loading wasm library
Initializing...
Compiling module...
Creating callback...
Instantiating module...
Extracting export...
Calling export...
Calling back...
 Hello World!
All finished!
home: ~/source/articles/embedding/wasmtime
>
```

Figure 2: The first wasmtime example program in action

❹ PROVIDING A WASI ENVIRONMENT TO EXECUTE AN FPC GENERATED PROGRAM

The previous demonstration program only used an imported function (`hello`) and an exported function (`run`) to communicate with the outside world. In particular, it did not use any **WASI** functionality. The **webassembly RTL** of **Free Pascal** does use the **WASI** functionality. **wasmtime** does not make the **WASI** interface available to a **webassembly** module unless you instruct it to.
Since the **FPC RTL** for webassembly relies on the **WASI** interface, we'll execute a **FPC**-generated program to demonstrate how to provide the **WASI** functionality to a **webassembly** module.
The **FPC** program is a very simple 'Hello, world' :

```
begin
  Writeln('"Hello, World!" from FPC webassembly');
end.
```

When the **Free Pascal webassembly** compiler and **RTL** are installed, then compiling this program can be done so:

```
ppcrosswasm32 hello.pp
```

If all went well, it results in a `hello.wasm` webassembly module.
The following **Free Pascal** program will load the webassembly module and provide the **WASI** environment. The variable declaration block closely resembles the one in the previous example, we only list the additional variables that were not present in the previous program:

```
var
  linker  : Pwasmtime_linker_t;
  wasi_config : Pwasi_config_t;
begin
  Writeln('Loading wasm library');
  Loadwasmtime('./'+libwasmtime);
  Writeln('Initializing...');
  engine  := wasm_engine_new();
  store   :=wasmtime_store_new(engine, nil,nil);
  context :=wasmtime_store_context(store);
  linker  := wasmtime_linker_new(engine);
  error   :=wasmtime_linker_define_wasi(linker);
  if (error<>Nil) then
  exit_with_error('failed to define link wasi', error, Nil);
```

Here we create a webassembly linker, and use it to link the **WASI** functionality to our **webassembly** module: the `wasmtime_linker_define_wasi` function makes the WASI standard functions available in the webassembly module.

However, the **WASI** functions needs to be configured: what file system directories are available, what are the environment variables, command-line parameters ?
What to do with standard input, output and error output file descriptors ?

All this can be specified by creating aWASI configuration, using the `wasi_config_new` function:

```
wasi_config:=wasi_config_new();
if (wasi_config=nil) then
  exit_with_error('failed to create wasi config', Nil, nil);
```

The wasi configuration must be configured with one or more configuration functions:

`wasi_config_set_argv`
Sets values for the command-line parameters of the wasm module.

`wasi_config_inherit_argv`
Uses the values of the host program for the commandline parameters of the wasm module.

`wasi_config_set_env`
Sets the values for the environment variables of the wasm module.

`wasi_config_inherit_env`
Uses the values of the host program environment variables for the wasm module.

`wasi_config_set_stdin_file`
Specifies a file to be used as standard input for the webassembly program.

`wasi_config_set_stdin bytes`
Specifies a memory block to be used as standard input for the webassembly program.

`wasi_config_inherit_stdin`
Sets the standard input of the host program as standard input for the webassembly program.

`wasi_config_set_stdout_file`
Specifies a file to be used as standard output for the webassembly program.

`wasi_config_inherit_stdout`
Sets the standard output of the host program as standard output for the webassembly program.

`wasi_config_set_stderr_file`
Specifies a file to be used as standard error output for the webassembly program.

`wasi_config_inherit_stderr`
Sets the standard error output of the host program as standard error output for the webassembly program.

`wasi config preopen dir`
Configures a "preopened directory" as base directory for WASI file APIs.

For our simple demonstration, we'll just inherit everything from the host environment, and make the current directory available:

```
wasi_config_inherit_argv(wasi_config);
wasi_config_inherit_env(wasi_config);
wasi_config_inherit_stdin(wasi_config);
wasi_config_inherit_stdout(wasi_config);
wasi_config_inherit_stderr(wasi_config);
wasi_config_preopen_dir(wasi_config,PAnsiChar('.'),PAnsiChar('.'));
error:=wasmtime_context_set_wasi(context, wasi_config);
if (error<>Nil) then
  exit_with_error('failed to instantiate WASI', error, nil);
```

The wasmtime `context_set_wasi` function couples the **WASI** configuration to the **WASI** environment of the **webassembly** module.

We can now load the module and execute it.

Loading the **webassembly** module differs somewhat from our previous program:
instead of loading a **webassembly** text format and compiling it, we're loading an already compiled `.wasm` module:

```
F:=TMemoryStream.Create;
try
  F.LoadFromFile('hello.wasm');
  wasm_byte_vec_new_uninitialized(@wasm, F.Size);
  Move(F.Memory^,wasm.data^,F.Size);
finally
  F.Free;
end;
// Now that we've got our binary webassembly we can create our module.
Writeln('Creating module...');
error:=wasmtime_module_new(engine, Puint8_t(wasm.data), wasm.size, @module);
wasm_byte_vec_delete(@wasm);
if (error <> nil) then
  exit_with_error('failed to compile module', error, nil);
```

This time we use the **webassembly** linker to instantiate the module, as the linker needs to provide the **WASI** functionality to the webassembly module. The function to do so is called `wasmtime_linker` module:

```
function wasmtime_linker_module(
        linker: Pwasmtime_linker_t;
        store:Pwasmtime_context_t;
        name:PAnsiChar;
        name_len:Tsize_t;
        module:Pwasmtime_module_t): Pwasmtime_error_t;
```

The name of the module can be specified in the `name` and `name_len` variables.
We don't use them here: they are only needed when various modules must be linked together, because the linker will link `imports` from one module to `exports` of another module using the module name.

Since we're loading only one module, it is not necessary to specify a name:

```
 // Instantiate the module
error:=wasmtime_linker_module(linker, context, Nil, 0, module);
if (error<>nil) then
  exit_with_error('failed to instantiate module', Nil, Nil);
```

A module can have a default exported function: This is the '`_start`' symbol which starts the Free Pascal program. We extract the value of this function using `wasmtime_linker_get_default`, and call it to start the **FPC** generated program:

```
error:=wasmtime_linker_get_default(linker, context, nil, 0, @func);
if (error<>nil) then
  exit_with_error('failed to locate default export for module', error, nil);
// And call it!
Writeln('Calling export...');
error:=wasmtime_func_call(context, @func, nil, 0, nil, 0, @trap);
if wasmtime_error_exit_status(error,@status)<>0 then
  Writeln('Wasm program exited with status: ',Status)
else
  exit_with_error('Error while running default export for module', error, trap);
```

The exit proc routine in the **WASI** specification exits the **Webassembly** program.
The **Free Pascal** runtime for **webassembly** calls this when the program is halted. In wasmtime, the `exit_proc` routine raises an error to halt the program, thus, oddly enough, the result of running the `start` function is an error condition!
Luckily, the `wasmtime_error_exit_status` function can be used to check for this special case.

When the program has exited, all that is left to do is to clean up, just as in the previous example program:

```
// Clean up after ourselves at this point
Writeln('All finished!\n');
wasmtime_module_delete(module);
wasmtime_store_delete(store);
wasm_engine_delete(engine);
end.
```

The result of this can be seen in figure 3 on page 12.

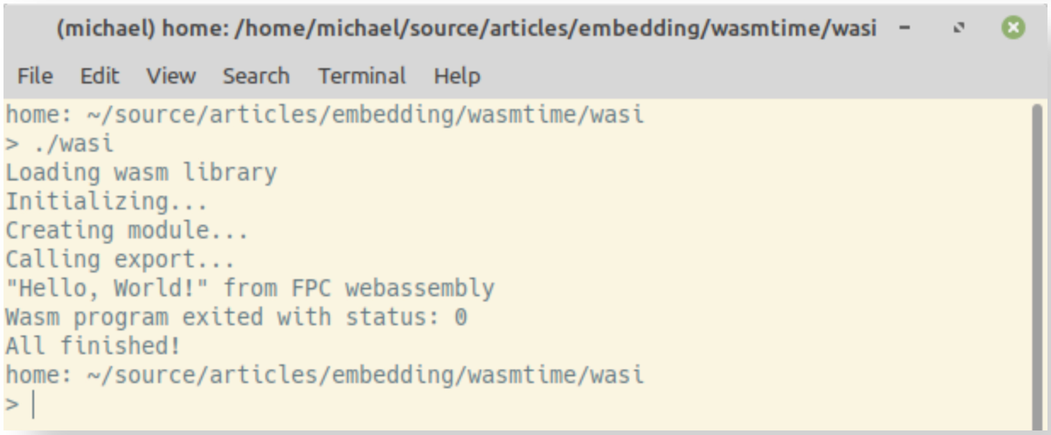
❺ USING WASMEDGE

A second library that can be used to embed **WebAssembly** programs is `wasmedge`.
Installation instructions can be found on

`https://wasmedge.org/docs/start/install/`

The unit that imports this library is called libwasmedge.
The library works largely similar to the wasmtime library, but differs in the details.
In some ways it is simpler than the wasmtime library.

It only exposes 300 functions - still a considerable number, but less than the **wasmtime** library.
The sample program we will demonstrate loads and runs the following `webassembly` function which calculates the **fibonacci** series:


(michael) home: /home/michael/source/articles/embedding/wasmtime/wasi
File
Edit
View
Search
Terminal
Help

```
home: ~/source/articles/embedding/wasmtime/wasi
> ./wasi
Loading wasm library
Initializing...
Creating module...
Calling export...
"Hello, World!" from FPC webassembly
Wasm program exited with status: 0
All finished!
home: ~/source/articles/embedding/wasmtime/wasi
> |
```

Figure 3: The Free pascal WASI-based RTL in action

```
(module
  (func $fib (export "fib") (param $n i32) (result i32)
    local.get $n
    i32.const 2
    i32.lt_s
    if
      i32.const 1
      return
    end
    local.get $n
    i32.const 2
    i32.sub
    call $fib
    local.get $n
    i32.const 1
    i32.sub
    call $fib
    i32.addzzz
    return
  )
)
```

Without going into the details of the **webassembly** format, you can see in the second line that it defines a function 'fib' which accepts a 32 bit integer as a parameter and which returns another 32-bit `integer`.
The program to call this function is relatively simple:

```
uses ctypes, libwasmedge;

var
  ConfCxt   : PWasmEdge_ConfigureContext;
  VMCxt     : PWasmEdge_VMContext;
  Returns, Params : Array[0..0] of TWasmEdge_Value;
  FuncName  : TWasmEdge_String;
  Res       : TWasmEdge_Result;
  pmodule   : pcchar;
begin
  Writeln('Loading library...');
  Loadlibwasmedge('./'+libwasmname);
  ConfCxt:=WasmEdge_ConfigureCreate();
  Writeln('Adding WASI environment...');
  WasmEdge_ConfigureAddHostRegistration(ConfCxt, WasmEdge_
  HostRegistration_Wasi);
  Writeln('Creating engine...');
  VMCxt:=WasmEdge_VMCreate(ConfCxt,Nil);
```

After loading the library, a configuration context is created using `WasmEdge_ConfigureCreate`.
The **WASI** environment is added to the configuration using the `WasmEdge_configureAddHostRegistration` routine.

Within this context, a 'virtual machine' is created that will execute the webassembly module.
The `WasmEdge_VMCreate` function is used to create this virtual machine:

```
function WasmEdge_VMCreate(
         ConfCxt: PWasmEdge_ConfigureContext;
         StoreCxt: PWasmEdge_StoreContext):PWasmEdge_VMContext;
```

The parameters are the configuration and a store (*similar to what is used in* **wasmtime**).
The store is not needed for this example.
To call the fibonacci function, a parameter is needed. This parameter is generated by the `WasmEdge_ValueGenI32` function.
Webassembly knows only a few basic types (integer, float) so the number of functions that you must use to create values is limited: there are only 8 functions of which you will use 4 in practice.

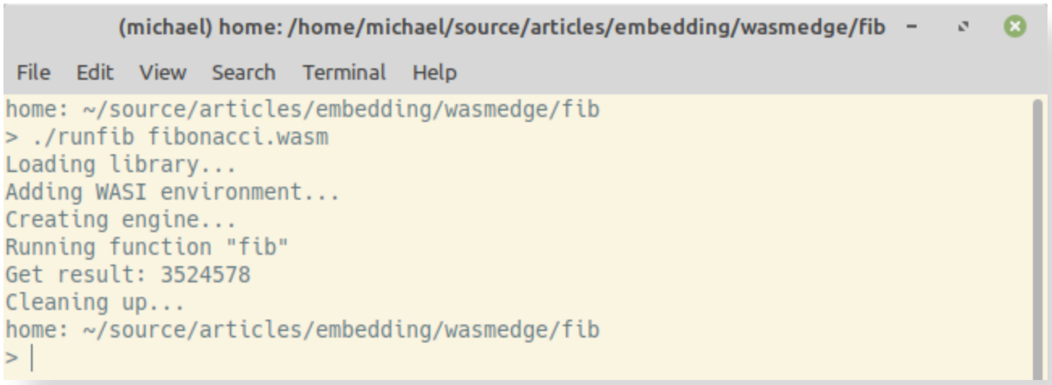
```
Params[0] := WasmEdge_ValueGenI32(32);
FuncName:=WasmEdge_StringCreateByCString(Pcchar(Pansichar('fib')));
```

The second line creates a string 'fib' that can be used by the wasmedge library. This string is used to call the **fibonacci**. Calling a function in a webassembly module can be done in a single call with the `WasmEdge_VMRunWasmFromFile` convenience function:

```
function WasmEdge_VMRunWasmFromFile(
       Cxt       :PWasmEdge_VMContext;
       Path      :pcchar;
       FuncName  :TWasmEdge_String;
       Params    :PWasmEdge_Value;
       ParamLen  :Tuint32_t;
       Returns   :PWasmEdge_Value;
       ReturnLen:Tuint32_t):TWasmEdge_Result;cdecl;
```

The arguments to this function are pretty straightforward: `path` is the filename of the module to load. `FuncName` is the function to load, `Params` and `ParamLen` specify the parameters that must be passed to the function and `Returns` and `ReturnLen` indicate the location where the return values of the function must be stored.
In the case of the `fibonacci` function, the function is used as follows:

```
pmodule:=pcchar(PAnsiChar(ParamStr(1)));
Writeln('Running function "fib"')
Res := WasmEdge_VMRunWasmFromFile(VMCxt, pmodule, FuncName,
                                  @Params, 1,
                                  @Returns, 1);
if (WasmEdge_ResultOK(Res)) then
  Writeln('Get result: ', WasmEdge_ValueGetI32(Returns[0]))
else
  Writeln('Error message: ', PAnsiChar(WasmEdge_ResultGetMessage(Res)));
```


(michael) home: /home/michael/source/articles/embedding/wasmedge/fib
File Edit View Search Terminal Help

```
home: ~/source/articles/embedding/wasmedge/fib
> ./runfib fibonacci.wasm
Loading library...
Adding WASI environment...
Creating engine...
Running function "fib"
Get result: 3524578
Cleaning up...
home: ~/source/articles/embedding/wasmedge/fib
> |
```

Figure 4: The 32th fibonacci number calculated by a webassembly program.

After checking the result of the 'run' function with `WasmEdge_ResultOK`, the return value is retrieved from the first element in the `Returns` array. The `WasmEdge_ValueGetI32` is one of the eight functions which can be used to convert a **webassembly** return value to a **Pascal** native value, in this case a 32-bit integer.
All that is left to do is to clean up the various resources that were allocated:

```
  Writeln('Cleaning up...');
  WasmEdge_VMDelete(VMCxt);
  WasmEdge_ConfigureDelete(ConfCxt);
  WasmEdge_StringDelete(FuncName);
end.
```

The result of this program can be seen in figure 4 on page 14 of this article.

❻ EMBEDDING A FPC GENERATED PROGRAM WITH WASMEDGE.

To embed a **FPC** generated webassembly module is not so different from the previous program. The start of the program is similar, the differences are in the way the **webassembly** virtual machine is created.

```
uses ctypes, libwasmedge;
var
   ConfCxt    : PWasmEdge_ConfigureContext;
   VMCxt      : PWasmEdge_VMContext;
   Returns, Params : Array[0..0] of TWasmEdge_Value;
   FuncName   : TWasmEdge_String;
   Res        : TWasmEdge_Result;
   pmodule    : pcchar;
   WasiModule : PWasmEdge_ModuleInstanceContext;
   ModName    : TWasmEdge_String;
begin
   Writeln('Loading library...');
   Loadlibwasmedge('./'+libwasmname);
   Writeln('Adding WASI environment...');
   ConfCxt:=WasmEdge_ConfigureCreate();
   WasmEdge_ConfigureAddHostRegistration(ConfCxt, WasmEdge_HostRegistration_Wasi);
   Writeln('Creating engine...');
   VMCxt:=WasmEdge_VMCreate(ConfCxt,Nil);
```

Until here, there is no difference.
Like in the case of the wasmtime library, the next step is retrieving an instance of the **WASI** module and configuring it.
This is done using the `WasmEdge_VMGetImportModuleContext` and `WasmEdge_ModuleInstanceInitWASI` functions.
The first of these two functions returns the module context of the `predefined_WASI_`module: this predefined module was enabled using the `WasmEdge_ConfigureAddHostRegistration` function, and must be configured with the

```
procedure WasmEdge_ModuleInstanceInitWASI(
   Cxt: PWasmEdge_ModuleInstanceContext;
   Args: Ppcchar; ArgLen:Tuint32_t;
   Envs: Ppcchar; EnvLen:Tuint32_t;
   Preopens:Ppcchar; PreopenLen:Tuint32_t);
```

As you can see, the list of command-line parameters, environment variables and allowed directories for file access can be configured. Standard **input/output/error** cannot be configured as in **wasmtime**. For our case, neither command-line parameters or environment variables are needed, so the configuration is quite simple:

```
WasiModule:=WasmEdge_VMGetImportModuleContext(VMCxt,WasmEdge_
HostRegistration_Wasi);
WasmEdge_ModuleInstanceInitWASI(WasiModule,Nil,0,Nil,0,Nil,0);
```

With this we can load our **webassembly** module and execute the function.
We cannot do this with the `WasmEdge_VMRunWasmFromFile` function, instead we need to load the module with the `WasmEdge_VMRegisterModuleFromFile` function:

```
function WasmEdge_VMRegisterModuleFromFile(Cxt: PWasmEdge_VMContext;
ModuleName: TWasmEdge_String; Path: pcchar): TWasmEdge_Result;
```

The module name must be specified and must be unique.
When multiple modules are loaded, the engine will link together the modules by name.
If module 'a' needs to import function 'b.proc1' then the name 'b' must be provided when loading the **webassembly** module which contains procedure 'proc1': modules do not have a name associated with them, and the filename is not related to the module name.

Since the module name is passed on as a `TWasmEdge_String type`, we must allocate it with `WasmEdge_StringCreateByCString` before loading the module:

```
ModName:=WasmEdge_StringCreateByCString(Pcchar('prog'));
pmodule:=pcchar(PAnsiChar('hello.wasm'));
Res:=WasmEdge_VMRegisterModuleFromFile(VMCxt, modname, pmodule);
if (WasmEdge_ResultOK(Res)) then
  Writeln('Loaded OK')
else
  Writeln('Error message: ', PAnsiChar(WasmEdge_ResultGetMessage(Res)));
```

Now that the module is loaded, we can actually run the start function:

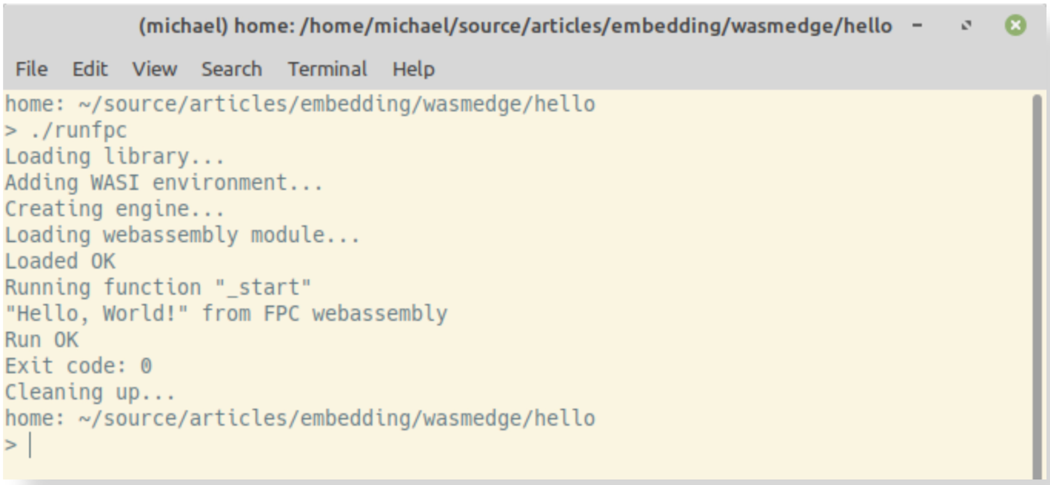
```
Writeln('Running function "_start"');
FuncName:=WasmEdge_StringCreateByCString(Pcchar('_start'));
Res:=WasmEdge_VMExecuteRegistered(VMCxt, ModName, FuncName,@Params, 0,@Returns,0);
if (WasmEdge_ResultOK(Res)) then
  begin
    Writeln('Run OK')
    Writeln('Exit code: ',WasmEdge_ModuleInstanceWASIGetExitCode(WasiModule));
  end
else
  Writeln('Error message: ', PAnsiChar(WasmEdge_ResultGetMessage(Res)));
```

NOTE that we retrieved the exit code of the FPC program with the `WasmEdge_ModuleInstanceWASIGetExitCode` function:
in difference with wasmtime, the wasmedge library does not use a trap to set the exit code.

And with that, all that is left to do is clean up, similar to the previous sample program:

```
  Writeln('Cleaning up...');
  WasmEdge_VMDelete(VMCxt);
  WasmEdge_ConfigureDelete(ConfCxt);
  WasmEdge_StringDelete(FuncName);
end.
```

With this, the program can be tested, and the output should look like in figure 5 on page 17


(michael) home: /home/michael/source/articles/embedding/wasmedge/hello
File Edit View Search Terminal Help

```
home: ~/source/articles/embedding/wasmedge/hello
> ./runfpc
Loading library...
Adding WASI environment...
Creating engine...
Loading webassembly module...
Loaded OK
Running function "_start"
"Hello, World!" from FPC webassembly
Run OK
Exit code: 0
Cleaning up...
home: ~/source/articles/embedding/wasmedge/hello
> |
```

Figure 5: Using wasmedge to run a FPC-generated webassembly module in a native FPC program.

❼ CONCLUSION

In previous articles we've shown that Free Pascal can be used to generate webassembly modules. And as shown in this article, using some external libraries, native FPC programs can load webassembly modules, whether they are generated by FPC or by some other tool.

THOUGHTS ON USABILITY OF GUI APPLICATIONS CREATED WITH LAZARUS IDE BY HELMUT ELSNER

ABSTRACT
Usability is a property of self-written programs that is not considered enough, especially by hobby programmers, but unfortunately also often by professionals. The programmer knows what the program does, which possibilities and functionality it has, but does also the potential user of the program know that? What is self-evident for the programmer, may create unsolvable problems for the user or, in the best case only cause incomprehension. But this can be easily remedied with the help of the Lazarus IDE. Likewise for Delphi

This article does not deal with scaling and DPI adjustment. That would be a separate topic.

WHAT MAKES USABILITY HAPPEN?

Without claiming to be exhaustive, here are my thoughts on it:

❶ **Clarity**
❷ **Feedback**
❸ **Structure**
❹ **Consistency**
❺ **User support**
❻ **Help**

❶ CLARITY

After starting the application, the user should immediately see what the main functions of the application are and how to reach them. Main functions or very frequently used functions should be placed prominently on the user interface and be recognizable as such. For this it's necessary to consider the prioritization of used functionality of the application before creating the GUI. For example, main functions can be made more visible with `TSpeedButton`. But, if you make a **SpeedButton** for all functions, this effect is lost.

If possible, the workflow should be visible from left to right or from top to bottom on the user interface. This could be achieved by arranging and grouping the controls accordingly (→ **Structure**). The component properties `TabStop` (`true/false`) and `TabOrder` are often neglected. The `TabOrder` order should follow the workflow. This requires only a little diligence.

Do not display too many controls on the interface. In normal cases set temporary non-usable controls to `Enabled:=false`. Use `Visible:=false` only if it is really necessary. It is usually better for the user to see that there is something, which cannot be used at the moment, than to see nothing at all.

❷ FEEDBACK

The developer should always give the user the feeling that the program will respond to him immediately. If anything takes longer, the user must be informed about it and something must be displayed to show movement in the matter. For example TProgressBar in CommonControls is available for this purpose. In any case the cursor should be adapted to the context. For longer lasting processes at least the cursor should show that there is something going on.

```
procedure ThatTakesTime;
begin
   ...
// Set wait-cursor for the application window
   Screen.Cursor.=crHourGlass;
   try
     ...
     DoSomething_ThatTakesTime;
     ...
   finally
     ...
// Reset cursor to default also if there was a exception
   Screen.Cursor:=crDefault;
   end;
end;
```

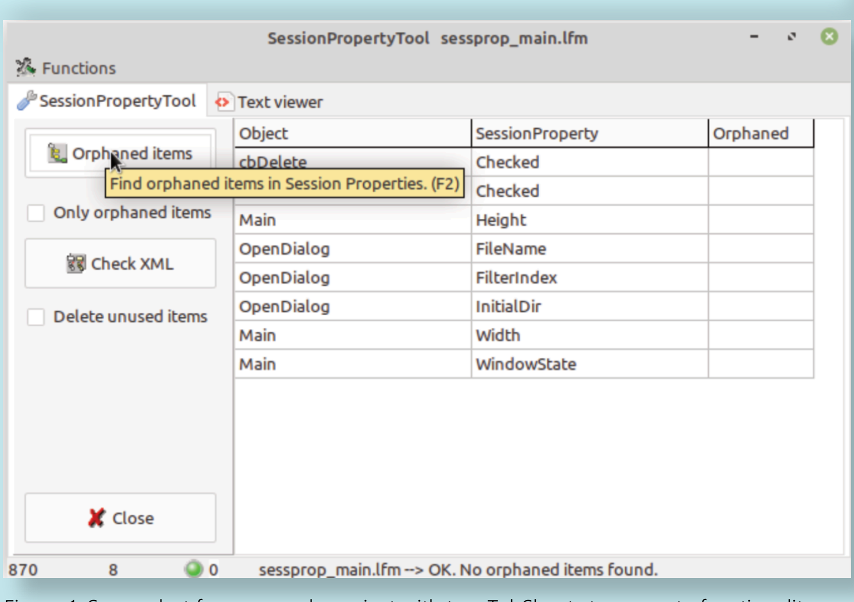
For short text output and status messages there is `TStatusBar`. Those should always tell the user what is happening in an understandable way.
For more complex programs you can also collect hints, status and error messages in a text field (*for example* `TMemo`) to show or file it.

❸ STRUCTURE
As mentioned above, the workflow should be visible on the application interface (*e.g.* **Load file → Edit file → Save file:** *from left to right*).
Related controls should be visibly separated. For this purpose there is `TGroupBox` at the standard components section.

If you have many controls, you should group them by functionality.
For this you can use multiple windows, but even there you should not overwhelm the user.

I appreciate it to work with `TPageControl` and its subordinated `TTabSheet`.
This allows a clear and structured user interface in spite of many control options.
For example, you can put all settings on one or more `TabSheets`.
So the settings are easy to reach, but do not overload the normal user interface.
In addition, `TabSheets` can be used to arrange general and function-specific control elements separately.

Avoid too colorful and by (background) images confusing user interfaces.



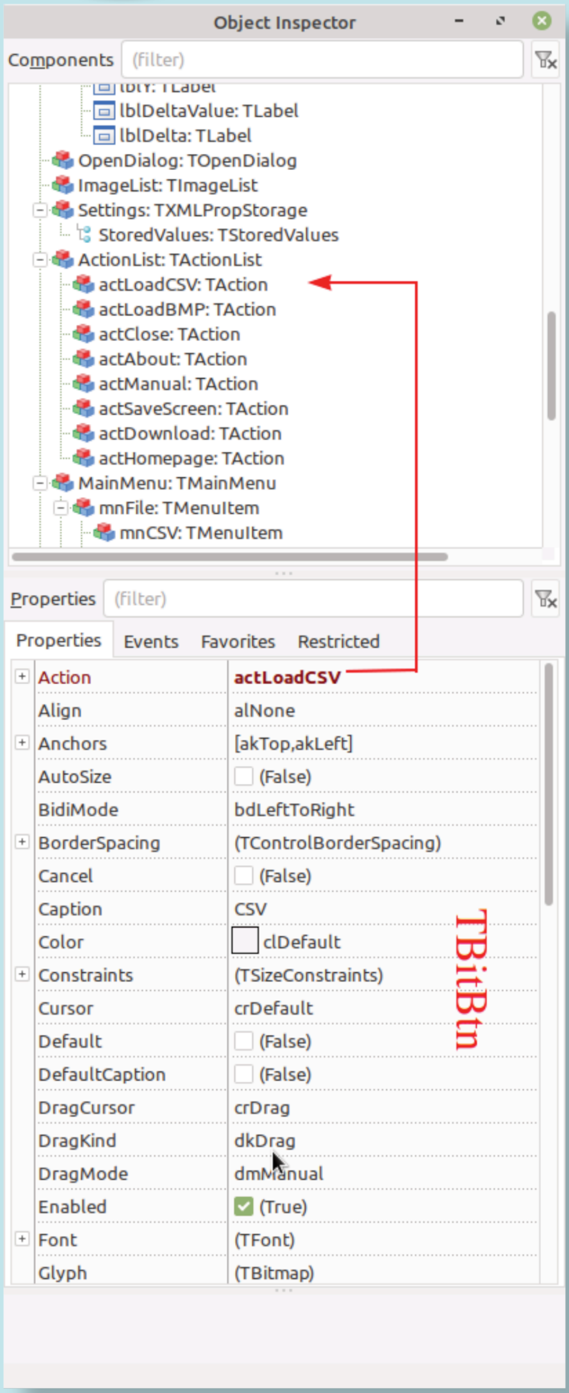
Object	SessionProperty	Orphaned
cbDelete	Checked	
	Checked	
Main	Height	
OpenDialog	FileName	
OpenDialog	FilterIndex	
OpenDialog	InitialDir	
Main	Width	
Main	WindowState	

```
870        8        0    sessprop_main.lfm --> OK. No orphaned items found.
```

Figure 1: Screenshot from example project with two TabSheets to separate functionality.

❹ CONSISTENCY

The user interface should be consistent, i.e. the same functions should have the same names or name stems, the same images (*glyph's*) and be treated in the same way.



Object Inspector

Components (filter)

- lblY: TLabel
- lblDeltaValue: TLabel
- lblDelta: TLabel
- OpenDialog: TOpenDialog
- ImageList: TImageList
- Settings: TXMLPropStorage
  - StoredValues: TStoredValues
- ActionList: TActionList
  - actLoadCSV: TAction
  - actLoadBMP: TAction
  - actClose: TAction
  - actAbout: TAction
  - actManual: TAction
  - actSaveScreen: TAction
  - actDownload: TAction
  - actHomepage: TAction
- MainMenu: TMainMenu
  - mnFile: TMenuItem
    - mnCSV: TMenuItem

Properties (filter)

Properties | Events | Favorites | Restricted

Action	actLoadCSV
Align	alNone
Anchors	[akTop,akLeft]
AutoSize	(False)
BidiMode	bdLeftToRight
BorderSpacing	(TControlBorderSpacing)
Cancel	(False)
Caption	CSV
Color	clDefault
Constraints	(TSizeConstraints)
Cursor	crDefault
Default	(False)
DefaultCaption	(False)
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual
Enabled	✓ (True)
Font	(TFont)
Glyph	(TBitmap)

Figure 2: Assign OnClick of the control to the action

An essential tool to achieve consistency and reduce development and maintenance effort is the `TActionList`.
The functions are encapsulated in `TActions` and then assigned to the control elements, which then take over the names (`Caption`), hints and glyphs.
If changes are made in the actions the control elements (***e.g. menu items and buttons***) automatically take over the GUI-relevant (**Graphical User Interface**) properties from the actions assigned to them.

`TImageList` also contributes to consistency and maintainability. All used glyphs will be collected in an **ImageList** and assigned to the actions itself or controls without associated actions. If you change an image (*glyph*) because it fits better to a consistent user interface, it will be changed automatically everywhere it was used.
ActionLists and **ImageLists** from Lazarus are a great help to get and maintain a consistent look & feel.

❺ USER SUPPORT

The user's work should be made easy wherever it is possible. In most cases it makes sense to be able to enlarge or reduce the application's window size.
Only very rarely is a fixed size useful.

Areas that contain a lot of information (*e.g. log output or require longer input, e.g. path names*) should be sizable too, usually with the application window size.

It is particularly unfriendly if text output fields cannot be enlarged, so that the user is forced to scroll unnecessarily.
For this purpose one can adjust the component property Anchors (`akTop, akBottom, akLeft, akRight`) accordingly.

Sizable windows are standard in Lazarus. This should not be changed without a good reason.
However, you can and should prevent windows from being pulled too small so that controls overlap. For this purpose there is the **`property Constraints`** in `TForm`.

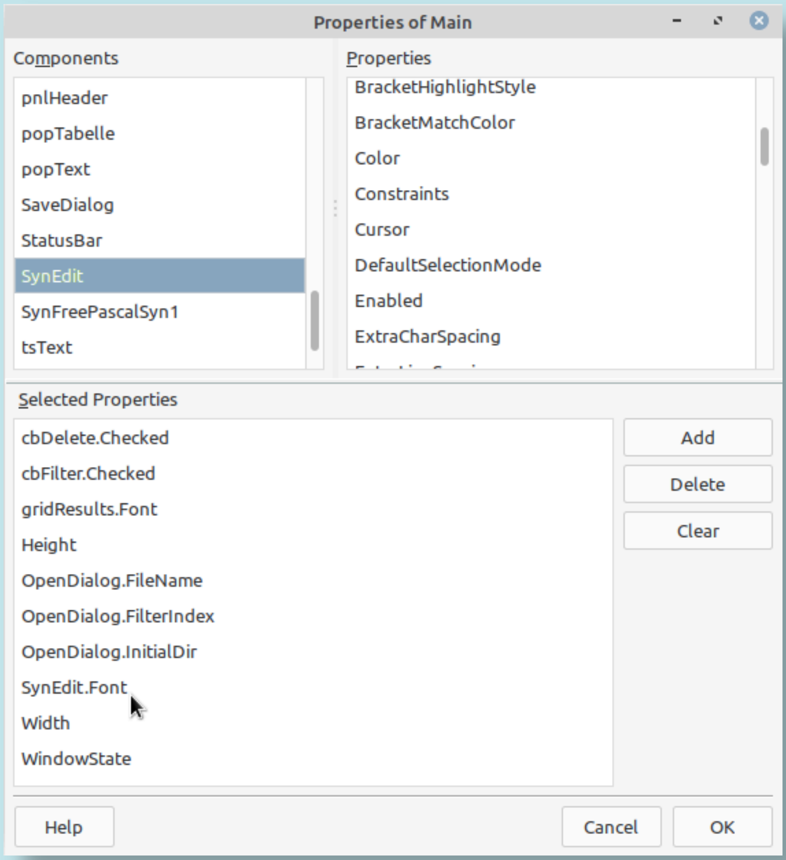
With `MinHeight` and `MinWidth` you can set the minimum size of the application window to a user-friendly size.

Important settings of the application and behavior of the application window should be saved for the next call. For this purpose Lazarus provides three formats in component section **Misc**:

- **INI file** `TIniPropStorage`
- **JSON** `TJSONPropStorage`
- **XML** `TXMLPropStorage`

One of these components will be placed on the form.
Which format one takes is left to the personal taste of the developer.
In Windows the file to store the values in the respective format will be saved in the application directory, for **UNIX-type OS** as a hidden file in the home directory.

The **`property SessionProperties`** of `TForm` makes it easy to manage the settings which should be saved.



Components: pnlHeader, popTabelle, popText, SaveDialog, StatusBar, SynEdit, SynFreePascalSyn1, tsText

Properties: BracketHighlightStyle, BracketMatchColor, Color, Constraints, Cursor, DefaultSelectionMode, Enabled, ExtraCharSpacing

Selected Properties:
cbDelete.Checked
cbFilter.Checked
gridResults.Font
Height
OpenDialog.FileName
OpenDialog.FilterIndex
OpenDialog.InitialDir
SynEdit.Font
Width
WindowState

Figure 3: Dialog to maintain SessionProperties in Lazarus-IDE

For text output, it should be possible to change the size of the letters to achieve better readability. Usually this is done with Shift + mouse scroll wheel. To do this, use the OnMouseWheelDown and OnMouseWheelUp events.

Example for `TStringGrid`:

```
procedure TForm1StringGrid1MouseWheelDown(Sender: TObject;
Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if ssCtrl in Shift then
   StringGrid1.Font.Size:=StringGrid1.Font.Size-1;
end;

procedure TForm1.StringGrid1MouseWheelUp(Sender: TObject;
Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
begin
  if ssCtrl in Shift then
   StringGrid1.Font.Size:=StringGrid1.Font.Size+1;
end;
```

❻ HELP

This is not about the infamous help systems that are supposed to replace a user manual, in the best case maybe it can, but about the help that the GUI design can provide. This includes the above mentioned preferably visible workflow, the clarity and structure.
But there is another powerful tool to help the user:
The **Hints**. Almost every control that the Lazarus component palette offers has a `Hint` property. To show the hints, you first have to set the `ShowHint` property to true in `TForm`.

Then you can assign strings with meaningful content to the controls in a resourcestring section. Hints can also be multiline.
These hints then appear everywhere where it makes sense when the user holds the mouse pointer over them.

For simple applications, this is actually all that is needed as documentation. The program is self-explanatory - the ideal case.The `property Hint` offers even more advanced possibilities when assigning hint contents depending on certain conditions.

Here is an example of how to assign a separate hint to each individual cell in a `TStringGrid`. This is useful if the text in the cell is larger than the cell and the hint displays the entire text, or if you want to display more detailed information about the content of the cell.

```
function GetCellInfo(grid: TStringGrid; col, row: integer): string;
begin
// Fill with conditions per cell to create a hint
// ...
// Simple example:
 result:='Column number '+IntToStr(col)+' - Line number '+IntToStr(row);
end;

procedure TForm1.StringGrid1MouseMove(Sender: TObject; Shift:
                   TShiftState; X, Y: Integer);
var
 colidx, rowidx: integer;

begin
// Find the cell below mouse pointer
 StringGrid1.MouseToCell(x, y, colidx, rowidx);
// Default if no condition was given
 StringGrid1.Hint:=StringGrid1.Cells[colidx, rowidx];

// Exclude header and call a function with hint conditions per cell
 if rowidx > 0 then
  StringGrid1.Hint:=GetCellInfo(StringGrid1, colidx, rowidx);
end;
```

What you should avoid are blocking message windows like `MessageDlg()` and other modal dialogs as well as forms that are called with `ShowModal`. Always use these only where it is unavoidable to stop further operation to force a decision.

There is nothing more annoying than getting the message **"Do you really want to quit?"** after pressing the `Close` button. This only makes sense if an important work hasn't been saved yet, and that's what the message should be about.

Also an **AboutBox** does not necessarily have to be blocking (`modal`).

CONCLUSION

All in all, good usability requires a lot of planning and consideration, a lot of writing and additional effort, but it pays off in the acceptance of the application.

With rarely used applications, even if I wrote them myself, I am often grateful when I see a hint and remember easily what I had intended.

Example project: **SessionPropertyTool** in `SessionPropertyTool.zip`

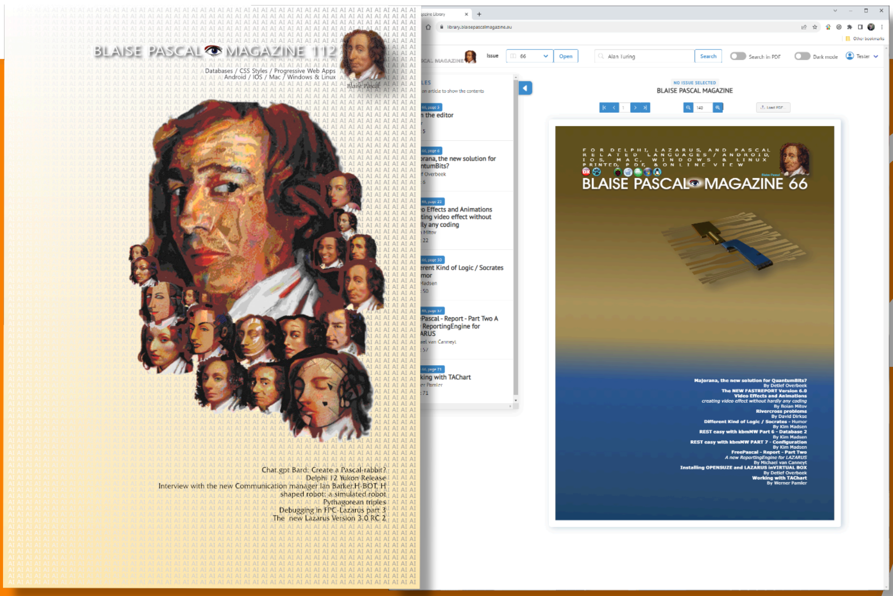
RECOMMENDATIONS

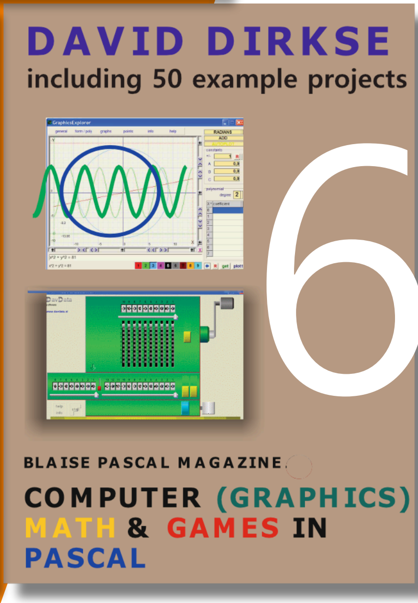
Design:
- Use `TActionList` to maintain application functionality
- Use `TImageList`
- resourcestring section for **Captions** or **Titles**
- `TStatusBar` and `TProgessBar` for messages to user
- Set `Screen.Cursor` context dependent
- TGroupBox, TPageControl with `TTabSheet` to separate the controls

TForm:
- Constraints for window size
- Use `SessionProperties`
- `ShowHint` to true and create useful `Hint` texts
- Avoid `ShowModal`

Components, controls:
- Set `Anchors` for optimal sizes of text and input fields
- Save `Assign TabStop, TabOrder`
- Use rather `Enabled` instead of `Visible`
- Create good **Hints**
- Use only really necessary `modal` dialogs
- Font sizable

ADVERTISEMENT


1
2
BLAISE PASCAL MAGAZINE 112
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
Chat gpt Bard: Create a Pascal-rabbit?
Delphi 12 Yukon Release
Interview with the new Communication manager Ian Barker.H-BOT, H shaped robot: a simulated robot
Pythagorean triples
Debugging in FPC-Lazarus part 3
The new Lazarus Version 3.0 RC 2
BLAISE PASCAL MAGAZINE 66


DAVID DIRKSE
including 50 example projects
6
BLAISE PASCAL MAGAZINE
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

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

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

SUPER PACK
6 ITEMS
2023

PRICE € 150

NORMAL PRICE € 275

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

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

BLAISE PASCAL MAGAZINE

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

Blaise Pascal

COMPONENTS DEVELOPERS 4

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

COMPONENTS DEVELOPERS 4

D11

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

kbmMW Professional and Enterprise Edition v. 5.22.10
kbmMemTable v. 7.98.00 Standard and Professional Edition

5.22.00 is a release with containing new stuff, refinements and bugfixes.,**OpenSSL v3 support,** WebSocket support, further improvements to SmartBind, new high performance hashing algorithms, improved RemoteDesktop sample and much more.
This release requires the use of **kbmMemTable** v. 7.97.00 or newer.

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

kbmMemTable is the fastest and most feature rich in memory table for Embarcadero products.
- Easily supports large datasets with millions of records
- Easy data streaming support
- Optional to use native SQL engine
- Supports nested transactions and undo
- Native and fast build in M/D, aggregation/grouping range selection features
- Advanced indexing features for extreme performance

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

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

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

COMPONENTS 4 DEVELOPERS

D11

ADVERTISEMENT