

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

Umstellung auf 64-bit max-box
Hilfe file für den Internet LIBstick
Zählen von binären Rätsellösungen
Polygonale Erweiterung
Castle Game Engine Teil 2: die schlechte Art, Schach zu spielen:
3d Physik Spass mit der Castle Game Engine (teil 2)
Der Lazarus debugger teil 4: Einblicke - Watches
Senden von Debug-protokollen an den Server in PAS2JS
Einbettung von WebAssembly in ein FreePascal-Programm
Endlich ist es da: jetzt können wir es nutzen
Gedanken zur Benutzerfreundlichkeit



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

- Von der Redaktion Seite 4
- Umstellung auf 64-bit max-box Seite 6
- Hilfe file für den Internet LIBstick Seite 23
- Zählen von binären Rätsellösungen Seite 32
- Polygonale Erweiterung Seite 40
- Castle Game Engine Teil 2: die schlechte Art, Schach zu spielen: Seite 48
- 3d Physik Spass mit der Castle Game Engine (teil 2)*
- Der Lazarus debugger teil 4: Einblicke - Watches Seite 77
- Senden von debug-protokollen an den server in PAS2JS Seite 91
- Einbettung von WebAssembly in ein FreePascal-Programm** Seite 99
- Endlich ist es da: jetzt können wir es nutzen*
- Gedanken zur Benutzerfreundlichkeit Seite 116

ANNONCEN

- | | |
|---|-----------|
| Components for Developers | Seite 124 |
| David Dirkse computer math/games in Pascal | Seite 46 |
| Database Workbench | Seite 90 |
| Deutsches Lazarus Handbuch Pocket | Seite 98 |
| Help for Ukraine | Seite 123 |
| Lazarus Handbook Pocket | Seite 76 |
| Lazarus Handbook Pocket + Subscription | Seite 31 |
| Lazarus Handbook PDF + Subscription | Seite 75 |
| LIBRARY Internet Library | Seite 38 |
| LIBRARY Lib Stick | Seite 39 |
| Nexus DB 20 years | Seite 47 |
| New subscription model | Seite 22 |
| PDF Viewer 2023 Blaise Pascal Library USB stick | Seite 89 |
| Subscription 2 year | Seite 97 |
| Superpack 6 Items | Seite 122 |

Photo von Alexx Cooper on Unsplash
<https://unsplash.com/@hialexxlarioss>



Niklaus Wirth

Pascal ist eine imperative und prozedurale Programmiersprache, die Niklaus Wirth (links unten) in den Jahren 1968-69 entworfen und 1970 veröffentlicht hat. Es handelt sich dabei um eine kleine, effiziente Sprache, die gute Programmierpraktiken unter Verwendung von strukturierter Programmierung und Datenstrukturierung fördern soll. Ein Derivat, bekannt als Object Pascal, wurde 1985 für die objektorientierte Programmierung entwickelt. Der Name der Sprache wurde zu Ehren des Mathematikers und Erfinders der ersten Rechenmaschine gewählt: Blaise Pascal (siehe oben rechts).

Herausgeber: © Foundation Supporting Programming Language Pascal - mit Sitz in den Niederlanden
Registriertes Name: Stichting Ondersteuning Programmeertaal Pascal IJsselstein, Netherlands
VAT / BTW: NL814254147B01 Chamber of commerce (KVK) 30 202429 Handy: +31 6 21 23 62 68



MITARBEITER

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 @ fixments.com
Mattias Gärtnernc- gaertnma@netcologne.de	Max Kleiner www.softwareschule.ch max @ kleiner.com	John Kuiper john_kuiper @ kpnmail.nl	Wagner R. Landgraf wagner @ tmssoftware.com
Vsevolod Leonov vsevolod.leonov@mail.ru	Andrea Magni www.andreamagni.eu andrea. magni @ gmail.com www.andreamagni.eu/wp		Helmut Elsner Korrektor der Deutschen Ausgabe helmut.elsner@live.com
		Paul Nauta PLM Solution Architect CyberNautics paul.nauta @ cybernautics.nl	
Kim Madsen www.component4developers.com kbmMW		Boian Mitov mitov @ mitov.com	
	Jeremy North jeremy.north @ gmail.com	Detlef Overbeek - Editor in Chief www.blaisepascal.eu editor @ blaisepascal.eu	
Anton Vogelaar ajv @ vogelaar-electronics.com	Danny Wind dwind @ delphicompany.nl	Jos Wegman Corrector / Analyst	Siegfried Zuhr siegfried @ zuhr.nl

Chefredakteur

Detlef D. Overbeek, Niederlande Tel.: Mobil: +31 (0)6 21.23.62.68

Nachrichten und Pressemitteilungen nur per E-Mail an editor@blaisepascal.eu

Abonnemente können online unter <https://www.blaisepascalmagazine.eu/deutsche-Ausgabe/> oder per schriftlicher Bestellung abgeschlossen werden oder indem Sie eine E-Mail an office@blaisepascal.eu senden. Das Abonnement kann zu einem beliebigen Zeitpunkt beginnen. Alle Ausgaben, die im Kalenderjahr des Abonnements veröffentlicht werden, werden geschickt. Das Abonnement hat eine Laufzeit von 365 Tagen. Abonnemente werden nicht ohne Vorankündigung verlängert

Der Zahlungseingang wird per E-Mail verschickt. Sie können das Abonnement bezahlen, indem Sie die Zahlung an folgende Adresse senden: ABN AMRO Bank Konto Nr. 44 19 60 863 oder per Kreditkarte oder Paypal Name: Stiftung Pro Pascal (Stichting Programmeertaal Pascal) IBAN: NL82 ABNA 0441960863 BIC ABNANL2A Umsatzsteuer-Nr.: 81 42 54 147 (Stichting Programmeertaal Pascal) Abonnementsabteilung Edelstenenbaan 21 / 3402 XA IJsselstein, Niederlande Mobil: + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

Markenzeichen Alle verwendeten Markenzeichen sind Eigentum der jeweiligen Inhaber. Vorbehalt Obwohl wir uns bemühen sicherzustellen, dass die in der Zeitschrift veröffentlichten Informationen korrekt sind, können wir keine Verantwortung für Fehler oder Auslassungen übernehmen. Wenn Sie etwas bemerken, das möglicherweise nicht korrekt ist, wenden Sie sich bitte an den Herausgeber, und wir werden gegebenenfalls eine Korrektur veröffentlichen.



Mitglied der **Königlich Niederländischen Bibliothek**

KB

Mitglied und Spender von **WIKIPEDIA**

Subscriptions (2022 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 :	€ 64,20	€ 70		

COPYRIGHT-HINWEIS

Das gesamte in Blaise Pascal veröffentlichte Material unterliegt dem Copyright © SOPP Stichting Ondersteuning Programmeertaal Pascal, sofern nicht anders angegeben, und darf nicht ohne schriftliche Genehmigung kopiert, verbreitet oder neu veröffentlicht werden. Die Autoren erklären sich damit einverstanden, dass der zu ihren Artikeln gehörende Code nach der Veröffentlichung den Abonnenten zur Verfügung gestellt wird, indem er auf der Website der PGG zum Download angeboten wird, und dass Artikel und Code auf verteilbaren Datenträgern gespeichert werden. Die Nutzung von Programmlisten durch Abonnenten zu Forschungs- und Studienzwecken ist erlaubt, jedoch nicht zu kommerziellen Zwecken. Die kommerzielle Nutzung von Programmlistings und Code ist ohne die schriftliche Genehmigung des Autors untersagt.



Von der Redaktion

Guten Tag,
um gleich auf den Punkt zu kommen: Ich habe das Layout in ein viel praktikableres und lesbareres Layout geändert: flacher - weniger farbig - größerer Abstand zwischen den Zeilen und eine schärfere Schrift und die Kodierungswerte wurden geändert.
Ich hoffe, dass Sie alle mit diesem neuen Layout zufrieden sind.
Dies war das Ergebnis einer Reihe von Diskussionen, die ich mit Lesern geführt habe.
Einige liebten die Farben, aber ich dachte, wenn man mit dem Text arbeiten will, sollte er so einfach wie möglich zu lesen sein - mit wenig Kontrast, übersichtlicher.
Natürlich ist es jetzt ein viel flacheres Erscheinungsbild.
Aber das ist das eine oder das andere.
So, jetzt bitte ich Sie, mir zu sagen, ob Ihnen das neue Layout gefällt oder was Ihre Meinung dazu ist. Lassen Sie es mich wissen unter:
editor@blaisepascal.eu
Ich hoffe, Sie werden mir zustimmen, dass wir immer besser werden wollen und deshalb Dinge ändern müssen.

Wir waren gezwungen, den **Preis für die gedruckte Ausgabe zu erhöhen**, weil die Versandkosten erheblich gestiegen sind. Das Schlimme daran ist, dass jedes Heft, das ich verschicke, 10 € kostet. Die Kosten für jeden Versand würden immer noch mehr als 50 % der Gesamtkosten ausmachen, selbst wenn ich die Druckkosten senken könnte. Es wird also ein extrem teures Unterfangen.
Ich habe keine Ahnung, wie ich dieses Problem lösen kann.

Um unseren Service zu verbessern, haben wir die **Internetversion** der "Bibliothek aller Hefte" unserer Zeitschrift - nur zum Lesen - aufgenommen.
Wenn Sie die komplette Bibliothek mit allen Downloads und Codes haben wollen, müssen Sie den "LIB-Stick auf USB" kaufen - er sieht aus wie eine Kreditkarte ist aber ein USB stick.

In diesem Moment bauen wir die neue Version unserer Website auf und wir möchten Sie bitten, uns mitzuteilen, welche Art von Service Sie sich wünschen würden.
Ich würde Sie gerne dazu überreden, einen Artikel zu schreiben.
Es gibt viele von Ihnen mit brillanten Ideen. Lassen Sie es die Welt wissen.
Wir werden Ihnen beim Schreiben Ihres Artikels helfen.

In diesem Heft hat "Michalis Kamburelis" über seine Spiele-Engine geschrieben und wie man ein Spiel programmiert (codiert). Das ist wunderbar.
Es wird nun möglich sein, dies auf einfache Art und Weise zu tun.
Vielleicht können Sie sogar Ihr eigenes Spiel schreiben? Haben Sie schon einmal daran gedacht, eine Benutzeroberfläche mit in 3D gerenderten beweglichen Teilen zu kombinieren?
So dass der Benutzer mit einfachen Beispielen für die Verwendung Ihrer Anwendungsschnittstelle überrascht wird?
Spielerisches Lernen und Lehren?
Werfen Sie einen Blick auf das fantastische und sehr überzeugende Beispiel der 3D-Tiere, die atemberaubend schön sind und zur Verwendung zur Verfügung stehen.



From your editor

<https://sketchfab.com/features/free-3d-models>

haben viele kostenlose Beispiele und natürlich können Sie auch Ihre eigenen erstellen oder nach speziell entworfenen Modellen suchen:

<https://youtu.be/NTaHgf7okwk> - ein laufender Gepard.

Ich habe ein kurzes Video des schlechten Schachspiels auf unsere Website gestellt. Schauen Sie einfach mal rein.

Es ist sehr überzeugend und es kann alles in Pascal gemacht werden: In Delphi und auch in Lazarus.

Es hat eine enorme Entwicklung gegeben:

Webassembly für FreePascal.

Wir sind nun in der Lage, FPC innerhalb von Webassembly laufen zu lassen und das bedeutet, dass eine ganze Menge neuer Techniken auch für **Pas2js** verfügbar werden.

Wir werden darüber im nächsten Heft 114/115 schreiben, um all die kommenden Techniken zu erklären und Beispiele zu zeigen und was wir bis jetzt schon erreicht haben.

Embarcadero hat versprochen, eine neue Version 12 herauszubringen. Sie ist jetzt gerade vorgestellt worden, und so hoffe ich, Ihnen eine Menge neuer Dinge aus Delphi zeigen zu können.

Bitte teilen Sie mir Ihre Ideen zu diesem Artikel mit, und vielen Dank fürs Lesen.

Detlef





EINFÜHRUNG

Wenn Sie eine Codebasis von 32-Bit-Windows-Delphi-Anwendungen haben, die Sie auf 64-Bit-Windows konvertieren möchten, sollten Sie zunächst eine Reorganisation der Quellen vornehmen, um einen Überblick zu erhalten.

Der Quellcode ist in `_C` für Komponenten und `_R` für Runtime (native Units) und `_D` für Design Units (Skript-Mapping-Importe) gegliedert, wie die folgende Grafik von Package Neuralvolume von CAI NeuralNetwork als 4 Dateien zeigt:

The screenshot shows a GitHub web interface. In the top left, there's a search bar with 'neuralvolume' entered. Below it, a dropdown menu lists four files:

- source2022_4_R/neuralvolume.pas
- source2022_4_R/neuralvolumev.pas
- source2022_4_D/uPSI_neuralvolume.pas
- source2022_4_D/uPSI_neuralvolumev.pas

To the right, a code editor shows the beginning of a Pascal file:

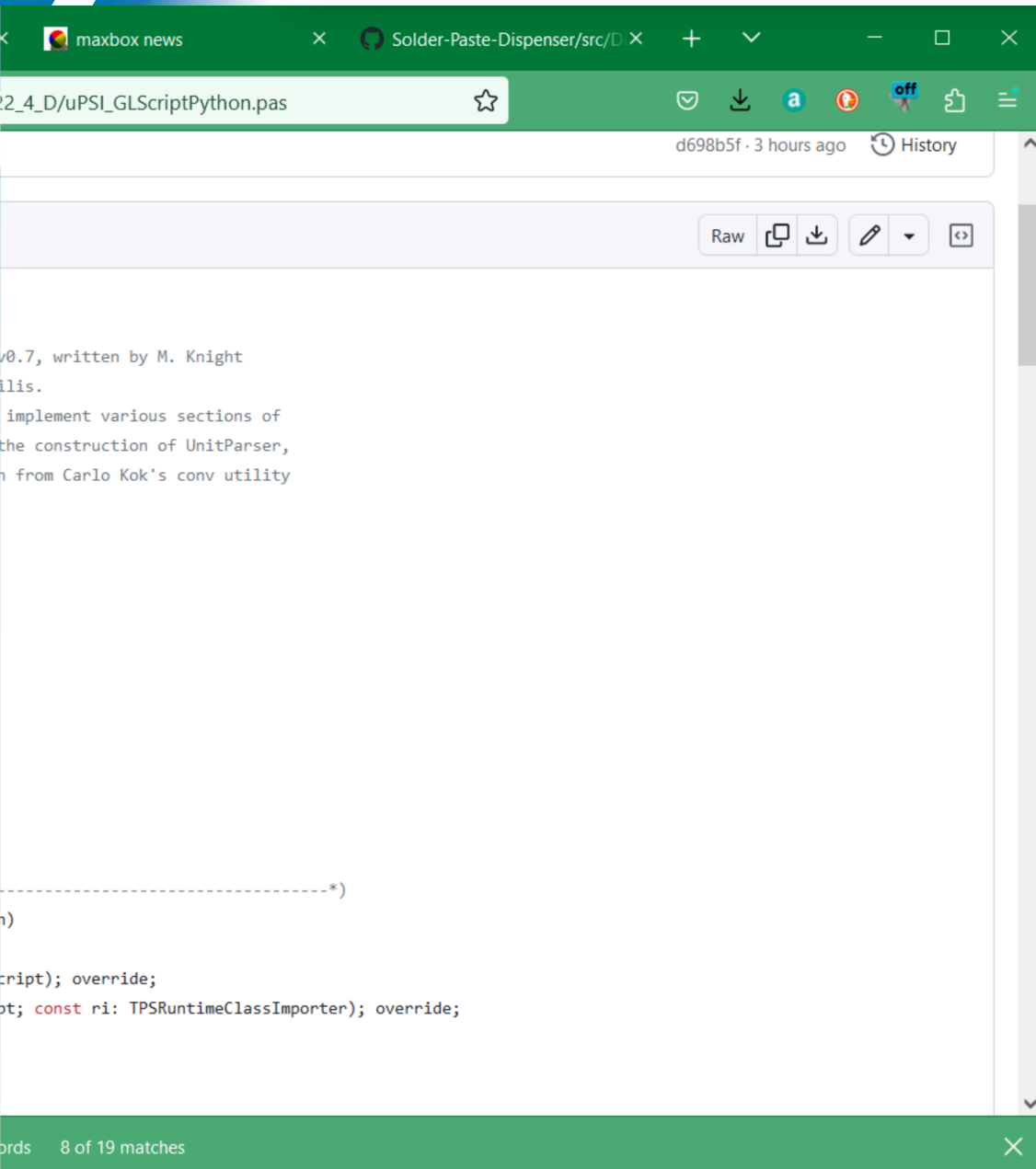
```

1  unit uPSI_GLScriptPython;
2  {
...
11
12  uses
13      SysUtils
14      ,Classes
15      ,uPSComponent
16      ,uPSRuntime
17      ,uPSCompiler
18      ;
19
20  type
21  (*-----
22  TPSImport_GLScriptPython = class(TPSPlugi
23  protected
24      procedure CompileImport1(CompExec: TPSS
25      procedure ExecImport1(CompExec: TPSScri
26  end;
27
28
    
```

maXbox Starter 113
"Lost in translation – ghost in application" – Hardcore code.
 Source: firstdemo_master11_cop21web.txt and my6_cannonball.txt blog



Es ist also nicht so einfach, Ihre 32-Bit-Anwendung in der IDE zu öffnen, die 64-Bit-Windows-Zielplattform hinzuzufügen und zu aktivieren, und Ihre Anwendung als 64-Bit-Windows-Anwendung zu kompilieren. Beim Graben oder Tauchen durch den Quellcode von maXbox4 scheint es unmöglich zu sein, über 3300 Einheiten (genau 3335) in eine anständige und ordnungsgemäße Weise zu maXbox5 alias 64.bit Version zu migrieren.



The screenshot shows a web browser window with a green header. The address bar contains the URL `2_4_D/uPSI_GLScriptPython.pas`. The page content is a code editor with a light gray background. The code is Pascal and includes comments and function definitions. A search bar at the bottom of the editor shows the text `ords` and `8 of 19 matches`.

```
0.7, written by M. Knight  
ilis.  
implement various sections of  
the construction of UnitParser,  
h from Carlo Kok's conv utility  
  
-----*)  
)  
cript); override;  
ot; const ri: TPSRuntimeClassImporter); override;
```

SOURCE ORGANISATION

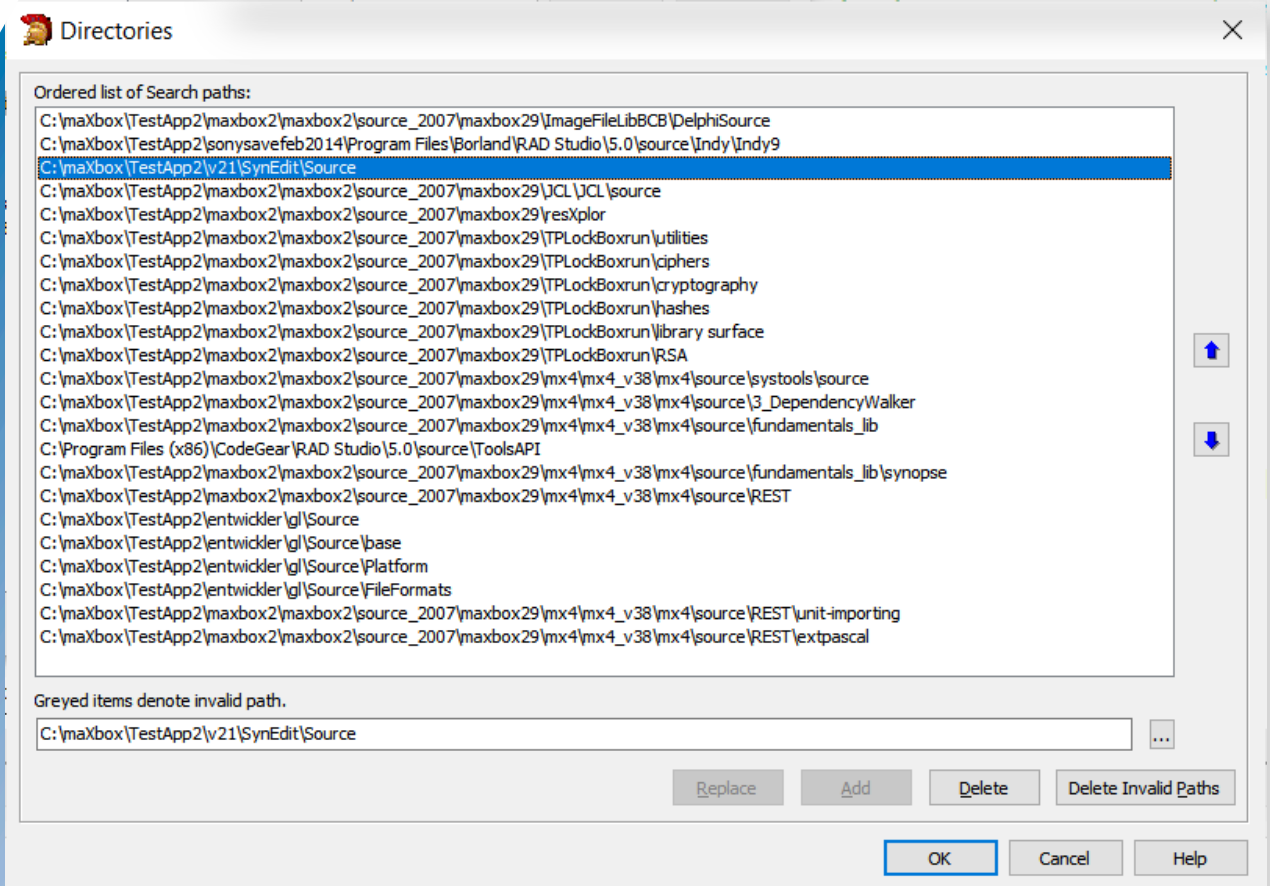
Das erste, was ich sagen muss, ist die fehlende Unterstützung jeglicher Art der ganzen "HiRes/4K oder DPI Awareness Resolution and Delphi forms" Revolution;

Es gibt keine "Mein Formular in allen Bild-Auflösungen richtig aussehen lassen" Checkbox oder einen Emulator, der alle Formulare durchspielt.

Aber Sie können auf die Einstellung "**keine Unterstützung für hohe DPI**" zurückgreifen.

Ich weiß, dass dies nicht die Verbesserung ist, die wir haben wollen, aber es verursacht die wenigsten Kopfschmerzen.

Wie beim **mX4** und dem kommenden **mX5** ist die App "out of the box" (*self containment*) und benötigt keine Installation oder Registrierung. Sie hat eine unabhängige Systemarchitektur (*ISA*). Für die Reorganisation der Quellen habe ich die letzte Revision mit den Patches aus *Heft #202 (Commit 86a057c)*, aber ich kann die Dateien, die Teil des `PascalScript_Core_D27.dpk` für diese Plattform sind, zunächst nicht kompilieren (**Core_D26**), weder für **Linux64**, **Win64** noch **MacOS64**.



In **Delphi** kann ich den Quellcode eines Ordners einbinden, indem ich ihn zum Suchpfad des Projekts hinzufüge oder indem ich ihn zum Bibliothekspfad hinzufüge.

Der Suchpfad gilt nur für das aktuelle Projekt, während der Bibliothekspfad für jedes mit der IDE geöffnete Projekt gilt.

Aber abgesehen davon, gibt es keinen funktionalen Unterschied zwischen dem Such- und dem Bibliothekspfad?

Wenn ich diesen Ordner in den Suchpfad von Projekt A einbeziehe, heißt es, dass eine bestimmte Datei in diesem Ordner nicht gefunden werden kann. Wenn ich ihn in den Bibliothekspfad einbeziehe, dann kompiliert **Testprojekt A** problemlos.

Soweit ich weiß, ist der **Browsing-Pfad** der Ort, an dem der **Debugger** nach Dateien, die sich nicht im **Bibliothekspfad** befinden, suchen sollte, wenn er Quellcode abarbeitet und Haltepunkte setzen soll. Nehmen wir an, Sie verwenden eine Komponente eines Drittanbieters. Sie geben den Bibliothekspfad auf das Verzeichnis an, in dem sich die vorkompilierten dcu-Dateien dieser Komponente befinden. Ihr Projekt verwendet beim Kompilieren diese dcu-Dateien. Klar, damit nicht jedes Mal neu kompiliert werden muss, wenn Sie einen Build durchführen. Dies ist offensichtlich, da es nicht jedes Mal neu kompiliert werden muss, wenn Sie einen **Build** durchführen.

Die **Standardeinstellungen** für die **VCL** zeigen dies. Im Bibliothekspfad haben sie \$(BSD)\Lib angegeben, und im Browserpfad haben sie \$(BDS)\SOURCE\WIN32 angegeben...

Hier ist einige Compiler-Ausgabe zunächst zum Vergleich mit **Delphi 10.3.2 Rio** und dann **10.4 dccosx64** oder **dcc64** Compiler (*ähnliche Ergebnisse gibt es für dclinux64*):

```
[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'
```

Wenn ich den beanstandeten Code auskommentiere, erhalte ich folgendes, das langsam nicht mehr so trivial aussieht...

```
[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'
```

Der Grund für alle Probleme unter **OSX64** (und **Linux64**, glaube ich auch) mit **PS** ist "": die **LongInt** und **LongWord** Datentypen sind auf 64-bit **POSIX*** Plattformen unterschiedlich. Um die Interoperabilität zwischen **Delphi** und der **POSIX*-API** zu erhalten, wird für **64-Bit POSIX*-Plattformen** die Größe der **LongInt**- und **LongWord**-Typen auf 64-Bit geändert. *Alle 32-Bit-Plattformen und 64-Bit-Windows-Plattformen behalten 32-Bit für die LongInt- und LongWord-Typen.*"

Die Behebung kann also sehr einfach sein - ändern Sie ALLE **LongInt**-Typen in **Integer**.
Dateien: **uPSCompiler**, **uPSComponent**, **uPSDebugger**, **uPSRuntime**, **uPSUtils**.

Aber nicht durch Auto-Replace von "Longint" auf "Integer" ändern, weil in diesem Fall Deklarationen wie `AddTypeCopyN('Integer', 'LongInt');` und andere - die Referenz als **Stringliteral** gebrochen wird.

Ich habe 2 Assembler-Routinen entworfen, die hoffentlich von jemandem, der ihre Absicht versteht, in reines **Pascal** übersetzt werden können. Ich bin mir nicht wirklich sicher, was der Assembler macht

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

Möglicherweise können wir wie vorgeschlagen einfach `{$DEFINE empty_methods_handler}` nehmen, um den **Assembler** zu vermeiden. Aber ich weiß nicht, was der **Assembler** zu tun versucht. Ist also ein Stub akzeptabel oder brauchen wir ihn, um etwas zu tun?

Wie man sehen kann, hat die Arbeit an der 64bit-Box begonnen, aber Libs, Maps, Objektdateien, Transpiler und die Registrierung sind voller Fallen. Wenn die Option "Use Debug DCUs" nicht aktiviert ist und ich unsere Anwendung debugge, kann ich nur einen Schritt durch meinen eigenen Code machen.

Das ist das, was wir in den meisten Fällen wollen, weil es unser Code ist, der fehlerhaft ist, und nicht der normale **Delphi-Code**. Es wäre ziemlich lästig, immer wieder in Delphi Code einzugreifen.

Die **Portable Operating System Interface (POSIX; IPA*)** ist eine Familie von Standards, die von der **IEEE Computer Society** zur Aufrechterhaltung der Kompatibilität zwischen Betriebssystemen spezifiziert wurden. **POSIX** definiert sowohl die Anwendungsprogrammierschnittstellen (**APIs**) auf System- und Benutzerebene als auch Befehlszeilenshells und Dienstprogrammchnittstellen für die Softwarekompatibilität (Portabilität) mit Varianten von **Unix** und anderen Betriebssystemen. **POSIX** ist auch eine Marke des **IEEE**. **POSIX** soll sowohl von Anwendungs- als auch von Systementwicklern genutzt werden. Das **Internationale Phonetische Alphabet (IPA)** ist ein alphabetisches System der phonetischen Notation, das hauptsächlich auf der lateinischen Schrift basiert.

REORGANISATION

Nachdem ich den Quellcode von **Delphi 2007** (siehe Bild 2 Artikelseite5) auf Github umstrukturiert und in **Delphi 10.4** konfiguriert hatte, begann ich, die folgenden Fälle zu überprüfen und zu bearbeiten (hauptsächlich im Zusammenhang mit Zeigeroperationen in **WinAPI-Heften, NativeInt-Größe und Assembly-Code**):

Wenn Sie Zeiger an `SendMessage/PostMessage/TControl.Perform` übergeben, sollten die Parameter `wParam` und `lParam` auf den Typ `WPARAM/LPARAM` gecastet werden und nicht auf `Integer/Longint`.

Correct: `SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));`

Wrong: `SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));`

Ersetze `SetWindowLong/GetWindowLog` with `SetWindowLongPtr/GetWindowLongPtr` für `GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDPARENT` und `GWLP_WNDPROC`, da sie Zeiger und Handles zurückgeben. Zeiger, die an `SetWindowLongPtr` übergeben werden, sollten in den Typ `LONG_PTR` und nicht in `Integer/Longint` umgewandelt werden.

Correct: `SetWindowLongPtr(hWnd, GWLP_WNDPROC, LONG_PTR(@MyWindowProc));`

Wrong: `SetWindowLong(hWnd, GWL_WNDPROC, Longint(@MyWindowProc));`

In der runtime library mussten mehrere UNITS bearbeitet werden:

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

Überprüfen Sie den `IFDEF WIN32` in dem Sinne: Versuchen Sie zu konvertieren oder lassen sie. In **Delphi** werden **Strings** als Ergebniswerte wie `var`-Parameter behandelt. Mit anderen Worten, eine Funktion wie `Foo` ist in der Tat kompiliert als:

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

Zeilen von `WIN32` nach `WIN64` zu konvertieren ist also mit Referenzen oder typlosen Referenzen wie `procedure Foo(var Result;)` möglich.

Overloads: Für Funktionen, die `PChar` benötigten, gibt es jetzt `PAnsiChar`- und `PWideChar`-Versionen, so dass die entsprechende Funktion aufgerufen wird. `AnsiXXX`-Funktionen sind eine Überlegung wert:

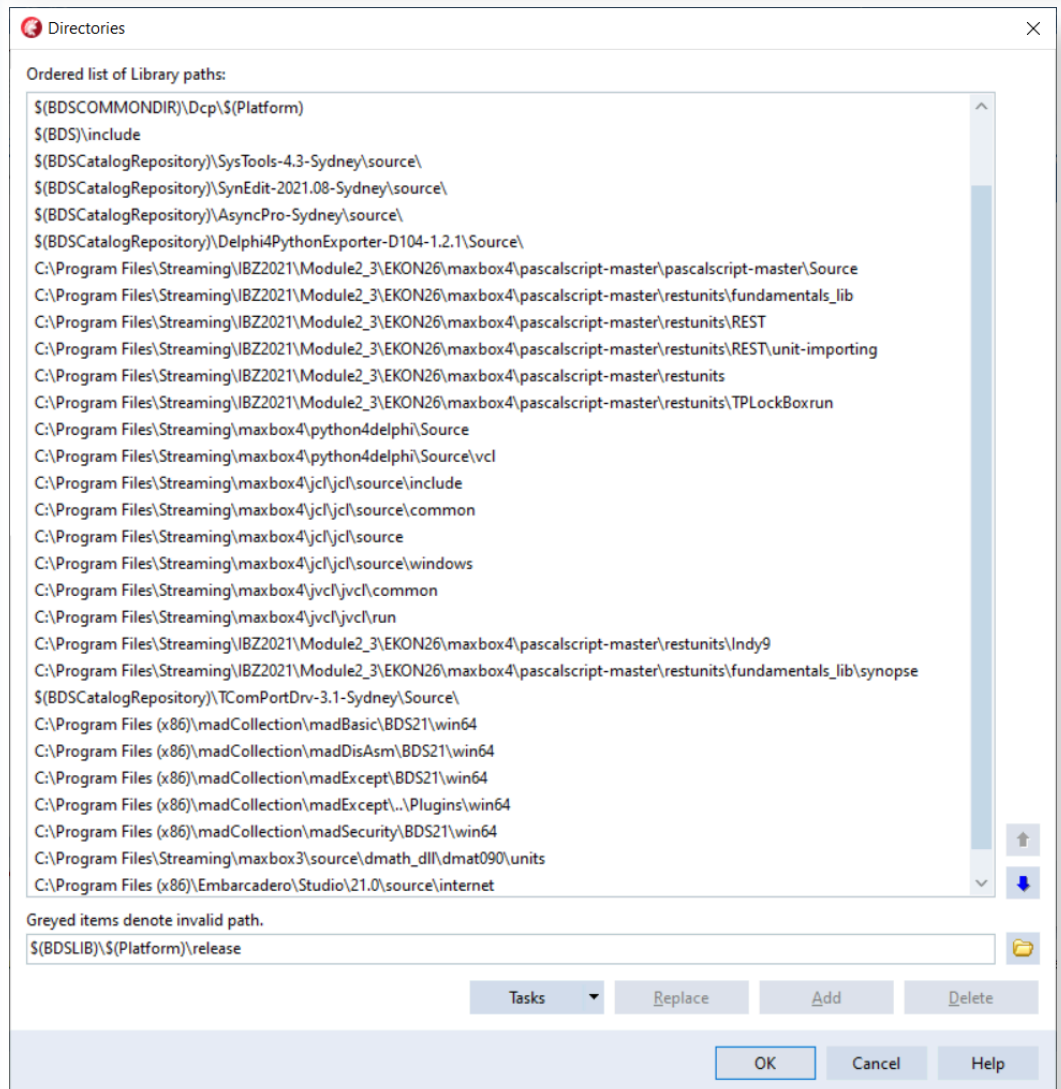
- **SysUtils.AnsiXXX**-Funktionen, wie z.B. `AnsiCompareStr`:
- Sie bleiben mit `string` und `float` zu `UnicodeString` deklariert.
- Bieten bessere Abwärtskompatibilität (keine Notwendigkeit, den Code zu ändern).

Die `AnsiXXXX`-Funktionen der Unit `AnsiStrings` bieten die gleichen Möglichkeiten wie die `SysUtils`. `AnsiXXXX`-Funktionen, funktionieren aber nur für `AnsiString`. Außerdem bieten die `AnsiStrings`. `AnsiXXXX`-Funktionen eine bessere Leistung für einen `AString` als die `SysUtils`. `AnsiXXXX`-Funktionen, die sowohl für `AnsiString` als auch für `UnicodeString` funktionieren, da keine impliziten Konvertierungen durchgeführt werden.

String-Informationsfunktionen:

- `StringElementSize` gibt die tatsächliche Datengröße zurück.
- `StringCodePage` gibt die Codepage der Stringdaten zurück.
- `System.StringRefCount` gibt die Anzahl der Verweise zurück.

RTL bietet viele Hilfsfunktionen, mit denen Benutzer explizite Konvertierungen zwischen Codepages und Elementgrößenkonvertierungen durchführen können. Wenn Entwickler die Funktion `Move` für ein `Zeichenarray` verwenden, können sie keine Annahmen über die Elementgröße treffen. Ein Großteil dieses Problems lässt sich entschärfen, indem man sicherstellt, dass alle `RValue`-Referenzen die richtigen Aufrufe an RTL generieren, um die richtigen Elementgrößen zu gewährleisten. In der Zwischenzeit habe ich den Import und die Liste in D10.4 gefunden:



Ein großes Durcheinander war oder ist das **Tencoding** (noch nicht fertig), denn **Tencoding** macht den eigentlichen Unterschied von **ANSI** zu Unicode:

- Standardmäßig wird die aktive Codepage des **Benutzers** verwendet.
- Unterstützt **UTF-8**.
- Unterstützt **UTF-16**, big und little endian.
- Unterstützung von **Byte Order Mark (BOM)**.
- Sie können absteigende Klassen für benutzerspezifische Kodierungen erstellen.

Sie müssen diese Schritte durchführen:

- Überprüfen Sie `char`- und `string`-bezogene Funktionen.
- Bauen Sie die Anwendung neu auf.
- Überprüfen Sie Surrogatpaare.
- Überprüfen Sie `String-Payloads`.

Nacht für Nacht erhielt ich viele **AV's (Access Violation)** dieser Art:

Ausnahmecode 0xc0000005 ist eine Zugriffsverletzung. Eine **AV** am Fehler-Offset `0x00000000` bedeutet, dass etwas im Code Ihres Dienstes auf einen Null-Zeiger zugreift. Sie müssen den Dienst debuggen, während er läuft, um herauszufinden, worauf er zugreift. Wenn Sie ihn nicht in einem Debugger ausführen können, dann installieren Sie zumindest ein **Exception-Logger-Framework** eines Drittanbieters, wie **EurekaLog** oder **MadExcept**, um herauszufinden, was Ihr Dienst zum Zeitpunkt der AV getan hat. In den meisten Fällen erhalten Sie eine AV in einer Unit mit einem Initialisierungsabschnitt von z.B. `Setmem` oder Speicherzuweisung.

maxbox

The screenshot shows a debugger window for a PythonEngine thread. The code being debugged is in `cyCompilerDefines.inc` and contains a series of conditional `raise` statements for different exception types. The line `raise Define(EPyImportError.Create('', s_type, s_value))` is highlighted in red, indicating the current execution point. The Events window at the bottom shows a list of events, including module loads and a first chance exception of type `EPyImportError` with the message `'ModuleNotFoundError: No module named 'sys''`.

```
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 ) )
  {SIFDEF MSWINDOWS}
  else if (PyErr_GivenExceptionMatches(err_type, PyExc_WindowsError^) <> 0) then
    raise Define( EPyWindowsError.Create('', s_type, s_value ) )
  {SENDIF}
  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 ) )
```

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 00007FFD313DC1F9. Exception class EPyImportError with message 'ModuleNotFoundError: No module named 'sys''. Process maxbox5_01alpha07.exe (13860)

Events Breakpoints Threads

Dies bedeutet, dass eine Ausnahme ausgelöst wurde, aber nirgendwo ein Catch-Handler dafür vorhanden ist. Dies ist höchstwahrscheinlich ein Programmierfehler, wahrscheinlich von Ihrer Seite. Es sieht so aus, als ob Sie eine **OpenCV**- oder **API**-Funktion aufgerufen haben, die mit einer `CV::Exception` fehlgeschlagen ist, aber Sie haben sie nicht abgefangen.

Dies würde normalerweise zu einem Absturz führen, aber da Sie sich in einem Debugger befinden, haben Sie die Möglichkeit, diese Ausnahme zu ignorieren. Genau das macht der Button Weiter in diesem Dialog. Anstatt eine **Exception** auszulösen, wird der Code einfach weiter ausgeführt, als ob nichts passiert wäre. Dies wird wahrscheinlich irgendwann fehl schlagen, da eine Fehlerbedingung nun ignoriert wurde.

Ein echter Horror war es, die `Format()`-Funktion nicht in **Delphi**, sondern in **PascalScript64** zur Laufzeit in ein Skript zu konvertieren, hier also die Funktion 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;
```

Im Kern ist es ein `wvsprintf`, aber die Frage war, ob **Unicode** verfügbar ist oder nicht?

Die **Byte Order Mark (BOM)** sollte zu Dateien hinzugefügt werden, um ihre Kodierung anzugeben, und die Funktion gibt einen `String` zurück.

Nach Neuimport und Neuaufbau und Umwandlung in `PwideCHAR` funktioniert es jetzt.

COMPATIBILITY COMPILATION

Wenn ein **32-Bit-Prozess** auf einer **64-Bit-Version** von **Windows** läuft und das Flag für große Adressen gesetzt ist, sind Zeiger im Bereich von 2-4 GB gültig. In diesem Fall könnte die Anforderung eines `varInt64`, wenn ich mich nicht irre, dazu führen, dass positive Werte im Bereich von 2-4GB zurückgegeben werden. Wenn `NativeInt` ein vorzeichenbehafteter 32-Bit-Int ist, würde dies zu einer Bereichsverletzung führen. Ich bin mir nicht sicher, ob das `NativeInt / NativeUInt` Cast dazwischen hier überhaupt benötigt wird. `NativeInt` und `NativeUInt` sind vorzeichenbehaftet/unvorzeichenbehaftet und ihre Größe hängt von der Zielplattform ab, also 32 oder 64 Bit. Deshalb glaube ich, dass die Cast-Änderungen nicht erforderlich sind. Die meiste Zeit beschäftigen Sie sich mit Pointern oder Assembler Code wie diesem:

```
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;
```

Directories

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\vc1
- 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

Tasks Replace Add Delete

OK Cancel Help

Ich habe Fortschritte gemacht, indem ich das TestApplications-Beispiel mit **CrossVCL 1.27 für Mac64 und Linux64** kompiliert habe.

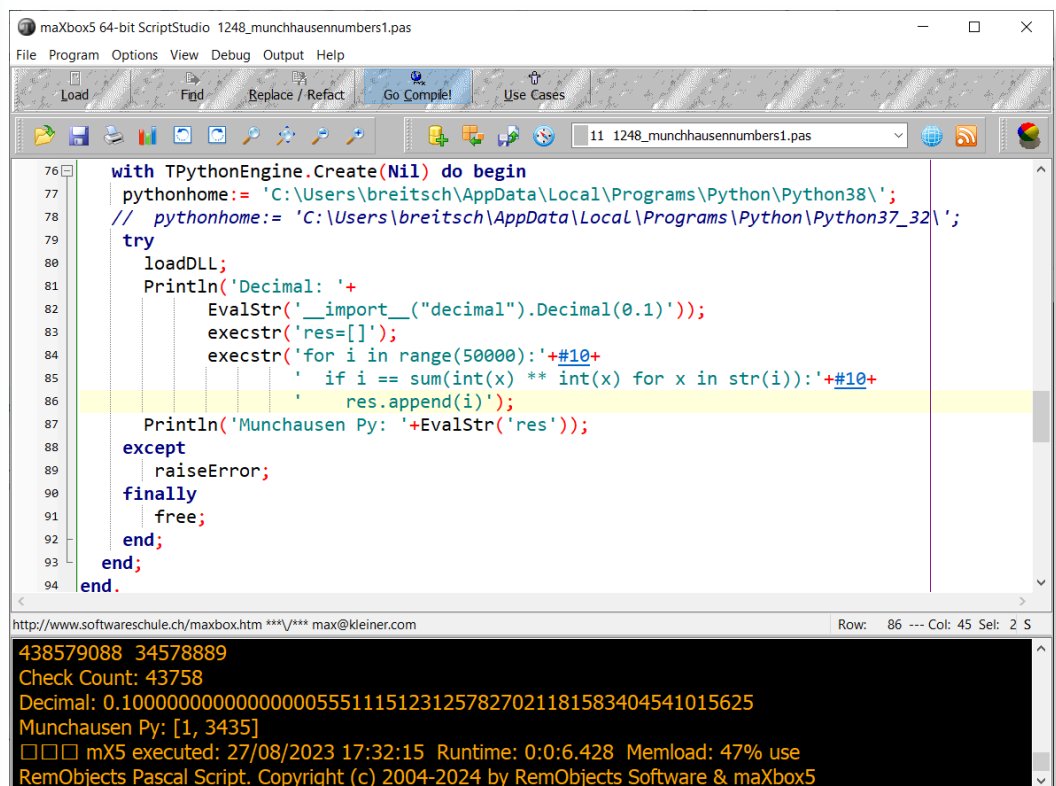
Wenn ich eine zweite Kompilierung in einer **CrossVCL** aus dem Menü wählen, erhalte ich den folgenden Fehler.

Erste Chance Exception bei \$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*) (0x00007ffefbfe038)
Vcl.Menus.TMenuItem.Click() (0x0000000201734960)
Vcl.Menus.TMenu.DispatchCommand(unsigned short) (0x0000000201734340,2)
Vcl.Forms.TCustomForm.WMCommand(Winapi.Messages.TWMCommand&) (0x0000000205039ff0,0x00007ffefbfe878)
:000000010001132B System::TObject::Dispatch(void*)
```

Und dann habe ich vielleicht aus Intuition einen Build gemacht und das AV ist verschwunden und ich habe meinen ersten Bildschirm bekommen, kompiliert und das Skript ausgeführt:



The screenshot shows the ScriptStudio IDE window titled "maxBox5 64-bit ScriptStudio 1248_munchhausennumbers1.pas". The code editor displays the following Pascal code:

```
76 with TPythonEngine.Create(Nil) do begin
77   pythonhome:= 'C:\Users\breitsch\AppData\Local\Programs\Python\Python38\';
78   // pythonhome:= 'C:\Users\breitsch\AppData\Local\Programs\Python\Python37_32\';
79   try
80     loadDLL;
81     Println('Decimal: '+
82       EvalStr('__import__("decimal").Decimal(0.1)'));
83     execstr('res=[]');
84     execstr('for i in range(50000):'+#10+
85       ' if i == sum(int(x) ** int(x) for x in str(i)):'+#10+
86       ' res.append(i)');
87     Println('Munchausen Py: '+EvalStr('res'));
88   except
89     raiseError;
90   finally
91     free;
92   end;
93 end;
94 end.
```

The output window at the bottom shows the following results:

```
438579088 34578889
Check Count: 43758
Decimal: 0.100000000000000055511151231257827021181583404541015625
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
```

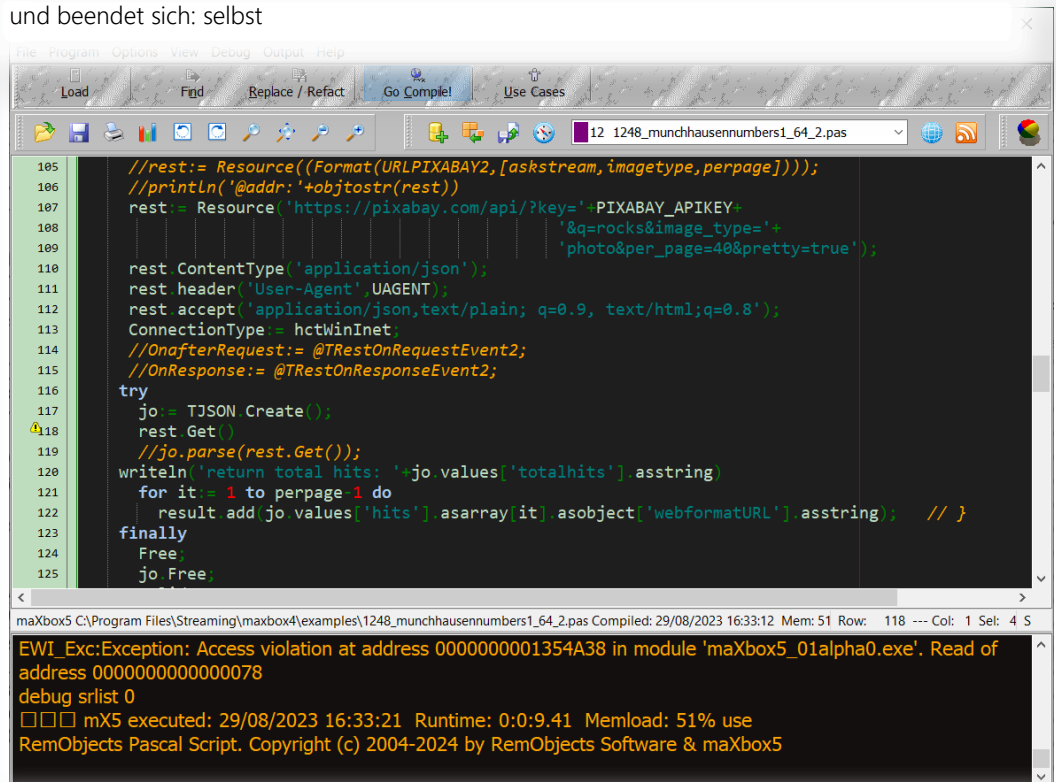
Abbildung 3

Eines der ungelösten Probleme ist es, eine Zugriffsverletzung zu bekommen, anstatt die Anwendung zum Absturz zu bringen. Es ist wie der Code in `uPSRuntime`, er konnte die Ursache für Halt nicht abfangen oder etwa beenden wie folgendes:

```
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;
```

Das Seltsame war, dass in einer der vorherigen Alpha-Versionen (*siehe unten*) das Abfangen der Access Violation vorhanden war, aber in der Zwischenzeit blieb die App stecken und beendet sich: selbst



```
105 //rest:= Resource((Format(URLPIXABAY2,[askstream,imagetype,perpage])));
106 //println('@addr:'+objtostr(rest))
107 rest:= Resource('https://pixabay.com/api/?key=' + PIXABAY_APIKEY
108                '&q=rocks&image_type='+
109                'photo&per_page=40&pretty=true');
110
111 rest.ContentType:= 'application/json';
112 rest.header('User-Agent' UAGENT);
113 rest.accept('application/json,text/plain;q=0.9, text/html;q=0.8');
114 ConnectionType:= hctWinInet;
115 //OnafterRequest:= @TRestOnRequestEvent2;
116 //OnResponse:= @TRestOnResponseEvent2;
117 try
118   jo = TJSON Create();
119   rest Get();
120   //jo.parse(rest.Get());
121   writeln('return total hits: '+jo.values['totalhits'].asString)
122   for it = 1 to perpage 1 do
123     result add jo.values['hits'].asarray[it].asobject['webformatURL'].asString; // }
124 finally
125   Free
126   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 000000001354A38 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

Nach dem Debuggen wurde mir klar, dass es sich um eine **First Chance Exception** handelt, die funktioniert, solange der Debugger mit `break` oder `continue` läuft, aber ohne Debugger verschwindet die App, ohne die AV auf der Ausgabe weiterzuleiten, wie z.B. AV at address xyz read of address 000. Sie können den Debugger anweisen, bestimmte Arten von Exceptions zu ignorieren. *Abbildung 3* zeigt die Sprachausnahmeoptionen von Delphi. Fügen Sie der Liste eine Ausnahmeklasse hinzu, und alle Ausnahmen dieses Typs und aller davon abhängigen Typen werden in Ihr Programm übernommen, ohne dass **Delphi** eingreift. Sie können Delphis "**erweiterte Breakpoints**" verwenden, um die Ausnahmebehandlung in einem bestimmten Bereich des Codes zu deaktivieren. Dazu setzen Sie einen BreakPoint in der Codezeile, in der die IDE Ausnahmen ignorieren soll.

Werfen wir nun einen letzten Blick auf eine ausgewählte Testanwendung/ein ausgewähltes Testskript mit einzelnen zu übersetzenden Texten aus Ihren eigenen Daten. Wir haben zwei nützliche Funktionen geschrieben. Die erste gibt den mit einer Zielsprache übersetzten Text zurück. Die zweite akzeptiert einen Satz als Argument mit der Spracherkennung als Parameter "auto". Dann wird der Text in **JSON** oder als Datei angezeigt.

Oder Sie können von **Assembler** zu **Pascal-Code** wechseln:

```
{$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;
```

Der Fehler "unit is compiled with a different version of..." ist sehr ärgerlich. Er tritt in einer Situation wie der folgenden auf:

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

Unit B und **C** sind bereits kompiliert, aber aus irgendeinem Grund ist die Quelle von **Unit B** nicht verfügbar. Nun wird **Unit C** geändert und deshalb neu kompiliert. Die **dcu** von **Unit C** unterscheidet sich nun von der **Unit C**, die von **Unit B** verwendet wird, also muss **Unit B** ebenfalls neu kompiliert werden. Aber leider ist der Quelltext von **Unit B** nicht verfügbar, so dass der Compiler aufgibt.

UNIT-UNICODE-PRÜFUNG

Mit dieser App kann man Text aus vielen verschiedenen Sprachen übersetzen oder erkennen und mX5 mit Unicode testen. Deshalb möchte ich, dass dieser Endpunkt nahtlos in googletrans integriert wird und zwischen den Endpunkten umschaltet, wenn ein Endpunkt mit 4xx/5xx-Fehlern konfrontiert ist.

```

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

function Text_to_translate_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;

```

Der kostenlose Dienst von **Google** übersetzt Wörter, Sätze, Texte und Webseiten sofort aus dem Englischen in über 100 andere Sprachen.
So rufen wir die Funktion auf:

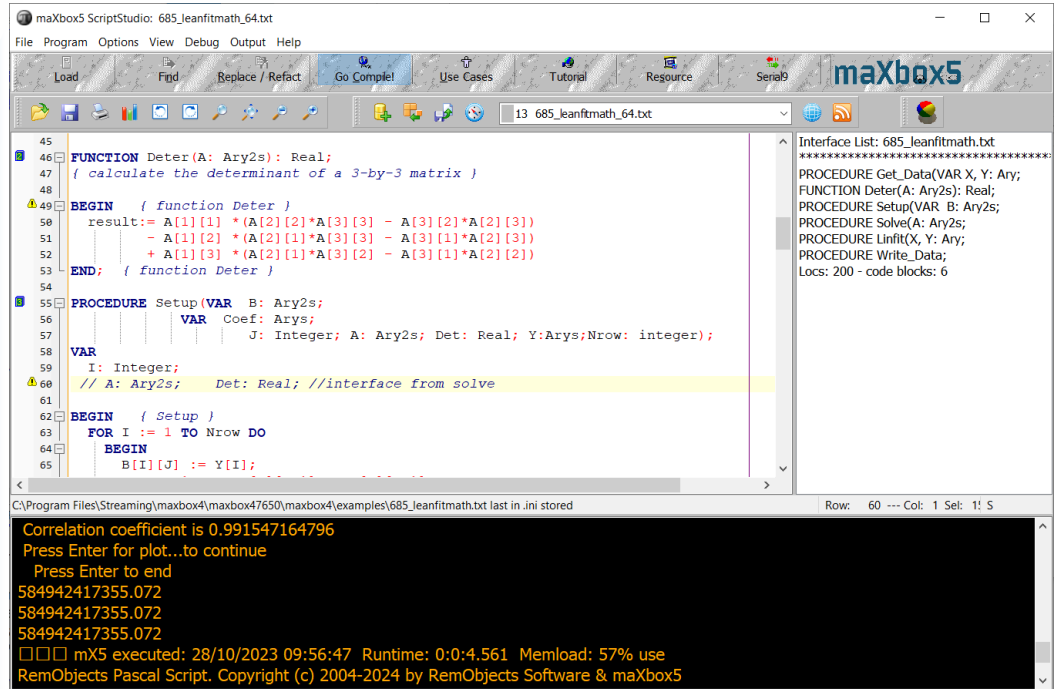
```

atext:= 'bonjour mes amis da la ville';
writeln(utf8ToAnsi(Text_to_translate_API2(AURL,'dict-chrome-
                                          ex','auto','es',atext)));

```

und das Ergebnis: Server: ESF
["Hola mis amigos en la ciudad","fr"]

Google Translate ist jetzt eine Form von **Augmented Reality** und wird für Bildungszwecke angepasst. Diese Anwendung bietet den Nutzern Werkzeuge zum Übersetzen zwischen Sprachen und jetzt auch eine Bildoption; die Nutzer fotografieren ein Schild, ein Stück Papier oder Form von geschriebenem Text und erhalten eine Übersetzung in der Sprache ihrer Wahl.



Diese visuelle Technik, die zum Verständnis dessen, was einzelne Texte darstellen, verwendet wird, nennt man semantische Analyse. Zum Thema: [https://en.wikipedia.org/wiki/Semantic_analysis_\(Linguistik\)](https://en.wikipedia.org/wiki/Semantic_analysis_(Linguistik))

Ich habe einen weiteren Endpunkt zum Testen mit **Unicode** im Quellcode einer der Erweiterungen von google translate auf **VSCoDe** gefunden.

```

"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
}
    
```

Das Ergebnis sieht in etwa so aus:

```

[[["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,[]]
    
```

für die Anfrage: https://translate.googleapis.com/translate_a/single?client=gtx&dt=t&sl=en&tl=ja&q=Hello, wie geht es Ihnen heute?

Und etwas in dieser Art:

```

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

String unicode (\uxxxx) encoding und decoding.

Nach einigen Tests mit Request Headers und F12 Tools - Inspect (siehe unten), habe ich die Lösung für den verstümmelten Text gefunden, der es sein kann. Setzen Sie einfach die User-Agent-Header auf die, die Google Chrome verwendet.

EXAMPLE:

```
import requests
word = 'اذه لعفت اذامل'
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 Detected]: ' + request_result['src'])
except:
    pass
```

maxbox

The screenshot shows the maxbox5 64-bit ScriptStudio IDE. The main window displays Pascal code for a Runge-Kutta 4th order method. The code includes comments in German and a case statement for error handling. The console at the bottom shows the execution results, including the runtime (0:0:1.583) and memory load (53% use).

```
680 add('Value of X at'+
681         Floattostr(lowerlimit)+' : '+Floattostr(Initialvalue));
682 add('Value of X'' at '+Floattostr(aLowerLimit)+
683         ' : '+Floattostr(InitialDeriv));
684 add('Number of intervals: ' + intostr(trunc(aUpperlimit/CalcInterval));
685 add(' ');
686 add(' t          Value of X          Derivative of X          ');
687 RungeKutta2ndOrderIC(aLowerLimit, aUpperLimit, InitialValue, InitialDeriv,
688         ReturnInterval, CalcInterval, aError,
689         @PendulumFunc, @PendulumCallBackFunc);
690 selstart:=0; sellength:=0; {move back to top of memo display}
691 case aError of
692 1 : add('The number of values to return must be greater than zero.');
```

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

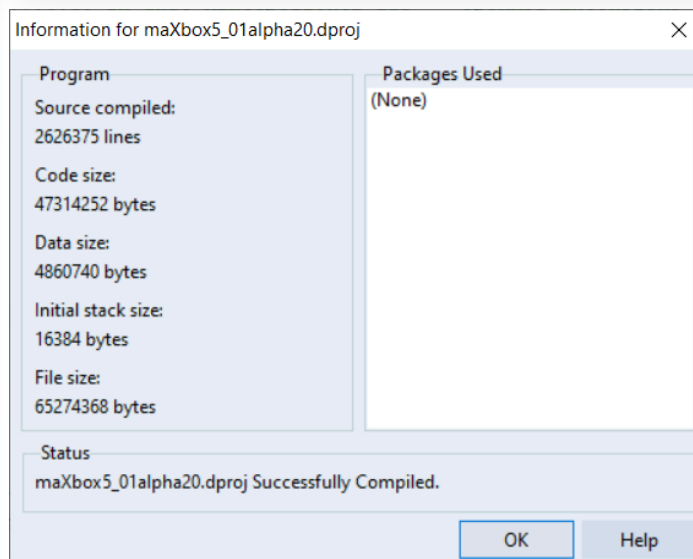
Ver: 5.0.1.22 (501). Workdir: C:\Program Files\Streaming\maxbox4\maxbox47650\maxbox4

SCHLUSSFOLGERUNG:

Natürlich ist die **64-Bit-Box** noch nicht fertig. Die aktuelle Version **5.0.1.22** ist eine frühe Beta-Version. Als nächster Schritt stehen Methodenzeiger (`func pointers`) und `TEncoding` auf der Liste. Außerdem wird bei der Entdeckung einer mit der Überladungsdirektive markierten Funktion nach einem neuen Funktionsnamen gefragt und dann Wrapper-Code erzeugt, der den neuen Methodennamen auf die ursprüngliche Version abbildet, aber bei einer Umleitung erhalten wir eine **Access Violation**.

Im Allgemeinen ist es unwahrscheinlich, dass ein System jenseits der 32 Bit mehr Nutzen bietet, möglicherweise eine kleine Geschwindigkeitssteigerung, mehr Speicher und mehr Register. Verlassen Sie sich aber nicht auf 64-Bit, um eine langsame Anwendung zu beschleunigen: Für größere Geschwindigkeitssteigerungen müssen Sie immer noch algorithmische Änderungen vornehmen.

64-Bit ist keine Wunderwaffe. Ansonsten müssen Sie nur dann etwas ändern, wenn Sie bereits an eine der Grenzen von 32-Bit gestoßen sind oder wenn Sie Plugins für 64-Bit-Anwendungen entwickeln oder einfach mit einem 64-Bit-Betriebssystem kompatibel sein wollen.

**REFERENZEN:****Kompiliertes Projekt:**

<https://github.com/maxkleiner/maXbox4/releases/download/V4.2.4.80/maXbox5.zip>

Vorbereitung:

<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>



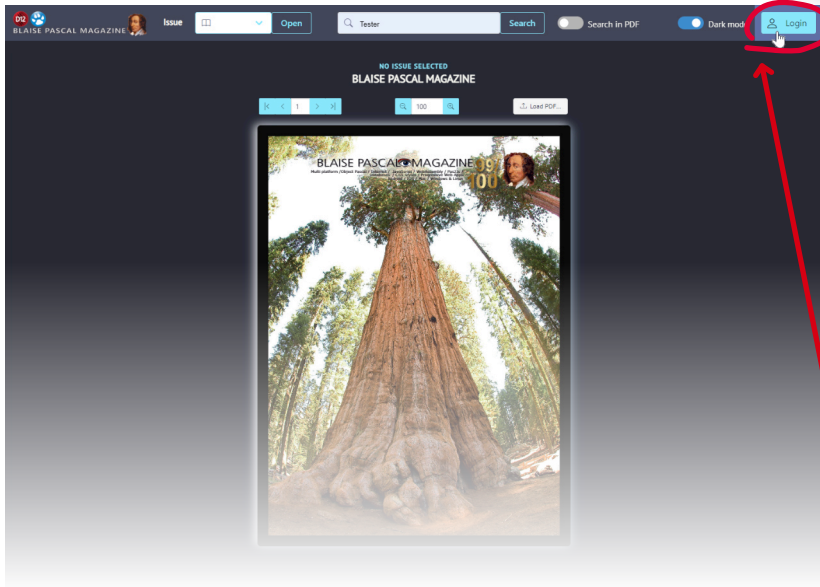
THE NEW SUBSCRIPTION MODEL BLAISE PASCAL MAGAZINE

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

The screenshot shows the Blaise Pascal Magazine website interface. The browser address bar displays 'library.blaisepascalmagazine.eu'. The page header includes the magazine logo, a navigation menu with 'Issue 66' and 'Open', a search bar with 'Alan Turing', and a 'Search' button. Below the header, there is a 'NO ISSUE SELECTED' message and the magazine title 'BLAISE PASCAL MAGAZINE'. The main content area features a large featured image of a portrait of Blaise Pascal, surrounded by smaller portraits of other historical figures. The background of the page is filled with a repeating pattern of the text 'AI AI AI'. On the left side, there is a sidebar titled 'ARTICLES' with a list of article titles and page numbers. The main content area also displays a list of article titles and page numbers, including 'From the editor', 'Majorana, the new solution for QuantumBits?', 'Video Effects and Animations creating video effect without hardly any coding', 'Different Kind of Logic / Socrates - Humor', 'FreePascal - Report - Part Two A new ReportingEngine for LAZARUS', and 'Working with TChart'. At the bottom of the page, there is a registration link: <https://www.blaisepascalmagazine.eu/register/>

USE WHERE EVER THE INTERNET IS

DAS NEUE **KOSTENLOSE** EXTRA INTERNET PDF LIBRARY ABONNEMENT

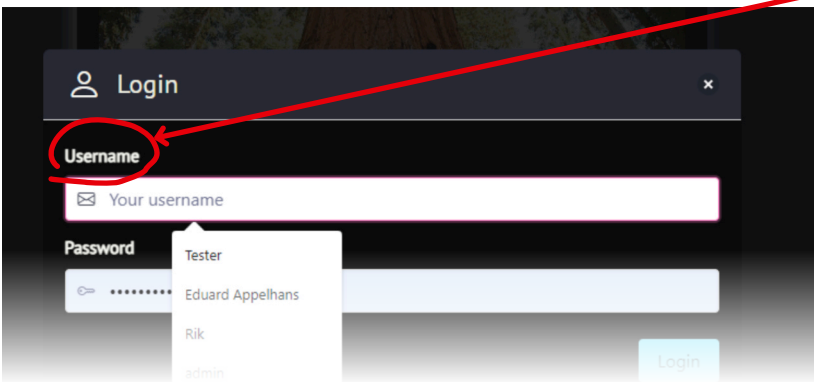
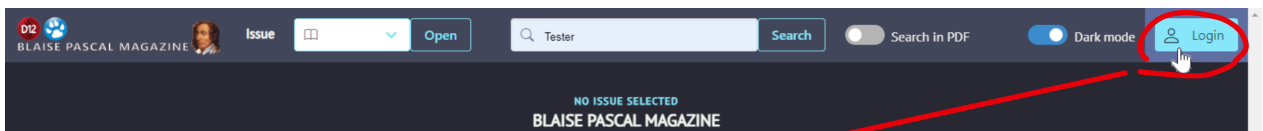


Ist eine kostenlose Ergänzung zu Ihrem normalen Abonnement. Sie können alle Hefte ansehen und durchsuchen, auch in allen Heften. Dieser Artikel soll Ihnen helfen zu sehen, was möglich ist. Er ist über das Internet verfügbar und kann jedes PDF öffnen. Wenn Sie ein Abonnement haben (zum Herunterladen oder in gedruckter Form), können Sie es ein Jahr lang kostenlos nutzen, d.h. für die Dauer Ihres Abonnements.

Sie erhalten automatisch ein Login und ein Passwort.

Wenn Sie zusätzliche Wünsche haben oder Verbesserungen vorschlagen möchten, lassen Sie es mich wissen: editor@blaisepascal.eu

1 Zum Starten: In der rechten oberen Ecke können Sie sich anmelden.

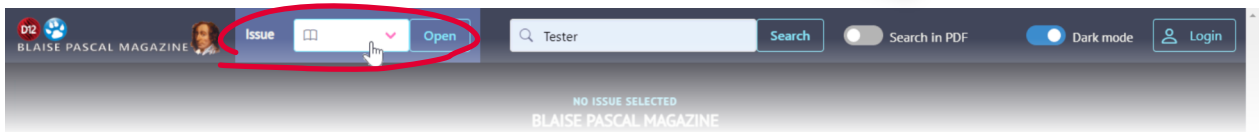


2 Geben Sie Ihren Benutzernamen ein. Diesen haben Sie bekommen mit der Korrespondenz über das Abonnement.

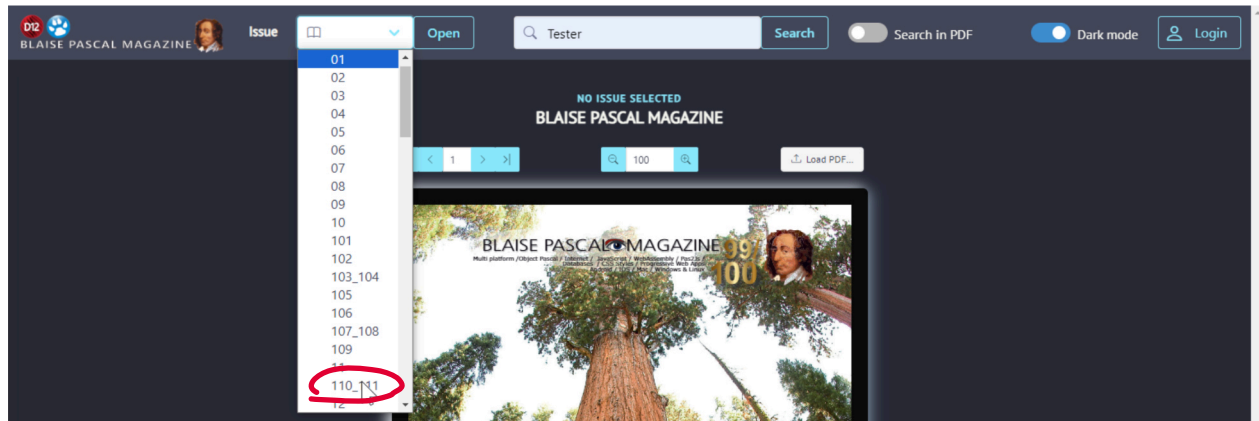


3 Geben Sie das Passwort ein. Klicken Sie auf Login

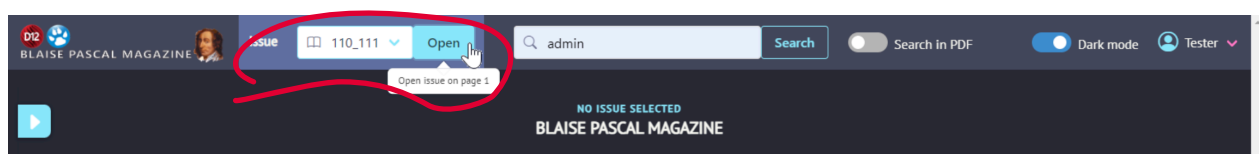




4 Um ein Heft zu öffnen: Klicken Sie auf die Dropdown-Liste



5 Um ein Heft zu öffnen: Klicken Sie auf die Dropdown-Liste

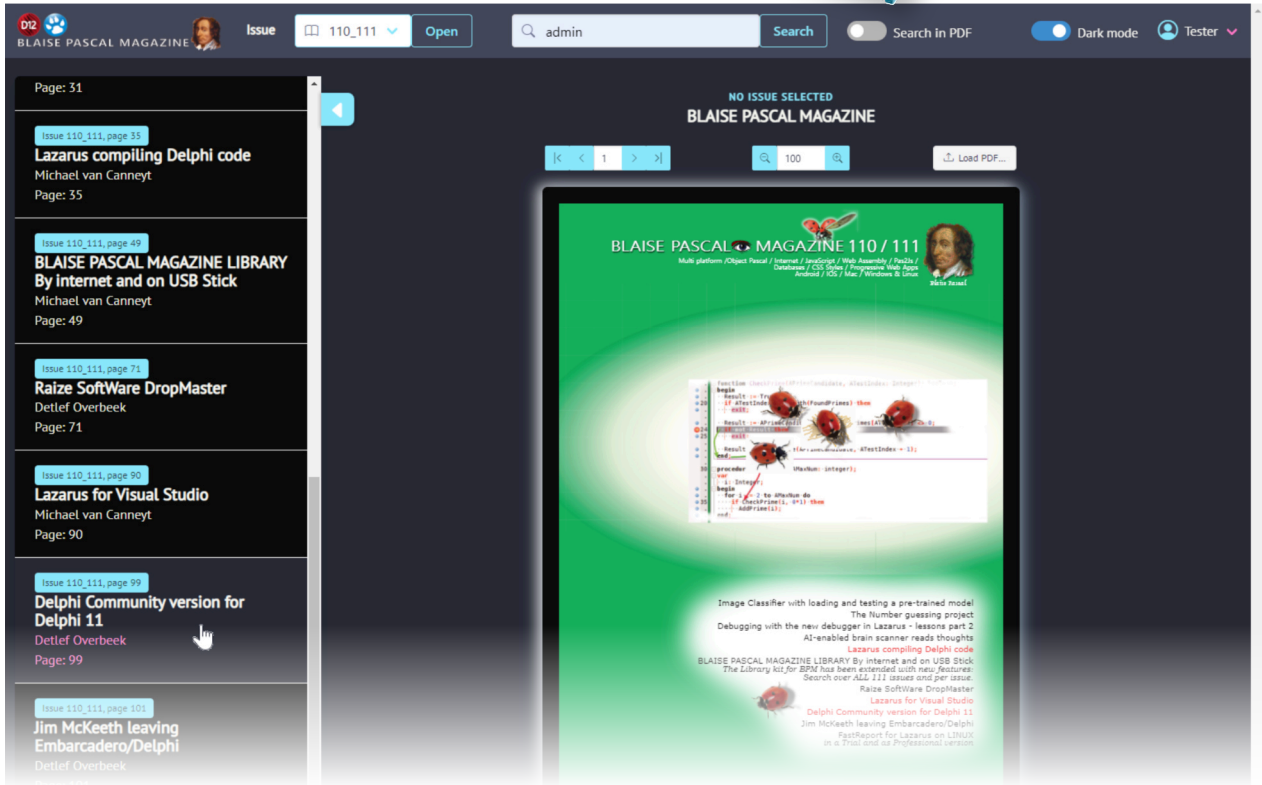


6 Es dauert einige Zeit und öffnet dann die erste Seite des jeweiligen Heftes.



7 Die erste Seite wird verfügbar.



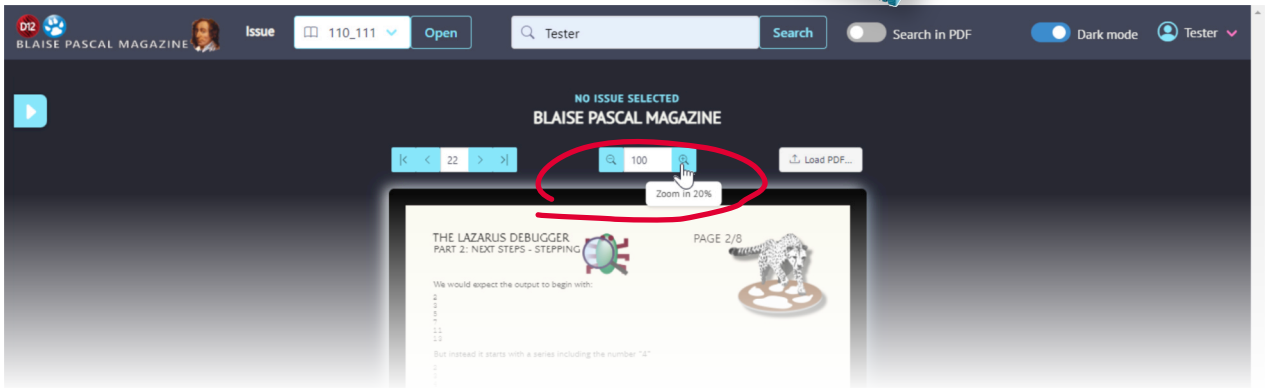


8 Die Liste auf der linken Seite zeigt die Artikel dieses Heftes

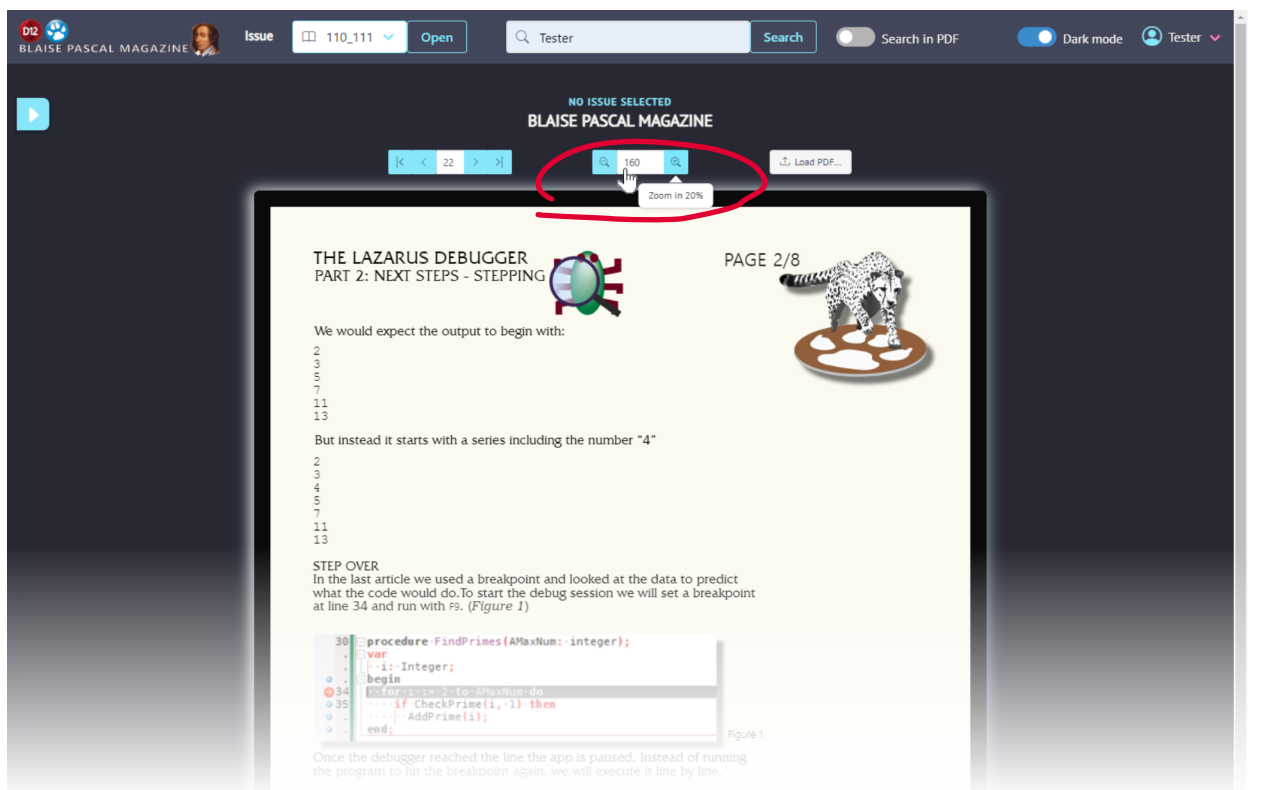


9 Der ausgewählte Artikel erscheint

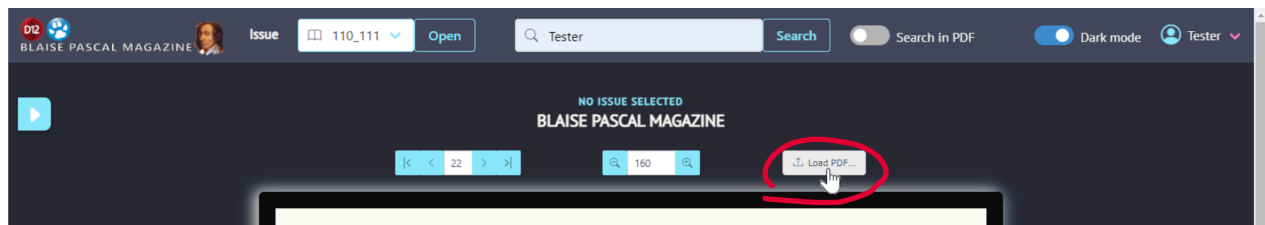




10 Sie können die Größe des pdf selbst vergrößern



11 Es ist einfacher, die Details zu sehen...



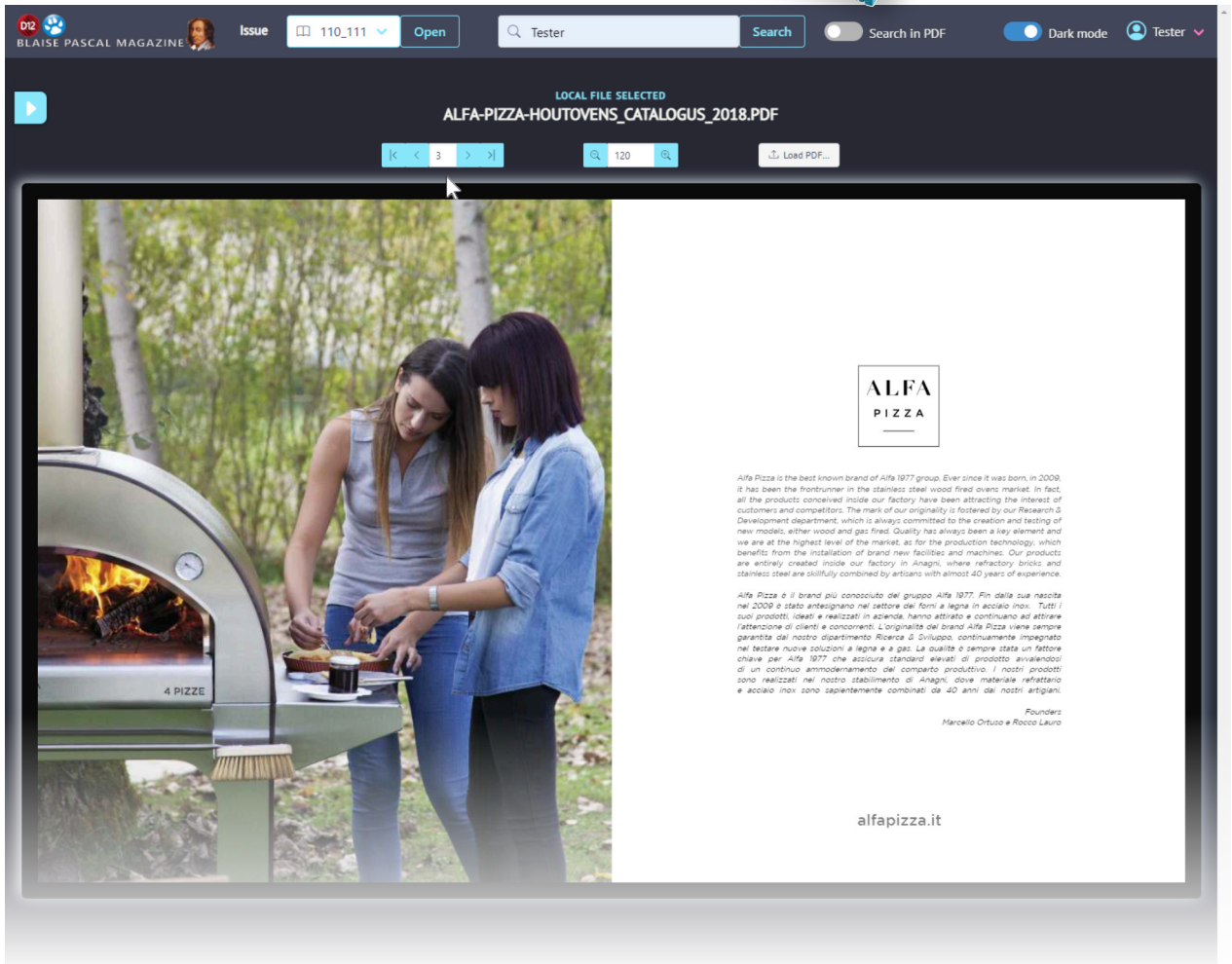
12 Sie können auch jede Art von PDF laden





13 Sie können auch jede Art von PDF laden, von Ihrer Festplatte, Ihrem USB-Stick oder einer anderen Quelle...





14 Die geöffnete pdf-Datei

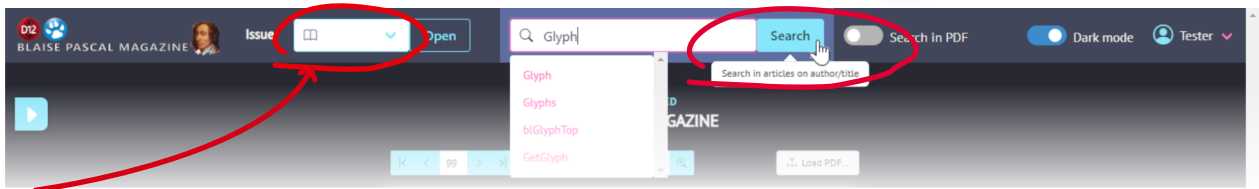


15 Dunkler Modus, wenn Sie es bevorzugen

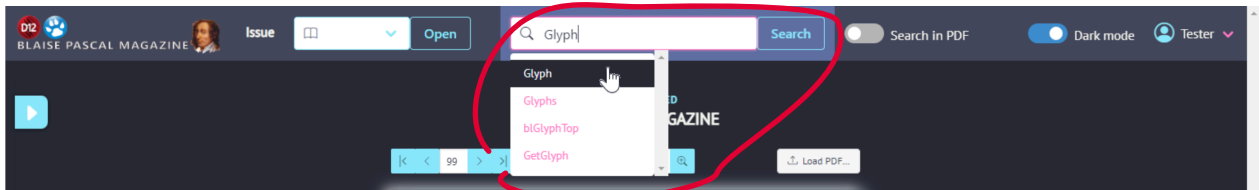


16 Hell ist natürlich auch möglich





17 Diese Liste muss leer sein, wenn Sie in ALLEN Heften nach Textelementen suchen wollen. Dies muss geschehen, bevor Sie den Text eingeben können, nach dem Sie suchen möchten



18 Danach können Sie das Wort oder den Text eingeben



19 Hier sehen Sie eine Vergrößerung des Wortes im Text, das aus der Liste links ausgewählt wurde. Der Artikel wird direkt angezeigt





Issue Open Search

Blaise Pascal Magazine | Site ab...
blaisepascalmagazine.eu

BLAISE PASCAL MAGAZINE

HOME YOUR DOWNLOADS **SUBSCRIPTIONS** EVENTS SHOP AI MATH & GAMING PASSWORD LOGIN SOFTWARE CONTACT LOGIN

Issue 112 UK Issue 110 / 111 DE Issue 110 / 111 UK Issue 109 Deutsch Issue 109 UK Issue 107/108 Deutsch Issue 107/108 UK Issue 106 UK Issue 105 Issue 103/104 Issue 10

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 downtime is expected to last no longer than 45 minutes.

SPECIAL OFFERS

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

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

Database / CSS Styles / Progressive Web Apps
Android / iOS / Mac / Windows & Linux

LIB-STICK ON USB CREDIT CARD

BLAISE PASCAL MAGAZINE

THE NEW INTERNET BLAISE PASCAL LIBRARY 2023

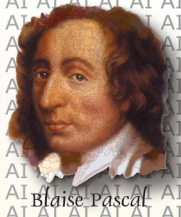
<https://library.blaisepascalmagazine.eu/>

👁️ Wenn Sie auf das Logo klicken, gelangen Sie direkt auf die Website des Blaise Pascal Magazine



BLAISE PASCAL MAGAZINE 112

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

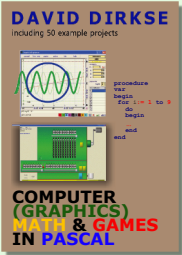


Price: € 75,00
LAZARUS HANDBOOK
POCKET + PDF AND
SUBSCRIPTION
ex Vat and Shipping



Ch...
Interview with the new C...
H-BOT...
...rabbit?
...please
...marker.
...ed robot
...Lazarus part 3
...sion 3.0 RC 2

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



EINFÜHRUNG

Ein Besucher meiner Website (davdata.n1/math) fragte sich nach der Anzahl der Lösungen eines binären Rätsels. Unten ist ein binäres Rätsel abgebildet, wie man es in Zeitungen findet. Links ist das ursprüngliche Rätsel, rechts ist der gelöste Zustand.

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

Die Aufgabe besteht darin, die leeren Zellen mit einer 0 oder einer 1 zu füllen, wobei folgende Einschränkungen gelten

- Eine Zeile oder Spalte darf nicht mehr als zwei aufeinanderfolgende Nullen oder Einsen enthalten.
- Jede Zeile und jede Spalte muss die gleiche Anzahl von Einsen und Nullen enthalten
- Keine zwei Spalten und keine zwei Zeilen dürfen gleich sein

Binäre Puzzles gibt es in den Größen 4x4, 6x6, 8x8 (wie oben), 10x10, 12x12, 14x14.

Ein gutes Rätsel hat nur eine Lösung.

FRAGEN SIND:

- Wie prüft man ein Rätsel auf Einzigartigkeit?
- Welche Ziffern können in einem gelösten Rätsel entfernt werden, um eine eindeutige Lösung zu erhalten?
- Wie viele Rätsel sind für ein nxn großes Rätsel möglich?

Die letzte Frage kann wie folgt umformuliert werden

- Wie viele Lösungen hat ein Rätsel mit nur leeren Zellen?

Dieses **Delphi-Projekt** versucht, die Antwort auf die letzte Frage zu finden.

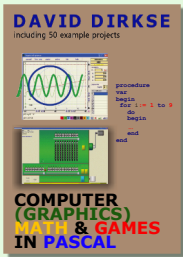
Wie in vielen anderen Fällen kann dieses Problem auf analytische oder numerische Weise angegangen werden. Es wird der numerische Ansatz verwendet. Alle möglichen Lösungen werden generiert und gezählt.

Nehmen wir ein 6x6-Rätsel als Beispiel. 6 Bits stehen für eine Zahl zwischen 0 und 63.

Aufgrund der Einschränkungen sind nur 14 dieser Nummern gültig, siehe unten:

1	0	0	1	0	1	1	complements wcount = 14 • search may stop if Acounter[1] reaches this number (wcount / 2 + 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	





Um überflüssige Arbeit zu vermeiden, werden diese gültigen Zahlen einmal generiert und im Array `numbers[1..]` gespeichert. Anstatt die 6-Bit-Werte zu verwenden, können wir auf Zeilen und Spalten verweisen über den Index des Arrays `numbers[]`.

Wenn Zeilen mit gültigen Zahlen gefüllt werden, können Spalten mit ungültigen Zahlen entstehen.

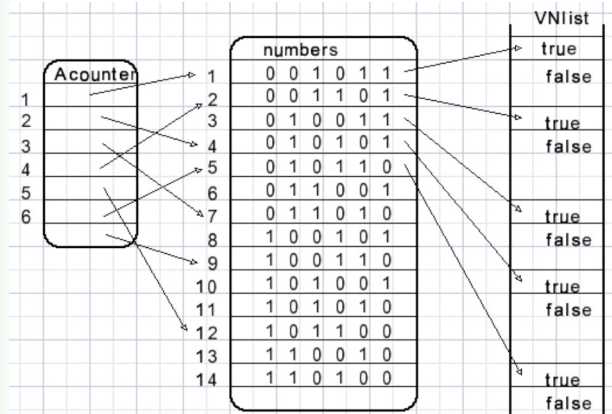
Daher ist es praktisch, jede Zahl `[0..63]` zu registrieren, wenn sie eine gültige Zahl darstellt. Das Array `VNlist[number]` von Boolean hat den Wert **true** für eine gültige Zahl

Um alle möglichen Lösungen zu generieren, wird ein Zählersystem benötigt.

Dieser Zähler heißt `Acounter[1..bitcount]`

der Indizes für das Zahlenfeld enthält.

Jedes Element des `Acounters` kann als eine Ziffer der Zahl `Acounter` betrachtet werden.



Bei einem $n \times n$ -Rätsel hat `Acounter` n Elemente.

Die VN-Liste hat 2^n Elemente.

Durch die Verwendung des Zahlenfeldes sind alle Werte gültig.

Schon bei der Auswahl einer neuen Zeile können wir Spalten mit mehr als zwei aufeinanderfolgenden Nullen oder Einsen vermeiden. Zu diesem Zweck gibt es

```

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}

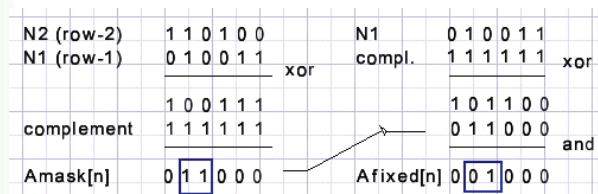
```

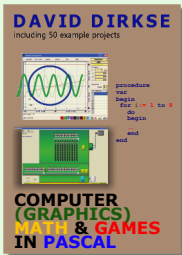
```

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)





Für Reihe n müssen die Bits 3 und 4 0,1 sein, um 3 aufeinanderfolgende Einsen oder Nullen in einer Spalte zu vermeiden.

Wenn alle 6 Zeilennummern ausgewählt sind, müssen diese Prüfungen durchgeführt werden

- ❶ Eine Zahl darf nur einmal in den Zeilen vorkommen
- ❷ Jede Spalte muss eine gleiche Anzahl von Einsen und Nullen enthalten.
- ❸ Eine Spalte darf nicht drei (*oder mehr*) aufeinanderfolgende Einsen und Nullen enthalten
- ❹ Eine Zahl darf nur einmal in den Spalten vorkommen

- ❶ Jede neue Zeile wird mit den vorherigen Zeilen verglichen, um ein erneutes Auftreten zu vermeiden.
- ❷ Für die Spaltenüberprüfungen müssen die Spalten als Zeilen geschrieben werden, was durch Spiegelung über die Diagonale von rechts oben nach links unten geschieht. Dies ist ein zeitaufwändiger Prozess. Bevor also die Spalten als Zeilen geschrieben werden die Zeilen summiert und diese Summe wird daraufhin geprüft, ob sie $(n/2)(2^n - 1) = 189$ für 6x6 Rätsel.
Ein 6x6-Rätsel hat drei 1en pro Spalte, also ist die Summe $3 * (111111)_{bin} = 3 * 63 = 189$.
Nur wenn dieser Test bestanden wird, werden die Spalten als Zeilen geschrieben.
- ❸ Eine gültige Spaltennummer wird einfach dadurch angezeigt, dass die `VNlist[number]` wahr ist.
- ❹ Die Prüfung auf mehrfaches Auftreten erfolgt durch den Vergleich aller Zahlen.

Diese Zeilensummenprüfung spart 65% der Zeit für ein 8x8-Rätsel.

BEACHTEN SIE, dass die Zeilennummern und die Lösungen in Komplementen erscheinen. Für jede Lösung gibt es also eine weitere, bei der alle Zellen komplementär sind.

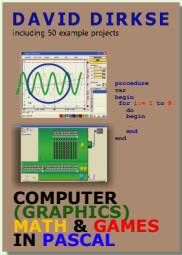
Die Hälfte der Zeit wird gespart, wenn man nur die ersten 50% der Zahlen in Reihe 1 zählt und die Lösungen in 2er-Schritten. Bei 8x8 Rätseln wird die Anzahl der Lösungen in 4,3 Sekunden gezählt. Bei einem 10x10-Rätsel wurden etwa 4 Millionen Lösungen pro Minute gezählt. Die erwartete Zählzeit für alle Lösungen beträgt 16 Stunden.

ERGEBNISSE

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	?

Es sieht so aus, als ob die Anzahl der Lösungen für leere Rätsel mit jeder weiteren (geraden) nxn-Größe um einen Faktor 1000 zunimmt.





DAS PROGRAMM

Der Kern des Programms ist:

```
function NextAccount : boolean;
```

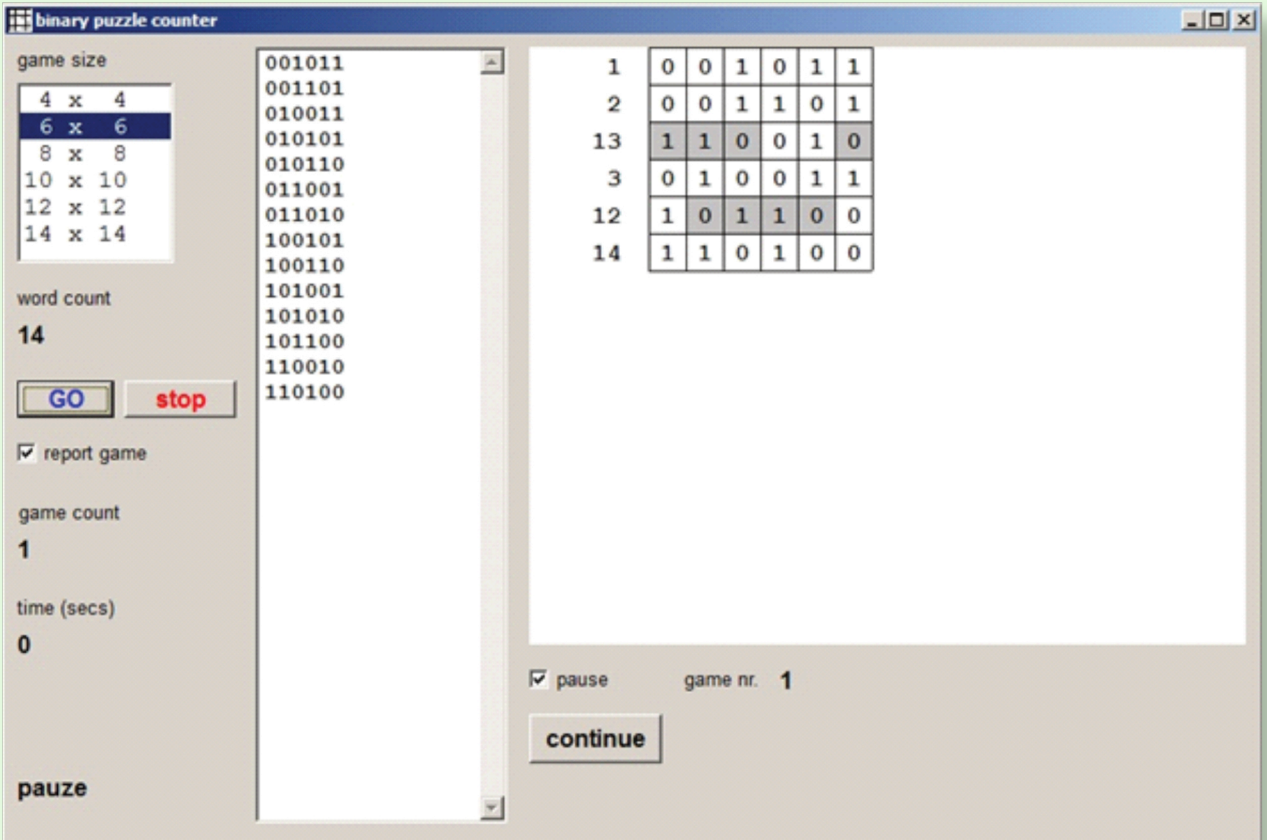
Aktualisiert den Acounter und gibt true zurück, wenn ein neuer Wert gesetzt wurde (*kein Überlauf*). Wcount (*word count*) ist die Anzahl der gültigen Werte im Array numbers[] . LInc ist ein Label.

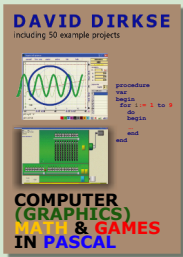
Bitcount = 4 für 4x4-Rätsel, 8 für 8x8....

Ein falscher Exit (*Ende der Suche*) tritt ein, wenn Acounter[1] auf eine Zahl aktualisiert wird, bei der das **MSB** (*most significant bit*) gesetzt ist.

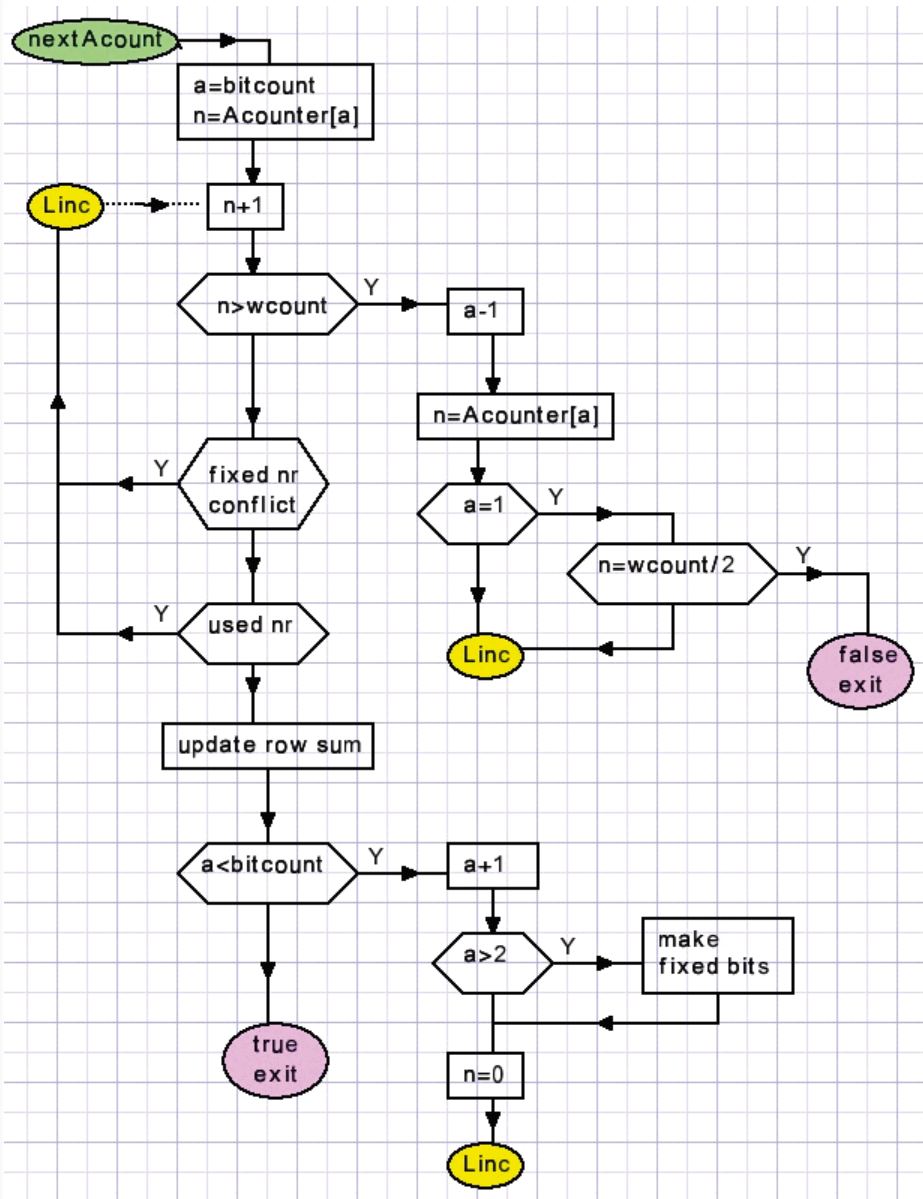
Dadurch wird verhindert, dass die Suchzeit verdoppelt wird, während nur die Komplemente der früher gefundenen Lösungen gezählt werden.

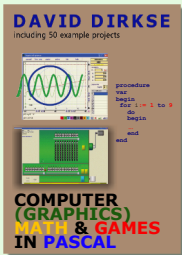
Ein wahrer Exit findet statt, wenn Acounter[bitcount] ohne Konflikt aktualisiert wird.





Dies ist das Flussdiagramm:





Beachten Sie, dass in diesem Fall die Verwendung eines `Labels` und von `GOTO`-Anweisungen weitaus lesbarer Code als die strukturierten Programmieranweisungen `while` oder `repeat`.

ANDERE FUNKTIONEN UND PROZEDUREN:

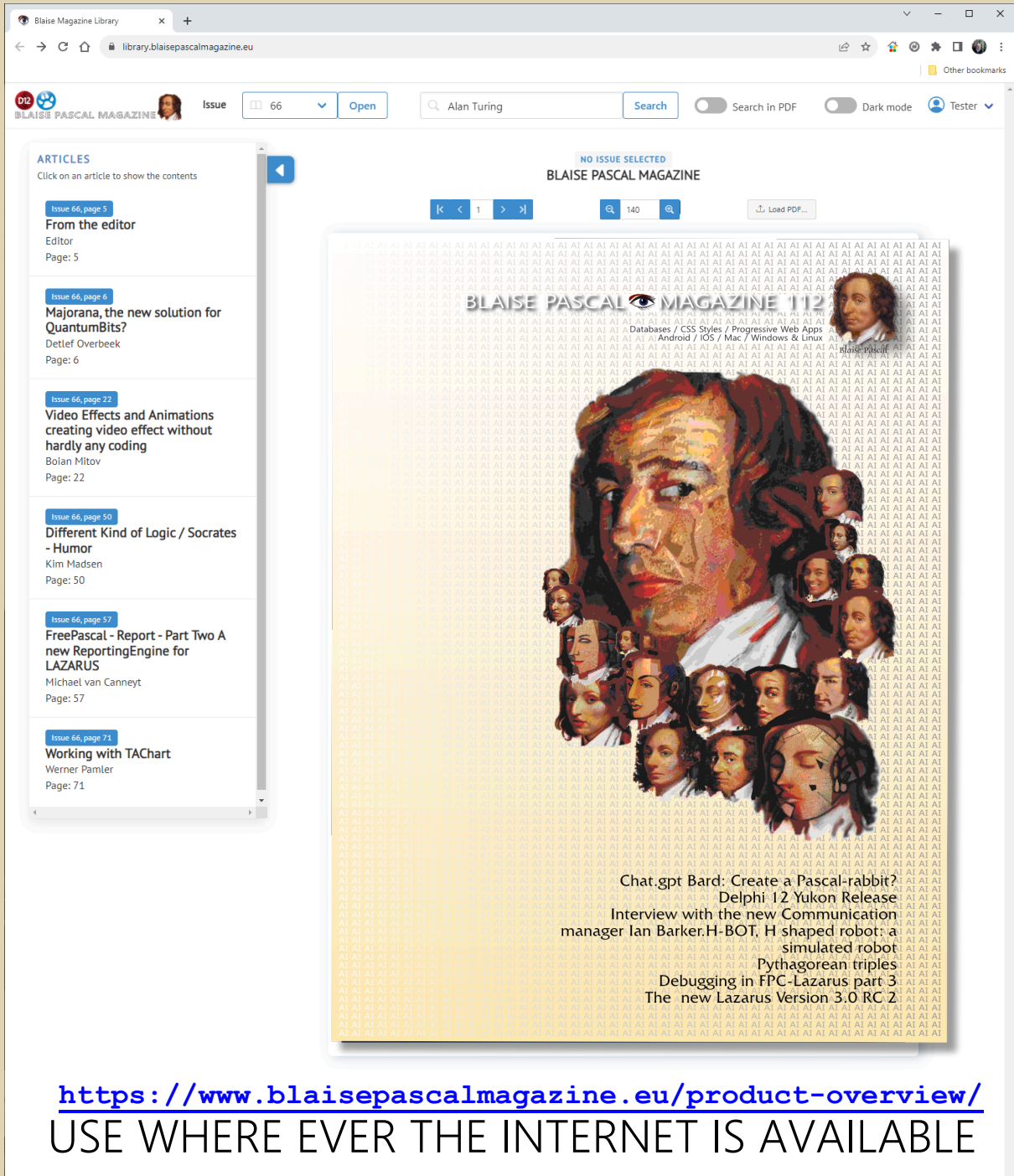
- **function testValid(w : word) : boolean;**
Gibt true zurück, wenn eine Zahl (w) gültig ist (*siehe Einschränkungen*)
- **procedure MakeNumbers;**
Erzeugt die `List numbers[]` und die `VNlist[]`.
- **procedure makeFixedbits(ai : byte);**
Setzt die Einträge `AMask[ai]` und `AFixed[ai]`, um falsche Zahlen in einer Spalte zu vermeiden.
- **function UsedWord(a : byte; wix : word) : boolean;**
Gibt true zurück, wenn der Index `wix` nicht in `Acounter[1..a-1]` vorhanden ist. Verhindert identische Zeilen.
- **Prozedur setGame1;**
Wird am Anfang aufgerufen, um das erste Spiel zu liefern.
- **Prozedur makeVGame;**
Überträgt Spalten auf Zeilen im Array `VGame[]`.
- **function checkVGame : boolean;**
Gibt true zurück, wenn das `VGame`-Array gültige Zahlen enthält.

Damit ist die Beschreibung des Zählers für binäre Rätsellösungen abgeschlossen. Bitte lesen Sie den Quellcode für weitere Details.



THE NEW SUBSCRIPTION MODEL OF BLAISE PASCAL MAGAZINE

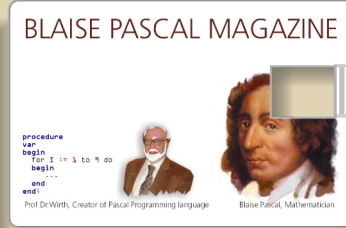
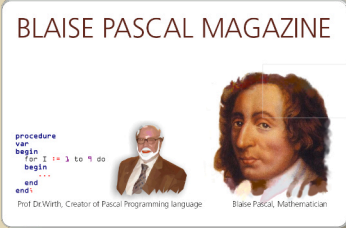
1. **SUBSCRIPTION**: PER YEAR - NOTHING CHANGES ISSUES STARTING AT THE LATEST ISSUE AVAILABLE +1 YEAR / CODE INCLUDED € 70,00 FOR ALL COUNTRIES INCLUDED **INTERNET** (LIBRARY) USE FOR ALL MAGAZINES FROM 1- THE LATEST ISSUE **FOR ALL COUNTRIES**
2. **LIB-STICK USB-CARD**: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY. € 120,00 FOR ALL COUNTRIES



The screenshot displays the Blaise Pascal Magazine website interface. The browser address bar shows the URL library.blaisepascalmagazine.eu. The page features a navigation bar with the magazine logo, a search bar containing "Alan Turing", and options for "Search in PDF" and "Dark mode". A sidebar on the left lists articles from Issue 66, including "From the editor", "Majorana, the new solution for QuantumBits?", "Video Effects and Animations creating video effect without hardly any coding", "Different Kind of Logic / Socrates - Humor", "FreePascal - Report - Part Two A new ReportingEngine for LAZARUS", and "Working with TACHart". The main content area shows a preview of the magazine cover for Issue 112, which features a large portrait of Blaise Pascal and a collage of smaller portraits. The cover text includes "BLAISE PASCAL MAGAZINE 112" and "Databases / CSS Styles / Progressive Web Apps / Android / IOS / Mac / Windows & Linux". Below the cover, a list of article titles is visible, such as "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", and "The new Lazarus Version 3.0 RC 2". At the bottom of the page, a URL is provided: <https://www.blaisepascalmagazine.eu/product-overview/>, followed by the text "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 Magazine Library

library.blaisepascalmagazine.eu

Other bookmarks

BLAISE PASCAL MAGAZINE

Issue 62 Open

Tester Search Search in PDF Dark mode Tester

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 kbm2W
Detlef Overbeek
Page: 21

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

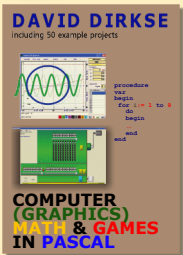
Issue 62, page 31
Introduction to video processing
Boian Mitov
Page: 31

NO ISSUE SELECTED
BLAISE PASCAL MAGAZINE

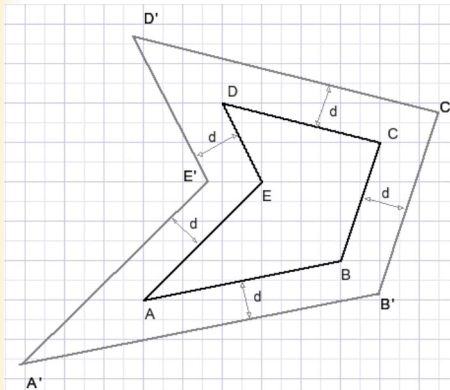
100 Load PDF...

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

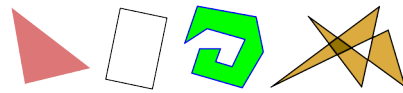


Ein Besucher meiner Website davdata.nl/math fragte: "Wie kann man ein **Polygon** erweitern?". Unten ist das **Polygon** ABCDE und die Erweiterung A'B'C'D'E' abgebildet

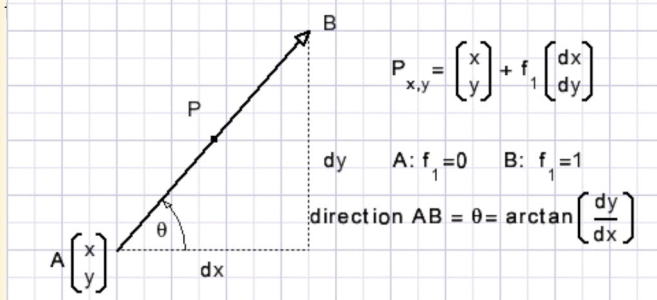


WIKIPEDIA

In der Geometrie ist ein **Polygon** eine ebene Figur, die aus Liniensegmenten besteht, die zu einer geschlossenen polygonalen Kette verbunden sind.



Alle Kanten werden um eine Strecke d nach außen verschoben. Die neuen Eckpunkte $A' \dots E'$ sind die Schnittpunkte der verschobenen Kanten $AB, BC, \dots EA$. Wie berechnet man eine solche Erweiterung? Die Kanten $AB, BC \dots$ sind **Vektoren**, sie sind durch ihre Länge und Richtung definiert. Vektor AB (siehe Bild unten): Jeder Wert von f_1 definiert einen Punkt auf AB . Punkt A : $f_1=0$, Punkt B : $f_1=1$. $f_1 > 1$ definiert einen Punkt auf der Verlängerung von AB , hinter B .

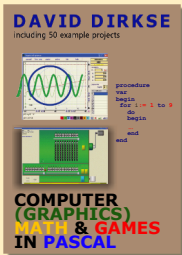


Das **Polygon** wird als eine Liste von **Vektoren** beschrieben.

```
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;
```

Die Richtung wird im Bogenmaß gemessen. Horizontal rechts ist die Richtung 0.





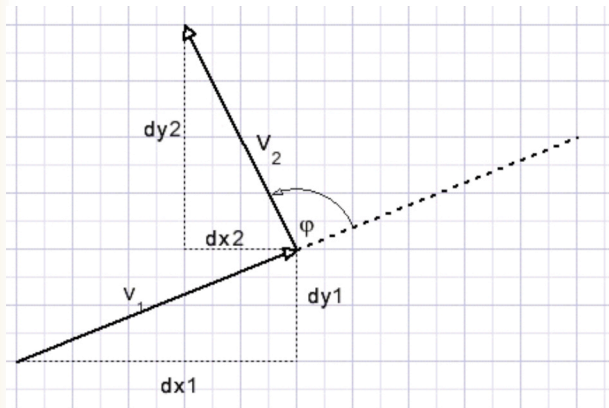
RICHTUNG EINES VEKTORS

```

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;

```

Richtungsunterschied (Winkel) zwischen Vektoren:
Das nächste Bild zeigt den Winkel zwischen den Vektoren v1 und v2.



```

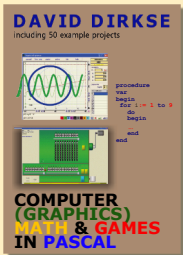
function V12angle(dir1,dir2 : double) : double;
// dir1,dir2 : direction in radians
// return angle between vectors v1,v2 in radians
// -pi.....+pi
begin
  result := dir2 - dir1;
  if result > pi then result := result-pi2
  else if result < -pi then result := result+pi2;
end;

```

Beim Verschieben einer Kante, z. B. AB, lautet das Problem: "links oder rechts"?
Wenn man sich um das **Polygon** bewegt und bei A beginnt, kann man entweder die Route ABCDE oder die Route AEDCB wählen. Bei der ersten Route müssen die Kanten zur Ausdehnung nach rechts verschoben werden, bei der zweiten Route müssen sie zur Ausdehnung nach links verschoben werden.

Die **Polygone** können im Uhrzeigersinn oder im Gegenuhrzeigersinn durchlaufen werden.
Das Summieren der Winkel (*Richtungsunterschiede*) liefert die Antwort.



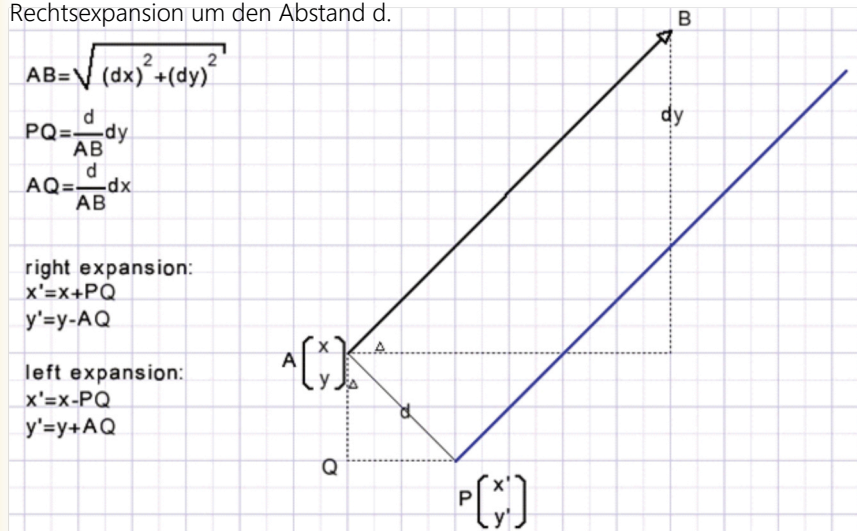


```

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;
    
```

Eine Summe von pi2 zeigt eine Linksdrehung an, -pi2 zeigt eine Rechtsdrehung an. Die CCW-Traversion erfordert eine "Rechts"-Expansion. Die CW-Traversion erfordert eine "linke" Ausdehnung.

Rechtsexpansion um den Abstand d.



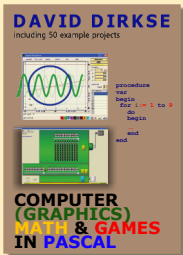
Die neuen (*erweiterten*) Punkte sind die Schnittpunkte der verschobenen (*blauen*) **Vektoren**.

```

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;

```





SCHNITTPUNKTES

Berechnung des Schnittpunktes S der Vektoren AB und CD

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

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

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

$$f_1(dx_1 \cdot dy_2 - dy_1 \cdot dx_2) = dy_2(x_2 - x_1) - dx_2(y_2 - y_1)$$

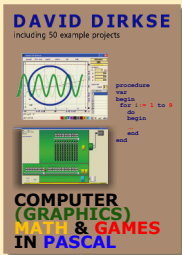
$\underbrace{\hspace{10em}}_d$
 $\underbrace{\hspace{10em}}_{vx}$
 $\underbrace{\hspace{10em}}_{vy}$

$$f_1 = \frac{[vx \cdot dy_2 - vy \cdot dx_2]}{d} \quad f_2 = \frac{[vx \cdot dy_1 - vy \cdot dx_1]}{d}$$

Koordinaten von S:

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





```

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

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

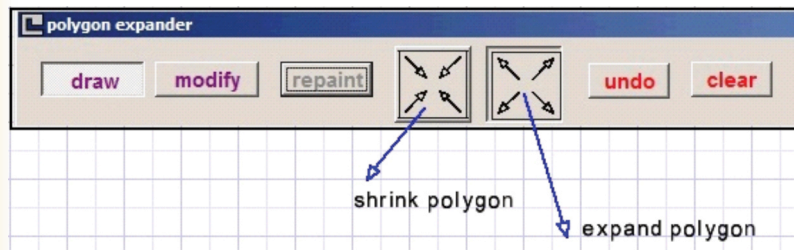
procedure vrsect(const v1,v2 : TVector);
// calculate intersection of vectors v1,v2
// line1 = (v1.x1,v1.y1) +f1*(v1.dx,v1.dy)
// line2 = (v2.x1,v2.y1) +f2*(v2.dx,v2.dy)
// return f1,f2,fvalid
var d,vx,vy : single;
begin
    d := v1.dx*v2.dy - v1.dy*v2.dx; // discriminant
    if d = 0 then begin
   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

```

DAS PROGRAMM



Zeichnen: Maus runter und bewegen, um den Vektor zu zeichnen.

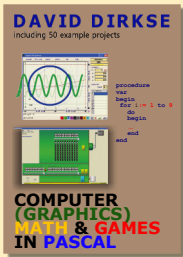
Modifizieren: Maus auf den Punkt, Maus gedrückt halten und bewegen, um den Vektor zu ändern.

Die anderen Buttons sind selbsterklärend.

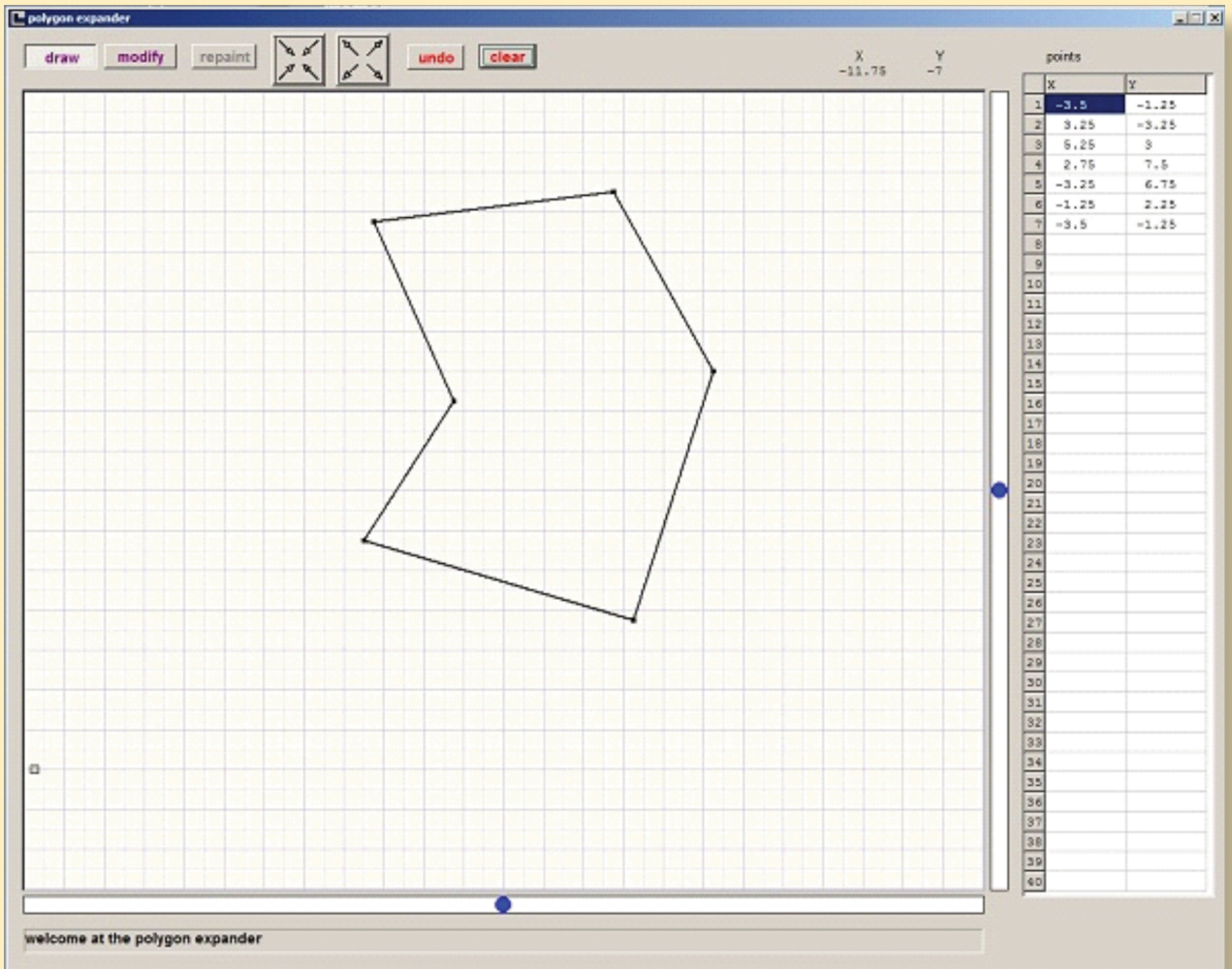
HINWEIS:

Der Cursor bewegt sich in 10-Pixel-Schritten, es sei denn, die SHIFT-Taste wird gedrückt gehalten.





David Dirkse's website:
davdata.nl/math

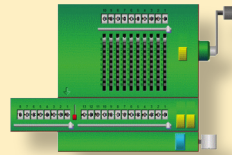


Damit ist die Beschreibung der Polygonerweiterung abgeschlossen.



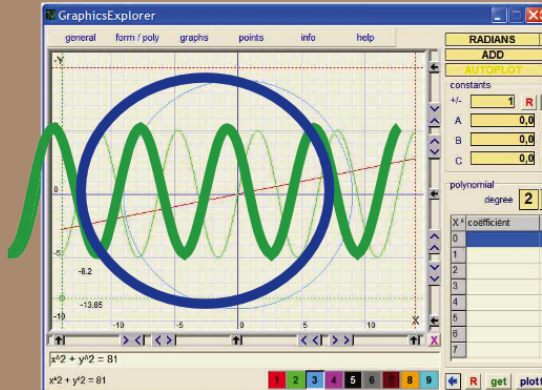
ADVERTISEMENT

David Dirkse's website: davdata.nl/math



DAVID DIRKSE

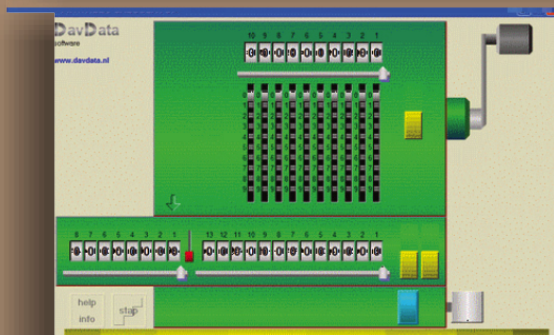
including 50 example projects



```

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

```



COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

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



Please see our products here:
<https://www.nexusdb.com>

20 years of



nexusdb

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



ÜBERBLICK ÜBER DIESEN ARTIKEL

- 1 Einführung
- 2 Codierung des Spiels
- 3 Übungen
- 4 Durch Verhaltensweisen dem Code zeigen "Was ist eine Schachfigur"
- 5 Auswahl von 3D-Objekten mit der Maus
- 6 Der Benutzer kann den Winkel und die Stärke des Schnipsens der Schachfigur wählen
- 7 Schnippe die Schachfigur!
- 8 Fazit und zukünftige Ideen



1 EINFÜHRUNG

Willkommen zum zweiten Teil des Artikels über die Erstellung eines einfachen 3D-Physikspiels mit **Castle Game Engine**. Castle Game Engine ist eine plattformübergreifende (*Desktop, Mobile, Konsolen*) 3D- und 2D-Spiele-Engine, die modernes **Pascal** verwendet. Sie ist **frei** und **quelloffen** und funktioniert sowohl mit **FPC** als auch mit **Delphi**.

Im ersten Teil haben wir gelernt, wie man den visuellen Editor benutzt und wir haben ein Schachbrett mit Schachfiguren entworfen. Dann haben wir die **Physik** benutzt, um die Schachfigur so zu werfen, dass sie mit anderen Schachfiguren zusammenstößt und diese umwirft.

Denkt daran, dass dies eine schlechte Art ist, Schach zu spielen. Aber es macht wirklich Spaß!

Wenn Sie den ersten Teil verpasst haben, können Sie an dieser Stelle noch einsteigen.

Sie können nach dem letzten **Heft 112 auf der Website des Blaise Pascal Magazins** suchen, in der **Internet LIBRARY Ihres Abonnements** <https://library.blaisepascalmagazine.eu/> suchen oder einfach die **Castle Game Engine** von

<https://castle-engine.io/> herunterladen und entweder das Schachbrett und die Schachfiguren selbst aufstellen oder verwenden Sie unser fertiges Beispielprojekt von <https://github.com/castle-engine/bad-chess/> im Unterverzeichnis **project/version_1_designed_in_editor**.

Diese Projektversion ist ein guter Ausgangspunkt für diesen Artikelteil.

Wir ermutigen Sie, diesem Artikel zu folgen und alle Schritte selbst auszuführen, um ein ähnliches Spielzeug zu erstellen. Wenn Sie einmal nicht weiterkommen, können Sie sich das fertige Projekt ansehen. Es befindet sich im Unterverzeichnis

project/version_2_with_code im gleichen Repository, <https://github.com/castle-engine/bad-chess/>.

"Es ist das endgültige Projekt, bei dem alles, was in diesem Artikel beschrieben wurde, fertig ist und funktioniert. Wenn Sie wirklich nur die schlechteste Version von Schach spielen wollen, können

Sie das fertig kompilierte Spiel (für Linux oder Windows) von

<https://castle-engine.itch.io/bad-chess> herunterladen. Viel Spaß damit!



2 CODIERUNG DES SPIELS

In diesem Teil geht es darum, zu lernen, wie man **Pascal-Code** verwendet, um Dinge in seinem Spiel zu realisieren.

Der Kern der **Castle Game Engine** ist nur ein Satz von **Pascal-Units**, die mit **FPC** und **Delphi** kompiliert werden können. Daher sind die Spiele, die wir erstellen, auch nur normale Pascal-Programme, die zufällig ein paar **Castle Game Engine-Einheiten** verwenden. Das bedeutet, dass Sie den Arbeitsablauf, den Sie bereits kennen und mögen, mit dem von Ihnen bevorzugten Pascal-Texteditor und Compiler verwenden können. Insbesondere unterstützen wir **Delphi, Lazarus, VS Code** oder jeden anderen benutzerdefinierten Editor (wie **Emacs**).

Wir haben eine spezielle Dokumentation mit einigen IDE-spezifischen Hinweisen auf https://castle-engine.io/manual_ide.php.

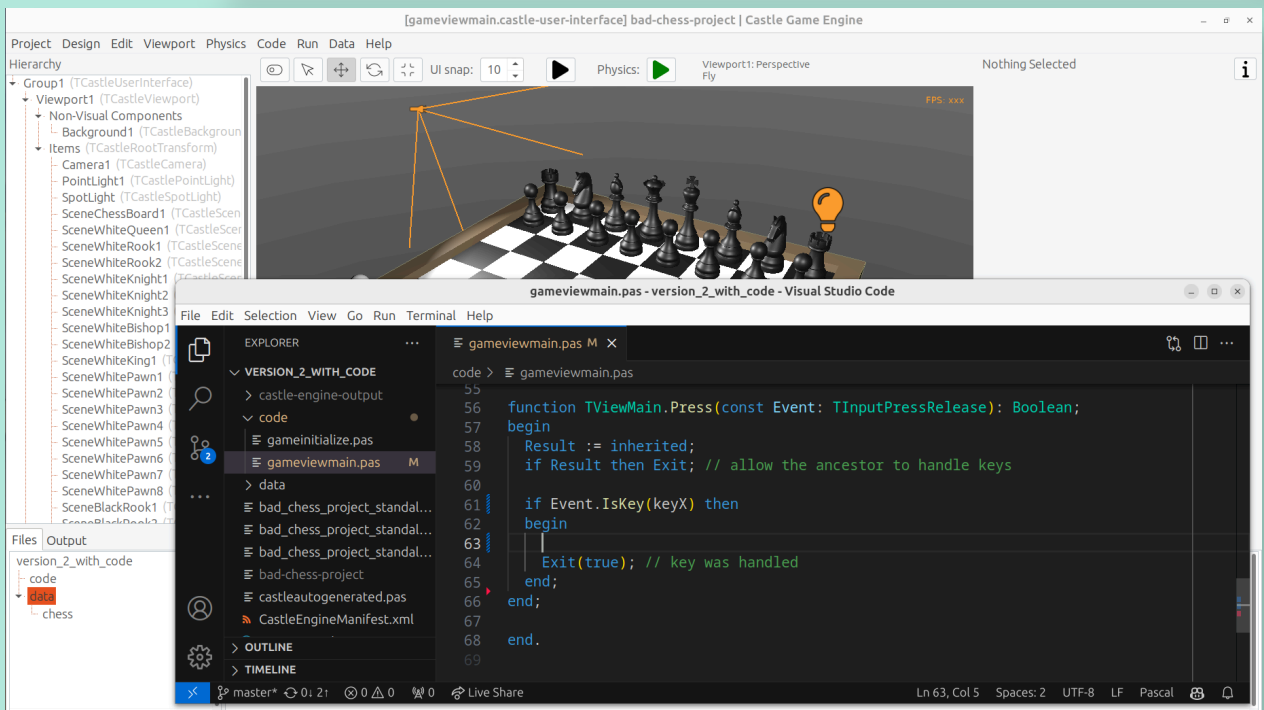




2 CODIERUNG DES SPIELS (FORTSETZUNG1)

Grundsätzlich öffnen Sie einfach im **Castle Game Engine Editor** das Panel "Preferences → Code Editor", stellen dort ein, welche **Pascal IDE** Sie verwenden, und alles sollte sofort funktionieren. Wenn Sie auf eine **Pascal-Datei** im **CGE-Editor** doppelklicken, wird sie in dem von Ihnen konfigurierten Texteditor geöffnet.

Speziell für **VS Code-Benutzer** enthält die Seite <https://castle-engine.io/vscode> Informationen, wie man **VS Code** mit dem **Castle Game Engine LSP Server** einrichtet, um eine großartige Code-Vervollständigung zu erhalten. Wir arbeiten gerade an einer speziellen **Castle Game Engine** Erweiterung für **VS Code**, die diese Integration noch einfacher machen wird.



HINWEIS: Obwohl der Schwerpunkt dieses Kapitels auf dem Schreiben von **Pascal-Code** liegt, hören wir nicht auf, den **Castle Game Engine Editor** zu verwenden. Es gibt ein paar Dinge, die Sie im Editor tun können, um das Design "freundlich" für die Code-Manipulation zu machen, und wir werden sie in diesem Artikel erkunden. Das Schreiben von **Pascal-Code** und die visuelle Bearbeitung des Designs gehen also Hand in Hand.



3 ÜBUNGEN

3.1. EINEN TASTENDRUCK VERARBEITEN, UM DIE POSITION EINES OBJEKTS ZU ÄNDERN

Fangen wir ganz einfach an. Erstes Ziel: Wenn der Benutzer eine Taste x drückt, wollen wir die Schachfigur des schwarzen Königs ein wenig höher stellen. Es ist ein einfacher Test, den wir durchführen können:

- Reagiere auf Benutzereingaben (*Tastendruck*).
- Als Antwort etwas Interessantes in der 3D-Welt tun (*eine Schachfigur bewegen*).





3.1. EINEN TASTENDRUCK VERARBEITEN, (FORTSETZUNG 1) UM DIE POSITION EINES OBJEKTS ZU ÄNDERN

Der meiste Code, den Sie in **Castle Game Engine** schreiben, wird in einer `Unit` platziert, die mit einer Ansicht verbunden ist. Eine Ansicht ist ein visuelles Design (*in data/gameviewmain.castle-user-interface*) und zugehöriger Code (*in code/gameviewmain.pas*).

Öffnen wir also die Datei `code/gameviewmain.pas` in Ihrer bevorzugten **Pascal-IDE**.

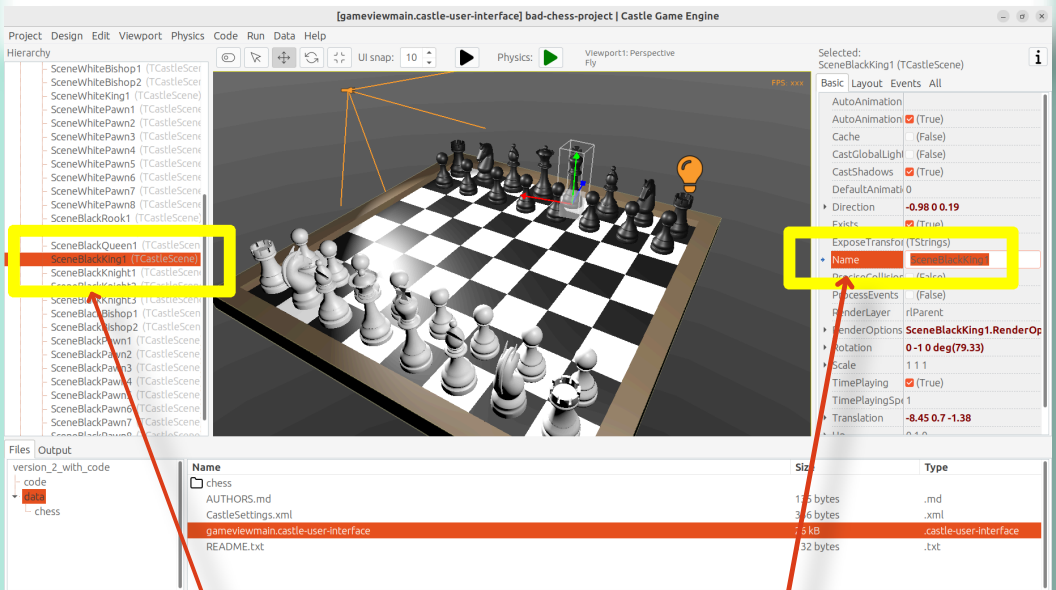
Im **Castle Game Engine Editor** können Sie einfach das untere "Files" Panel benutzen. Geben Sie das Unterverzeichnis `code` ein und doppelklicken Sie auf die Datei `gameviewmain.pas`. Alternativ können Sie auch einfach Ihre **Pascal-IDE** öffnen und von dort aus das **Pascal-Projekt** öffnen.

Die Projektdateien (wie `my_project.dproj` für **Delphi** oder `my_project.lpi` für **Lazarus**) wurden bereits für Sie generiert.

Halten Sie auch den visuellen Editor der **Castle Game Engine** offen, mit unserem **View Design data/gameviewmain.castle-user-interface**. Wir werden gelegentlich unser visuelles Design anpassen oder konsultieren, um sicherzustellen, dass es für unsere Codelogik nützlich ist.

Für den Anfang wollen wir den Namen der Komponente wissen, die den schwarzen König darstellt. Genauso wie Sie es bei der Gestaltung von **Lazarus**- und **Delphi**-Forms gesehen haben, hat jede Component einen Namen, der der Art und Weise entspricht, wie auf dieses Component vom Code aus zugegriffen werden kann.

Sie können die `ComponentNames` in **Castle Game Engine** bearbeiten, indem Sie entweder die Zeile Name im **Object Inspector** (*rechts*) oder den Namen in der Hierarchie (*links*) bearbeiten. Klicken Sie einfach auf den Komponentennamen in der Hierarchie oder drücken Sie **F2**, um in die Namensbearbeitung zu gelangen. Auf dem **Screenshot** unten können Sie sehen, dass der schwarze König `SceneBlackKing1` genannt wird. Man kann `Strg+C` verwenden, um diesen Namen in die Zwischenablage zu kopieren.





3.1. FORTSETZUNG (1)

HINWEIS: Für diese erste Code-Übung gehen wir davon aus, dass die Schachfigur (*SceneBlackKing1*) keine Physik-Komponenten hat. Wenn Sie *TCastleRigidBody*- oder *TCastleXxxCollider*-Komponenten als Verhaltensweisen von *SceneBlackKing1* hinzugefügt haben, entfernen Sie sie bitte zunächst. Wir werden sie in der nächsten Übung wieder einfügen.

Jetzt müssen wir die Variable mit genau demselben Namen in der Ansicht deklarieren. Sie wird automatisch initialisiert, um auf die Komponente zu zeigen, wenn wir die Ansicht starten. Dies geschieht im Abschnitt *published* der Klasse *TViewMain*. So sollte das Endergebnis aussehen:

```
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  
        ...
```

HINWEIS: Im Moment macht der Castle Game Engine Editor dies nicht automatisch für Sie. Das heißt, wir aktualisieren Ihre Pascal-Quellen nicht automatisch, um alle Komponenten zu deklarieren.

Wir planen, dies bald zu tun. Die Benutzererfahrung wird ein wenig anders sein als bei Delphi und Lazarus Formularen, weil die visuellen Designs des Spiels leicht Hunderte von Komponenten haben können, die nicht vom Code verwendet werden sollen, so daß die Synchronisation aller mit Pascal Code unnötiges Rauschen in der **Pascal** Unit erzeugen würde.

Wir werden stattdessen einen Button erstellen, der nur eine Teilmenge der entworfenen Komponenten für den Code freilegt.

Sobald Sie das veröffentlichte Feld deklariert haben, können wir vom Code aus auf *SceneBlackKing1* zugreifen, seine Eigenschaften abrufen und einstellen sowie seine Methoden an beliebiger Stelle aufrufen. Für diese Übung wollen wir die Eigenschaft *Translation* unserer Schachfigur ändern, die die Position des Objekts verändert.

Es handelt sich um eine Eigenschaft vom Typ *TVector3*. *TVector3* ist ein erweiterter Datensatz in der Castle Game Engine der einen 3D-Vektor darstellt - in diesem Fall eine Position, aber wir verwenden ihn auch in vielen anderen Fällen, z.B. um eine Richtung oder sogar eine RGB-Farbe darzustellen. Es gibt eine Reihe von nützlichen Dingen, die definiert wurden, um Ihnen die Arbeit mit *TVector3* zu erleichtern, insbesondere.





3.1. EINEN TASTENDRUCK VERARBEITEN, (FORTSETZUNG 2) UM DIE POSITION EINES OBJEKTS ZU ÄNDERN

Im Einzelnen:

- Die Funktion `Vector3(...)` gibt einen neuen `TVector3`-Wert mit den angegebenen Koordinaten zurück.
- Die arithmetischen Operatoren wie `+` funktionieren mit `TVector3`-Werten. Das bedeutet, dass wir Objekte leicht verschieben können, indem wir einen Code wie diesen schreiben:

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

Wo soll diese Anweisung stehen? Im Allgemeinen können Sie diesen Code überall in Ihrer Ansicht verwenden (*solange er nur ausgeführt wird, nachdem die Ansicht gestartet wurde*). In diesem Fall wollen wir auf das Drücken einer Taste `x` reagieren. Um das zu erreichen, bearbeiten wir in der Ansicht die Methode `TViewMain.Press`. Die leere Implementierung dieser Methode ist inklusive einiger hilfreicher Kommentare bereits vorhanden, so dass wir sie einfach mit unserem Code füllen können:"

```
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;
```

Erstellen und starten Sie das Spiel (z.B. durch Drücken von **F9** im **Castle Game Engine Editor**, oder in **Delphi**, oder in **Lazarus**) und drücken Sie **X** um zu sehen, wie es funktioniert.



3.2. SCHIEBEN SIE DIE SCHACHFIGUR MIT HILFE DER PHYSIK

Machen wir eine weitere Übung.

Wir wollen sicherstellen, dass wir mit Hilfe von Code eine Schachfigur mit Hilfe der **Physik** schieben (*schnippen, werfen*) können.

Die Schachfigur, die wir schieben und die Richtung, in die wir sie schieben, werden in dieser Übung fest kodiert sein. Aber wir werden die Gewissheit bekommen, dass wir die **Physik** mit **Pascal-Code** nutzen können.

Lassen Sie uns wieder den schwarzen König verwenden.

Dazu fügen Sie der entsprechenden Schachfigur die **Physik-Komponente** hinzu.

Wir haben im ersten Teil des Artikels beschrieben, wie man das macht.

Die kurze Zusammenfassung ist, dass man mit der rechten Maustaste auf die Komponente (*in diesem Fall **SceneBlackKing1***) klickt und aus dem Kontextmenü

"Add Behavior → Physics → Collider → Box (`TCastleBoxCollider`)" wählt.

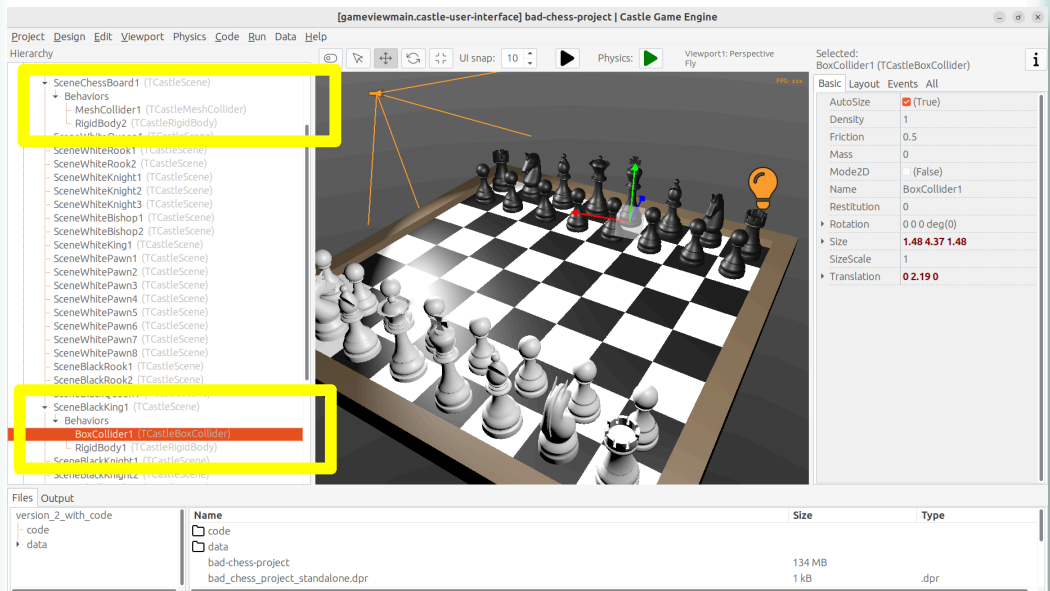
Stellen Sie sicher, dass Sie auch die Physik (*mit `TCastleMeshCollider`*) auf dem Schachbrett aktiv haben, sonst würde die Schachfigur aufgrund der Schwerkraft herunterfallen, sobald Sie das Spiel starten.





3.2. SCHIEBEN SIE DIE SCHACHFIGUR MIT HILFE DER PHYSIK (FORTSETZUNG 1)

So sollte es aussehen:



Um sie mit Hilfe der Physik zu schieben, wollen wir die Methode `ApplyImpulse` der Komponente `TCastleRigidBody` verwenden, die mit der Schachfigur verbunden ist.

- Sie können, wie unten gezeigt, die Komponente `TCastleRigidBody` mit der Methode `SceneBlackKing1.FindBehavior(TCastleRigidBody)` aufrufen. Alternativ können Sie auch die `RigidBody1:TCastleRigidBody` Referenz in dem public Abschnitt Ihrer Ansicht deklarieren und darauf zugreifen. Dieser Ansatz wird hier nicht gezeigt, weil die Verwendung des `FindBehavior` an dieser Stelle lehrreicher erscheint, d.h. Sie werden das `FindBehavior` in mehr Situationen nützlich finden.
- Die Methode `ApplyImpulse` benötigt zwei Parameter: die Richtung des Impulses (als `TVector3`; Länge dieses Vektors bestimmt die Stärke des Impulses) und die Position, von der der Impuls kommt (*am einfachsten ist es, hier einfach die Position der Schachfigur zu verwenden*).





3.2. SCHIEBEN SIE DIE SCHACHFIGUR MIT HILFE DER PHYSIK (FORTSETZUNG 2)

Letztendlich ist dies die modifizierte Version von `TViewMain.Press`, die Sie verwenden sollten:

```
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;
```

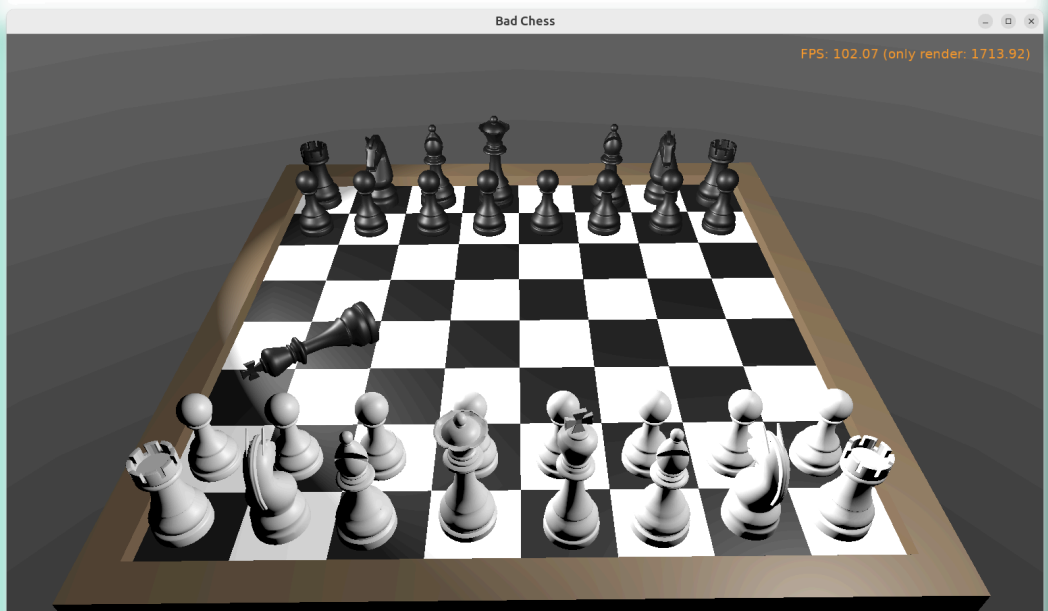
verwenden wir die Richtung `Vector3(0, 10, 0)`, was "nach oben, mit Stärke 10" bedeutet. Sie können mit verschiedenen Richtungen und Stärken experimentieren. Wenn wir die Schachfigur die Schachfigur horizontal schieben möchten, würden wir eine Richtung mit X- und/oder Z-Werten ungleich Null verwenden, und die Y-Achse bei Null lassen.

Fügen Sie der `uses`-Klausel auch die Einheit `CastleTransform` hinzu, um die Klasse `TCastleRigidBody` zu definieren.

Starten Sie wie üblich das Spiel und testen Sie es. Das Drücken von X sollte nun die Schachfigur nach oben befördern. Sie können X wiederholt drücken, auch wenn die Schachfigur bereits in der Luft ist.

Wie sie im Code sehen können - wir sichern uns nicht ab, also erlauben wir, ein Objekt zu schieben, das bereits fliegt.

Wir werden es in dieser Übung nicht behandeln, aber Sie könnten `MyBody.PhysicsRayCast` verwenden, um einen Strahl mit der Richtung `Vector3(0, -1, 0)` zu werfen und zu sehen, ob die Schachfigur bereits in der Luft ist.





4 CODE BEWUSST MACHEN "WAS IST EINE SCHACHFIGUR" VERHALTENSWEISEN VERWENDEN (FORTSETZUNG 1)

Um unsere gewünschte Logik zu implementieren, muss der Code irgendwie wissen, "was eine Schachfigur ist". Bisher ist unsere 3D-Welt eine Sammlung von `TCastleScene-Components`, aber das gibt uns nicht genug Informationen, um zwischen Schachfiguren und anderen Objekten (*wie einem Schachbrett*) zu unterscheiden.

Wir wollen etwas Verrücktes machen, aber wir wollen das Schachbrett nicht umdrehen! Zumindest nicht dieses Mal.

Um zu "markieren", dass die gegebene `TCastleScene`-Komponente eine Schachfigur ist, werden wir eine neue `class` namens `TChessPieceBehavior` erfinden, die von der `TCastleBehavior`-class abstammt.

Wir werden dann Instanzen dieser `class` an die `TCastleScene`-Komponente anhängen, die Schachfiguren darstellen. In der Zukunft kann diese `class` weitere Felder (*mit spezifischen Informationen zu dieser Schachfigur*) und Methoden haben.

Für den Anfang bedeutet allein das Vorhandensein einer `TCastleBehavior`-Instanz, die einer Szene zugeordnet ist, dass dies eine Schachfigur ist.

Wenn Sie mehr darüber wissen möchten, wie unsere Verhaltensweisen funktionieren, finden Sie unter <https://castle-engine.io/behaviors> eine Dokumentation und Beispiele.

Sie können auch ein neues Projekt aus der Vorlage "**3D FPS Game**" erstellen und sehen, wie die `TEnemy`-Class (*Nachkomme von* `TCastleBehavior`) definiert und verwendet wird. Die `Behaviors` sind ein sehr flexibles Konzept, um Informationen und Mechaniken zu Ihrer Welt hinzuzufügen und wir empfehlen, sie in vielen Situationen zu verwenden.

Unsere anfängliche `TChessPieceBehavior`-Definition ist wirklich nicht schwierig.

Es ist fast eine leere `Class`. Ich habe beschlossen, nur ein **boolesches** Feld hinzuzufügen, das angibt, ob die Schachfigur weiß oder schwarz ist:

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

Sie können sie am Anfang des Schnittstellenabschnitts der `Unit GameViewMain` deklarieren. Größere Verhaltensklassen sollten jedoch in eigenen `Units` untergebracht werden.

Wie fügt man die Verhaltensinstanzen den Szenen zu?

- Sie können dies visuell tun, indem Sie die `class TChessPieceBehavior` im Editor der `Castle Game Engine` registrieren. Dies ist eine sehr mächtige Methode, da sie es erlaubt, die Verhaltenseigenschaften visuell hinzuzufügen und zu konfigurieren. Siehe https://castle-engine.io/custom_components für eine Beschreibung, wie man dies verwendet.
- Oder Sie können es vom Code aus tun. In diesem Artikel habe ich mich für diesen Ansatz entschieden.

Dies ist etwas einfacher, wenn man das Verhalten 32 Mal an alle Schachfiguren anhängen muss, und es ist nicht notwendig den Anfangszustand des Verhaltens speziell zu konfigurieren.

32 Mal auf "Verhalten hinzufügen" zu klicken, wäre etwas mühsam und in unserem einfachen Fall auch unnötig (*in dieser Demo funktionieren wirklich alle Schachfiguren gleich*), also verwenden wir stattdessen Code, um die Schachfiguren einfach zu initialisieren.

Um ein Verhalten an unseren `SceneBlackKing1` anzuhängen, erstellen wir einfach eine Instanz von `TChessPieceBehavior` in der `Start`-Methode unserer Ansicht und fügen mit `SceneBlackKing1.AddBehavior` hinzuzufügen. Etwa so:





④ CODE BEWUSST MACHEN "WAS IST EINE SCHACHFIGUR" VERHALTENSWEISEN VERWENDEN (FORTSETZUNG 1)

Etwas so:

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

Aber das ist nicht gut genug für unsere Anwendung. Oben haben wir TChessPieceBehavior nur zu einer Schachfigur hinzugefügt. Wir wollen es zu allen 32 Schachfiguren hinzufügen.

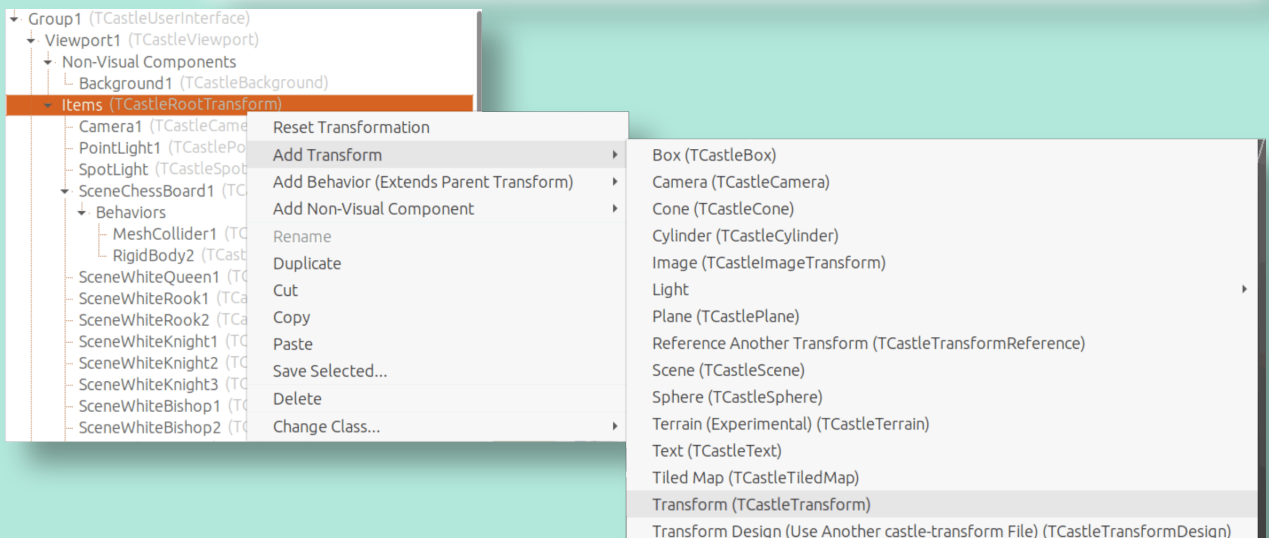
Wie ist das einfach zu bewerkstelligen? Wir müssen irgendwie über alle Schachfiguren iterieren. Und um das boolesche Feld Black zu setzen, sollten wir auch irgendwie wissen, ob es sich um eine schwarze oder weiße Figur handelt. Es gibt mehrere Lösungen:

- ① Wir könnten annehmen, dass alle Schachfiguren Namen wie SceneWhiteXxx oder SceneBlackXxx haben. Dann können wir über Viewport1.Items Kinder iterieren, und prüfen, ob ihr Name mit dem angegebenen Präfix beginnt.
- ② Oder wir könnten auf den Tag-Wert der Szenen schauen und eine Konvention haben, z.B. dass Tag = 1 für eine schwarze Schachfigur steht, Tag = 2 für eine weiße Schachfigur, und andere Tags (Tag = 0 ist der Standardwert) bedeutet, dass es sich nicht um eine Schachfigur handelt.
- ③ Wir könnten auch zusätzliche Transformationskomponenten einführen, die schwarze Schachfiguren von weißen Schachfiguren und von anderen Dingen (wie ein Schachbrett) trennen.

Ich habe mich für den letzteren Ansatz entschieden, da die Einführung von "zusätzlichen TCastleTransform-Komponenten zur Gruppierung bestehender Komponenten" in vielen anderen Situationen ein mächtiger Mechanismus ist.

So können Sie z. B. eine bestimmte Gruppe leicht ein- oder ausblenden (mit der Eigenschaft TCastleTransform.Exists).

Klicken Sie dazu mit der rechten Maustaste auf Viewport1.Items und wählen Sie im Kontextmenü "Add Transform → Transform (TCastleTransform)".



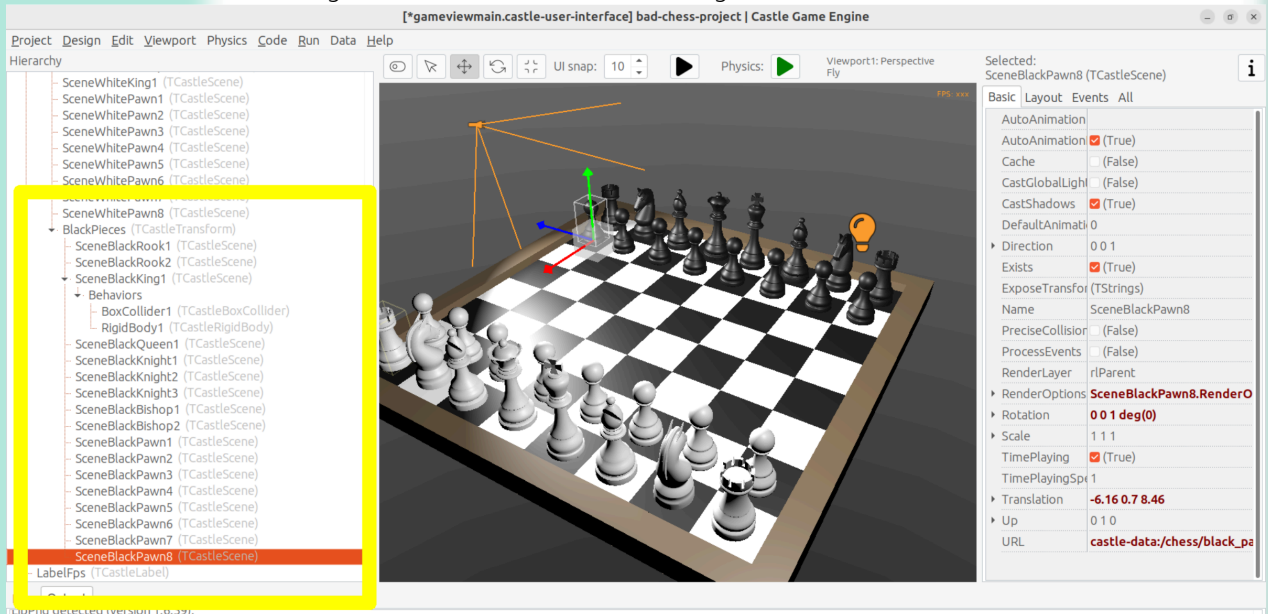


4 CODE BEWUSST MACHEN "WAS IST EINE SCHACHFIGUR" VERHALTENSWEISEN VERWENDEN (FORTSETZUNG 2)

Nennen Sie diese neue Component BlackPieces.

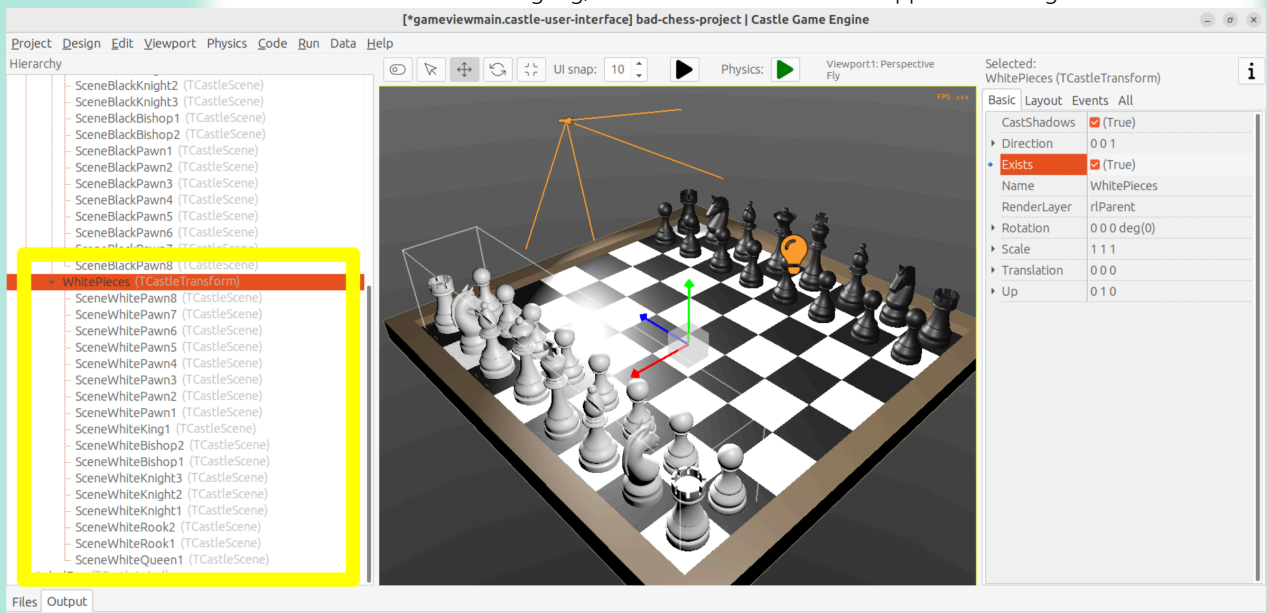
Ziehen Sie dann alle schwarzen Schachfiguren (SceneBlackXxx-Components) per Drag-and-Drop in die Editor-Hierarchie, damit sie Kinder von BlackPieces werden. Sie können alle 16 Szenen, die schwarze Figuren in der Hierarchie darstellen, einfach auswählen, indem Sie die Umschalttaste gedrückt halten und sie dann alle auf einmal in BlackPieces ziehen.

Das Endergebnis sollte in der Hierarchie wie folgt aussehen:



Machen Sie sich keine Sorgen, dass nur SceneBlackKing1 die Physik-Component hat. Wir werden die Physik-Component auch bald mit Code einstellen.

Wiederholen Sie nun den Vorgang, um eine WhitePieces-Gruppe hinzuzufügen.





4 CODE BEWUSST MACHEN "WAS IST EINE SCHACHFIGUR" VERHALTENSWEISEN VERWENDEN (FORTSETZUNG 3)

Diese Vorbereitung im Editor erleichtert uns die Arbeit am Code. Fügen Sie dem veröffentlichten Abschnitt von `TViewMain` die Deklaration der Felder `BlackPieces` und `WhitePieces` vom Typ `TCastleTransform` hinzu:

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

Iterieren Sie nun über die Gruppen der 2 Schachfiguren in der Methode `start`:

```
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;
```

Es scheint ratsam, an dieser Stelle eine grundlegende "Plausibilitätsprüfung" vorzunehmen. Lassen Sie uns die Anzahl der Schachfiguren protokollieren, die jede Seite hat. Fügen Sie den folgenden Code und das Ende der `start`-Methode hinzu:

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

Um `writelnLog` verfügbar zu machen, fügen Sie die Einheit `CastleLog` zur `uses`-Klausel hinzu. Wenn Sie nun das Spiel starten, sollten Sie ein Protokoll sehen

Configured 16 black and 16 white chess pieces

Bei meinem ersten Durchlauf habe ich tatsächlich zufällig gesehen, dass ich 17 Schachfiguren auf jeder Seite habe.

Ich habe versehentlich 3 statt 2 Springer hinzugefügt (*ein Springer stand genau an der gleichen Stelle wie ein anderer, so dass es nicht offensichtlich war*).

Dank dieses Protokolls habe ich die überzähligen Springerfiguren entfernt.

Das Aufspüren solcher Fehler ist genau der Grund, warum wir Protokolle hinzufügen und testen - ich möchte Sie also ermutigen, das auch zu tun.

Wenn wir schon dabei sind, können wir bei dieser Gelegenheit auch sicherstellen, dass alle Schachfiguren über Physik-Komponenten verfügen (`TCastleRigidBody` und `TCastleBoxCollider`). Sie müssen sie also nicht alle manuell hinzufügen.

Dies ist ein vernünftiger Ansatz, wenn die Components nicht manuell pro Schachfigur angepasst werden müssen.





④ CODE BEWUSST MACHEN "WAS IST EINE SCHACHFIGUR" VERHALTENSWEISEN VERWENDEN (FORTSETZUNG 4)

Erweitern Sie dazu unsere Methode `ConfigureChessPiece`:

```
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;
```

Wie Sie oben sehen, ist dieser Ansatz ziemlich direkt: Wenn Sie die erforderliche Komponente nicht haben, fügen Sie sie einfach hinzu. Wir machen uns nicht die Mühe, irgendeine Eigenschaft der neuen `TCastleRigidBody`- und `TCastleBoxCollider`-Instanzen zu konfigurieren, da ihre Standardwerte für unseren Zweck gut geeignet sind.

Dies alles war eine gute "Vorarbeit" für den restlichen Teil des Artikels. Funktionell hat sich in unserem Spiel eigentlich nichts geändert, Sie sollten es ausführen und sehen, dass sich... nichts geändert hat. Alle 32 Schachfiguren stehen einfach still am Anfang.



⑤ AUSWAHL VON 3D-OBJEKTEN MIT DER MAUS

5.1. Markieren und Auswählen der Schachfigur mit der Maus

Um die echte Interaktion zu implementieren, wollen wir dem Benutzer die Möglichkeit geben, mit der Maus zu wählen, welche Schachfigur er schnippen möchte. **Castle Game Engine** bietet eine fertige Funktion, die Ihnen mitteilt, was durch die aktuelle Mausposition (oder die letzte *Berührung auf dem Handy*) angezeigt wird. Dies ist die Funktion `TCastleViewport.TransformUnderMouse`. Um zu beginnen, stelle sicher, dass Sie die `Viewport`-Instanz in der `public` Sektion der Klasse `TViewMain` deklarieren, etwa so:

```
MainViewport: TCastleViewport;
```

Passen Sie den Namen Ihres **Viewports** in den Entwurf ein. Fügen Sie die `Unit CastleViewport` zur `uses`-Klausel hinzu, um den Typ `TCastleViewport` bekannt zu machen. Verwenden wir es, um die aktuelle Schachfigur an der Mausposition zu markieren.

Wir können einfach den Wert `MainViewport.TransformUnderMouse` in jedem `Update`-Aufruf überprüfen. Passen Sie den Namen Ihres Viewports in den Entwurf ein.

Fügen Sie die Einheit `CastleViewport` zur `uses`-Klausel hinzu, um den Typ `TCastleViewport` bekannt zu machen.

HINWEIS: Alternativ könnten wir `MainViewport.TransformUnderMouse` bei jedem `Motion`-Aufruf prüfen, der nur erfolgt, wenn sich die Maus- (oder Touch-) Position ändert. Aber es in `Update` zu tun, ist ein bisschen besser: da wir `Physik` verwenden, können sich einige Schachfiguren immer noch aufgrund der `Physik` bewegen, so dass sich die Schachfigur unter der Maus ändern kann, auch wenn sich die Mausposition nicht ändert.

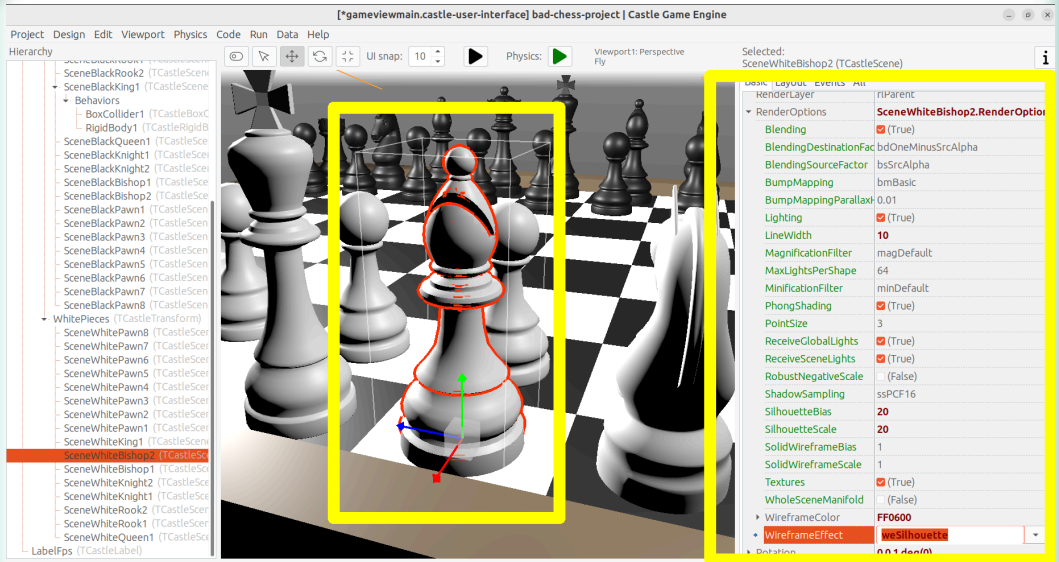
Um anzuzeigen, verwenden wir einen fertigen `Highlight` Effekt, der für jede `TCastleScene` verfügbar ist und der aktiviert werden kann, indem `MyScene.RenderOptions.WireframeEffect` auf etwas anderes als `weNormal` gesetzt wird. Dies ist die einfachste Art, das `Highlight` zu zeigen (andere Möglichkeiten werden in einem späteren Abschnitt diskutiert).

Bevor wir uns in den Code stürzen, empfehle ich, mit den perfekten Einstellungen von `RenderOptions` für `Highlight` im Editor zu experimentieren. Bearbeiten Sie einfach eine beliebige Schachfigur, bis sie ein hübsches `Highlight` haben, und merken Sie sich die gewählten Optionen. Die nützlichsten Eigenschaften zum Anpassen sind `WireframeEffect`, `WireframeColor`, `LineWidth`, `SilhouetteBias`, `SilhouetteScale`. Sie können sie auf der nächsten Seite hervorgehoben sehen - der Editor zeigt Eigenschaften, die keine Standardwerte haben, in fetter Schrift an.





5.1. MARKIEREN UND AUSWÄHLEN DER SCHACHFIGUR MIT DER MAUS
(FORTSETZUNG 1)



Ich habe mich entschieden, die aktuell (*an der Mausposition*) hervorgehobene Schachfigur mit einem hellblauen Drahtgitter darzustellen. Diese Schachfigur wird auch als Wert des privaten Feldes `ChessPieceHover` festgelegt. Sobald der Benutzer mit der Maus klickt (*wir können dies in Press erkennen*), wird die Schachfigur als ausgewählt betrachtet und erhält eine gelbe Markierung. Diese Schachfigur wird als `ChessPieceSelected`-Wert festgelegt. Das Merken der Werte `ChessPieceHover` und `ChessPieceSelected` ist für einige Dinge nützlich. Zum einen können wir den Effekt später deaktivieren (*wenn die Figur nicht mehr hervorgehoben oder ausgewählt ist*). Und es wird ermöglichen, die `ChessPieceSelected` in den nächsten Abschnitten zu flackern.

Wir könnten sie als Referenzen auf `TCastleScene` oder `TChessPieceBehavior` speichern. Das heißt, wir könnten deklarieren:
Entweder `ChessPieceHover, ChessPieceSelected: TChessPieceBehavior;...`
...oder `ChessPieceHover, ChessPieceSelected: TCastleScene;`
Beide Deklarationen wären für unsere Anwendung gut. Das heißt, wir müssen uns für das eine oder das andere entscheiden, da es einen etwas anderen Code bedeutet, aber die Unterschiede sind wirklich gering. Letztendlich können wir immer eine `TChessPieceBehavior`-Instanz von einer entsprechenden `TCastleScene` erhalten (*wenn wir wissen, dass es sich um eine Schachfigur handelt*) und wir können eine `TCastleScene` von einem `TChessPieceBehavior` erhalten. Um `TChessPieceBehavior` von der entsprechenden `TCastleScene` zu erhalten, würden wir folgendes tun:

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





5.1. MARKIEREN UND AUSWÄHLEN DER SCHACHFIGUR MIT DER MAUS (FORTSETZUNG 2)

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

Ich habe beschlossen, sie als TChessPieceBehavior zu deklarieren. Wenn Sie meinem Ansatz genau folgen wollen, fügen Sie dies dem privaten Abschnitt der Klasse TViewMain hinzu:

```
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);
```

Dann fügen Sie die Unit CastleColors zur uses-Klausel hinzu (der Schnittstelle oder der Implementierung der Unit GameViewMain, spielt in diesem Fall keine Rolle), um das HexToColorRGB-Dienstprogramm zu definieren.

Schließlich ist dies der Code der neuen Update-, Press- und ConfigureEffect-Hilfsmethoden:

The screenshot shows the Itch.io page for the game 'Bad Chess: 3D Physics Fun'. The page includes a title, a description of the game, controls, and a download button. The description states that the game is part of the 'Castle Game Engine' series and is published by 'Blaise Pascal Magazine'. The controls section lists: '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.', and 'Flick the chess piece by pressing Enter.' The download button is labeled 'Download Now' and has the text 'Name your own price' next to it. The page also features a 'More information' link and a 'PLAY THE GAME: ALL READY AND PREPARED' section with the URL 'https://castle-engine.itch.io/bad-chess'. There are three images showing the game in progress: a top-down view of the chessboard, a side view of the chessboard, and a 3D perspective view of the chessboard with a piece being flicked.





5.1. MARKIEREN UND AUSWÄHLEN DER SCHACHFIGUR MIT DER MAUS (FORTSETZUNG 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

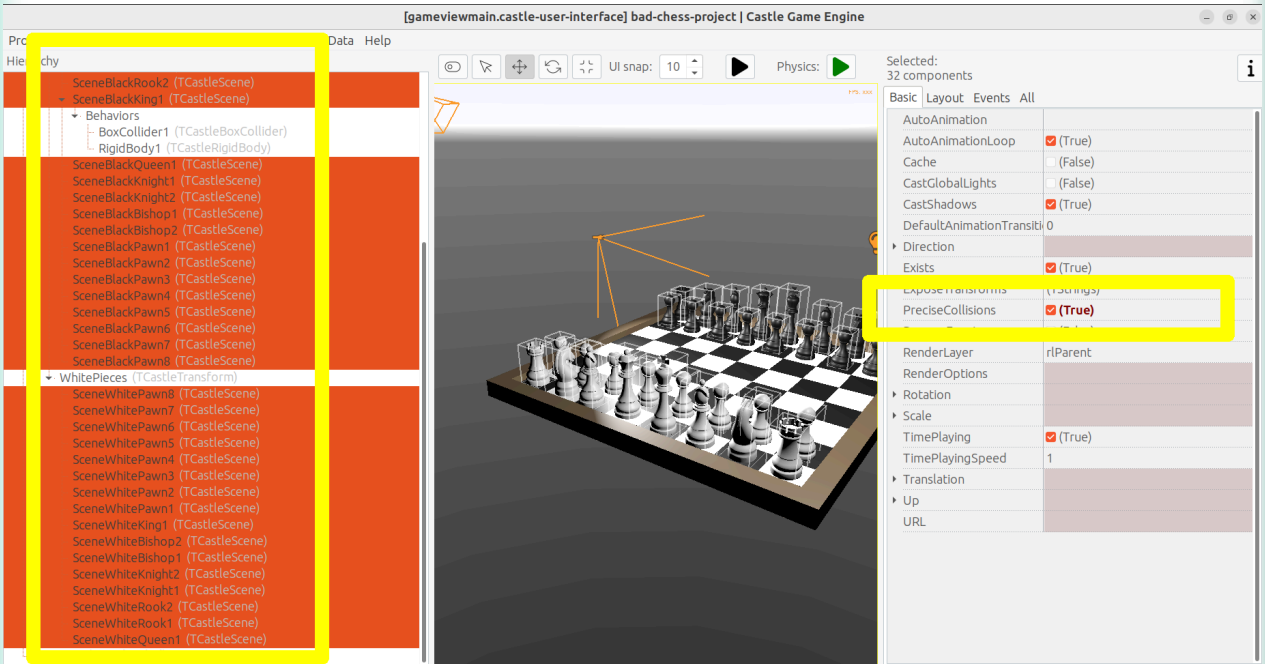
  if Event.IsMouseButton(buttonLeft) then
  begin
    OldSelected := ChessPieceSelected;
    if (ChessPieceHover <> nil) and
      (ChessPieceHover <> ChessPieceSelected) then
    begin
      ChessPieceSelected := ChessPieceHover;
      ConfigureEffect(OldSelected);
      ConfigureEffect(ChessPieceSelected);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



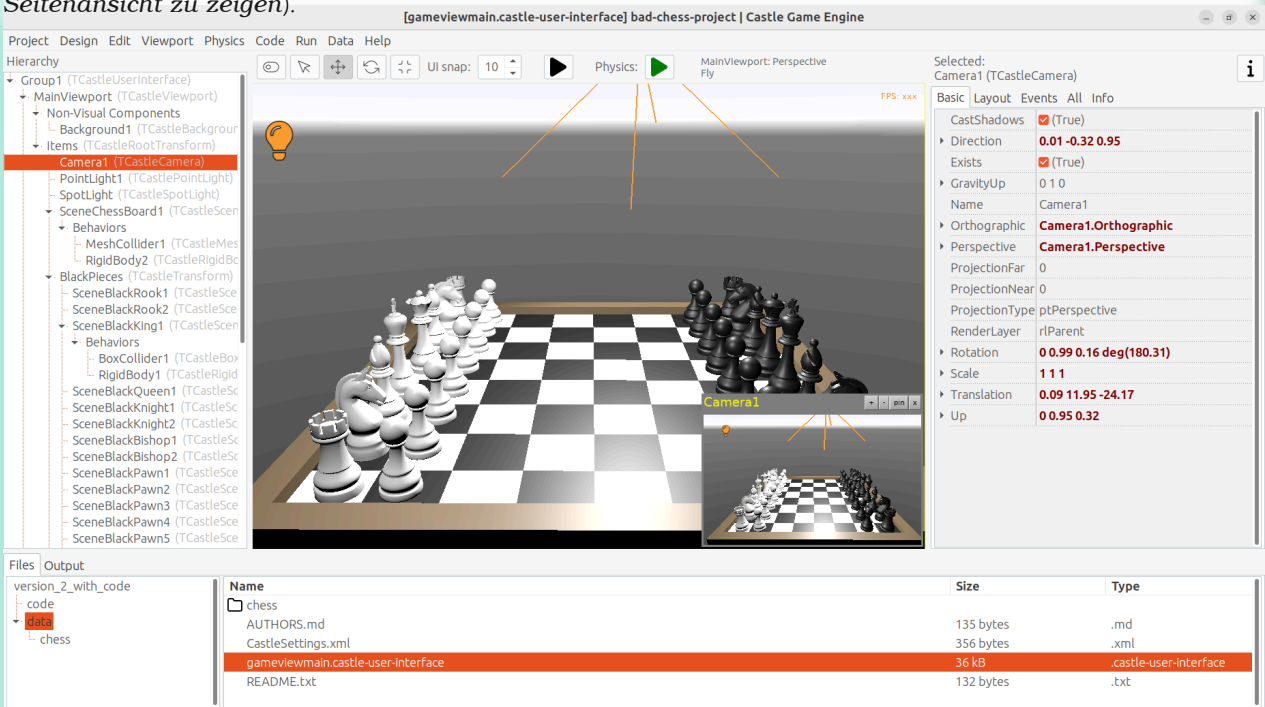


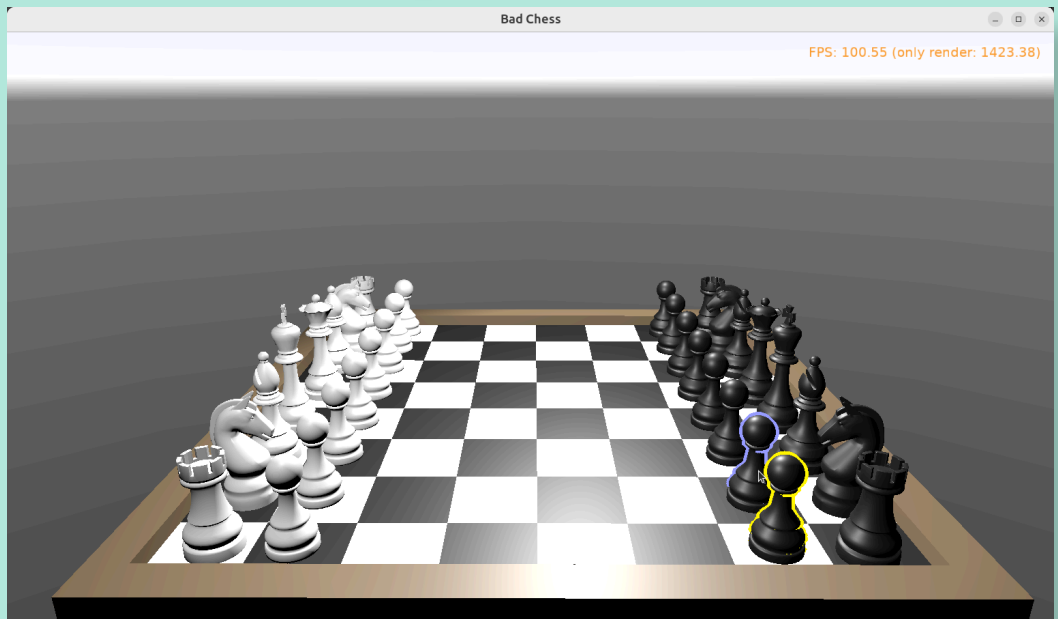
5.1. MARKIEREN UND AUSWÄHLEN DER SCHACHFIGUR MIT DER MAUS (FORTSETZUNG 4)

Wie immer sollten Sie den Code kompilieren und ausführen, um sicherzugehen, dass er gut funktioniert! Sie werden feststellen, dass `MainViewport.TransformUnderMouse` erkennt, was sich unter der Maus befindet, aber jede Schachfigur als Kasten behandelt. Die Erkennung ist also sichtbar ungenau. Um dies zu beheben, setzen Sie `PreciseCollisions` für alle Schachfiguren auf true. Sie können dies ganz einfach tun, indem Sie alle Schachfiguren im Editor mit Shift oder Strg auswählen und dann `PreciseCollisions` im Objektinspektor einschalten.



Ich beschloss, die Kamera an dieser Stelle ebenfalls zu bewegen (um beide Seiten, schwarz und weiß, aus der Seitenansicht zu zeigen).





5.2. NEBENBEMERKUNG: ANDERE MÖGLICHKEITEN, EIN HIGHLIGHT ZU ZEIGEN

Es gibt andere Möglichkeiten, die hervorgehobene (oder ausgewählte) Schachfigur anzuzeigen:

- **Dynamische Änderung der Materialfarbe.** Dies geschieht durch Zugriff auf eine Instanz von `TPhysicalMaterialNode` innerhalb der Knoten der Szene (`TCastleScene.RootNode`) und Änderung der `TPhysicalMaterialNode.BaseColor`. Siehe z.B. das Engine-Beispiel `examples/viewport_and_scenes/collisions/`, das dies verwendet.
- **Dynamisches Hinzufügen/Entfernen eines Shader-Effekts:** Dies bedeutet das Hinzufügen von `TEffectNode` und `TEffectPartNode` Knoten zur Szene und die Implementierung des Effekts mit GLSL (*OpenGL Shading Language*). Siehe z.B. das Engine-Beispiel `examples/viewport_and_scenes/shader_effects/`, das dies demonstriert.
- **Hinzufügen einer zusätzlichen Box,** die das ausgewählte Objekt umgibt: Der **CGE**-Editor selbst verwendet diese Technik, um markierte / ausgewählte 3D-Objekte anzuzeigen. Verwenden Sie die `TDebugTransformBox-Class`, um dies einfach zu implementieren. Wenn Sie neugierig geworden sind, werden Ihnen die obigen Informationen und Beispiele hoffentlich die richtige Richtung weisen.

5.3. NEBENBEMERKUNG: SCHATTEN

Ich habe beschlossen, an dieser Stelle Schatten zu aktivieren. Setzen Sie einfach `Shadows` für die Hauptlichtquelle auf `true`. Außerdem setzen Sie `RenderOptions.WholeSceneManifold` bei den Schachfiguren auf `true`.

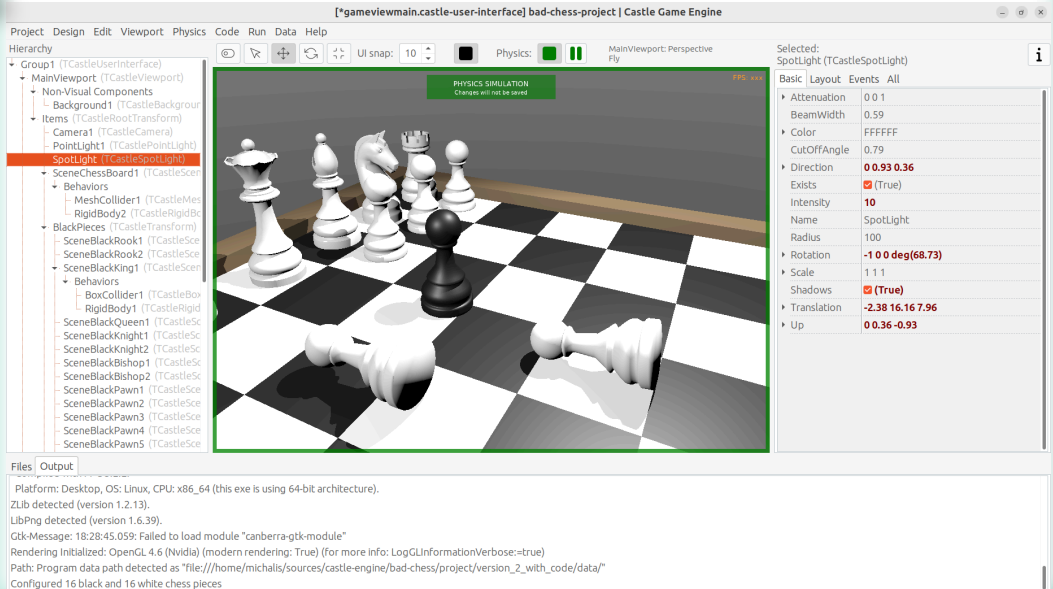
Dadurch sollte alles schöne Schatten werfen. Die Schatten sind dynamisch, was bedeutet, dass sie sich richtig verändern, wenn wir die Schachfiguren bewegen.

Siehe https://castle-engine.io/shadow_volumes für weitere Informationen über Schatten in **Castle Game Engine**.





5.3. NEBENBEMERKUNG: SCHATTEN (FORTSETZUNG)



6 LÄSST DEN BENUTZER DEN WINKEL UND DIE STÄRKE DES SCHNIPSENS DER SCHACHFIGUR WÄHLEN

Sobald der Benutzer eine Schachfigur ausgewählt hat, wollen wir die Möglichkeit schaffen, die Richtung und die Stärke zu konfigurieren mit der das gewählte Objekt geschnipst werden soll. Wir wissen bereits, dass das "Schnippen" der Schachfigur technisch gesehen bedeutet, dass eine physikalische Kraft auf den starren Körper der gewählten Schachfigur ausgeübt wird. Wir haben fast alles, was wir brauchen, aber wir müssen dem Benutzer die Möglichkeit geben, die Richtung und Stärke dieser Kraft zu wählen.



6.1. ENTWURF EINES 3D-PFEILS

Um die gewünschte Kraft zu visualisieren, werden wir ein einfaches 3D-Pfeilmodell verwenden, das entsprechend gedreht und skaliert wird.

Obwohl wir ein solches Modell auch in Blender oder einer anderen 3D-Authoring-Software entwerfen könnten, ist es in diesem Fall am einfachsten, es komplett im Editor der Castle Game Engine zu erstellen. Der Pfeil ist eine Komposition aus zwei einfachen Formen: Kegel (für die Pfeilspitze) und ein Zylinder.

Außerdem entwerfen wir den Pfeil unabhängig, als separates Design.

Der neue Entwurf wird eine Hierarchie von Komponenten enthalten, wobei die Wurzel TCastleTransform ist. Wir speichern es als Datei force_gizmo.castle-transform im Unterverzeichnis project data.

Dann fügen wir sie dem Hauptentwurf (gameviewmain.castle-user-interface) hinzu und schalten die Existenz, Drehung und Skalierung der visualisierten Kraft um.

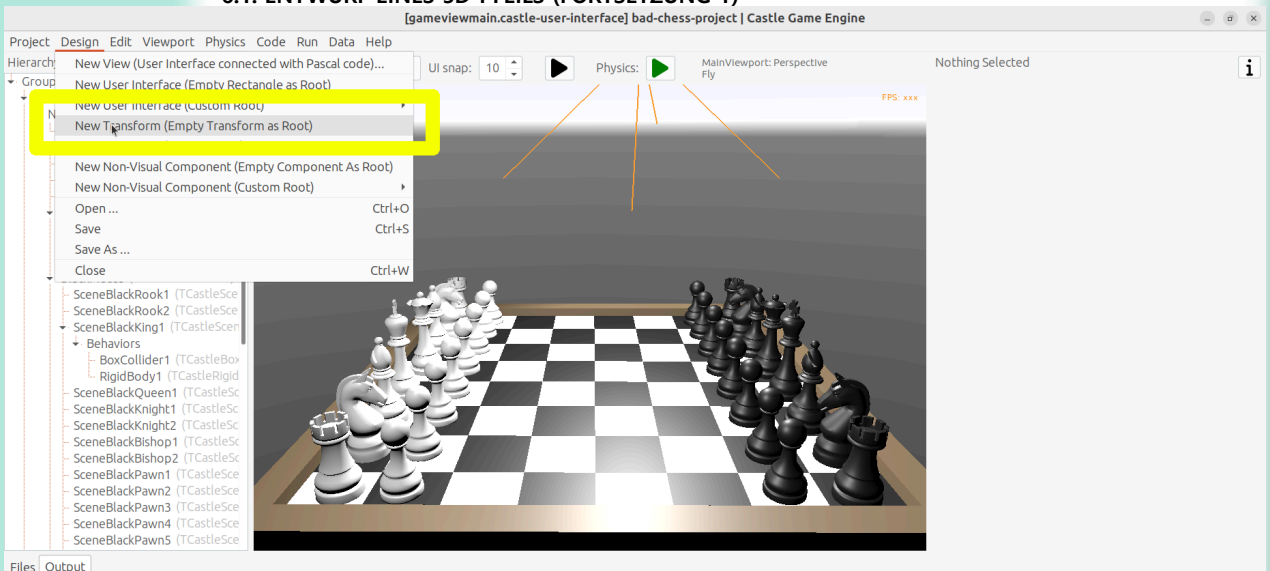
Die Verwendung einer separaten Entwurfsdatei für den 3D-Pfeil ist in diesem Fall zwar nicht unbedingt erforderlich, aber eine sehr leistungsfähige Technik. Wenn etwas als separate Designdatei gespeichert wird, können Sie es frei wiederverwenden und viele Male instanzieren (zur Designzeit oder durch dynamisches Spawnen während der Spiellaufzeit). Auf diese Weise können Sie z. B. Kreaturen in Ihrem Spiel haben: 3D-Objekte, die eine gemeinsame Logik haben und bei Bedarf gespawnt werden können.

Um mit der Gestaltung des Pfeils zu beginnen, wählen Sie im Editor den Menüpunkt "Design → New Transform (Empty Transform as Root)".





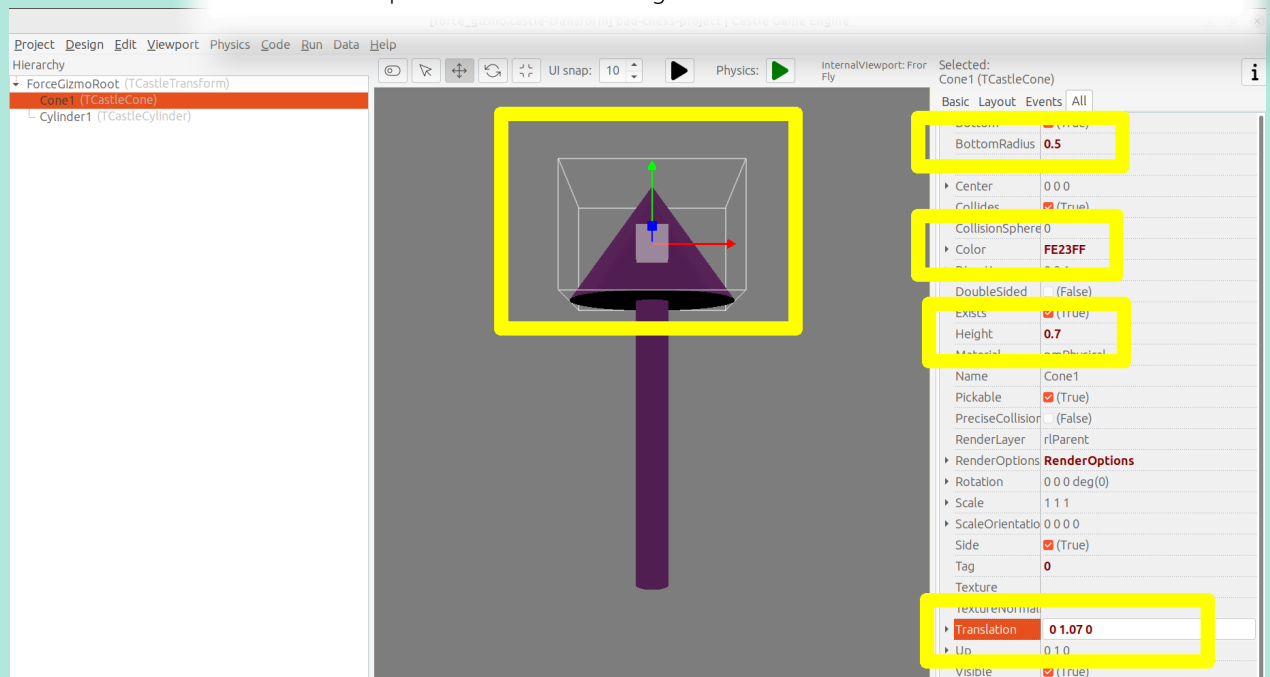
6.1. ENTWURF EINES 3D-PFEILS (FORTSETZUNG 1)



Darunter fügen Sie zwei Komponenten hinzu: `TCastleCylinder` und `TCastleCone`. Passen Sie ihre Höhe, Radius (*auf Zylinder*), *BottomRadius* (*auf Kegel*) und Übersetzung, um einen schönen 3D-Pfeil zu bilden.

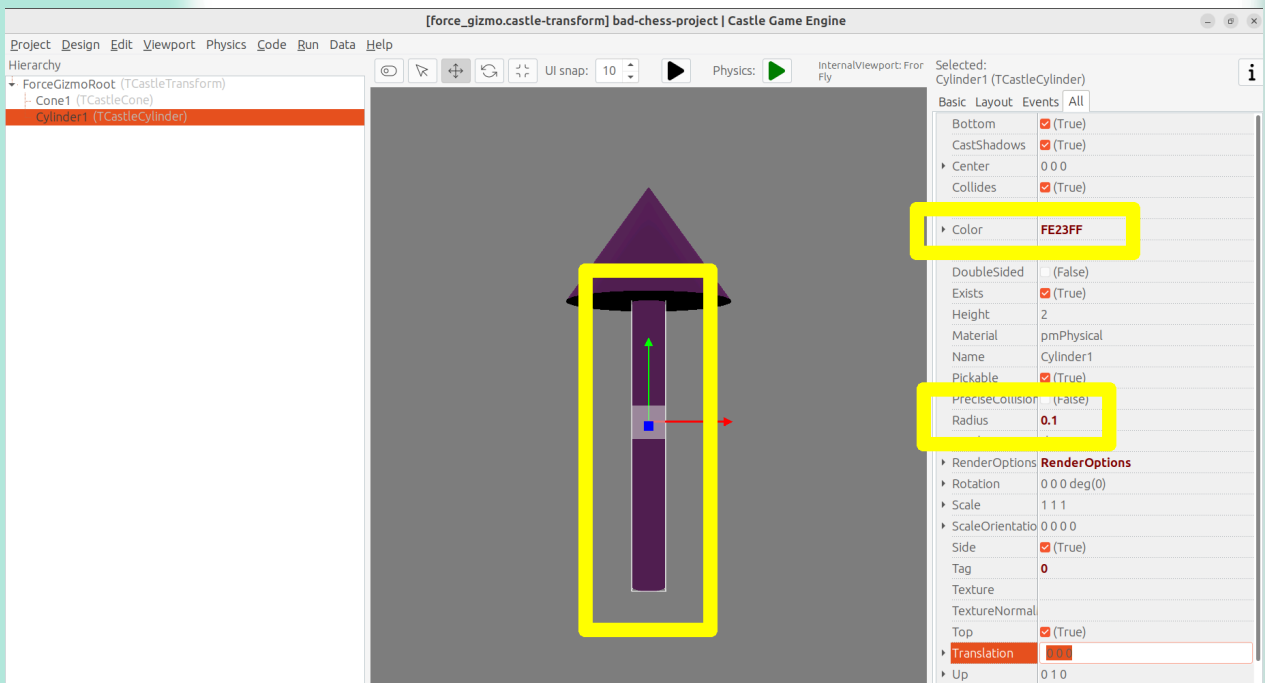
Stellen Sie die Farbe auf einen anderen Wert als den Standardwert ein, um die Sache hübscher zu machen. Denken Sie daran, dass der Pfeil später von den Lichtern beleuchtet wird, die wir im Hauptdesign (`gameviewmain.castle-user-interface`) eingerichtet haben, daher wird er wahrscheinlich heller sein als das, was Sie jetzt sehen.

Sie können die Werte, die ich gewählt habe, auf den Screenshots unten verfolgen, aber das sind wirklich nur Beispiele. Sie können Ihren eigenen 3D-Pfeil nach Belieben erstellen.





6.1. ENTWURF EINES 3D-PFEILS (FORTSETZUNG 2)



Jetzt kommt ein etwas schwieriger Teil.

Wir wollen einen Pfeil haben, der sich leicht um eine Dummy-Box drehen kann (*im eigentlichen Spiel wird er sich um eine Schachfigur drehen*). Idealerweise sollte ein Pfeil auch leicht skalierbar sein, um die Kraftstärke zu visualisieren. Ich verwende die Worte "leicht", um zu betonen, dass wir den Pfeil nicht nur im Editor drehen wollen, sondern dass wir dem Benutzer auch erlauben müssen, ihn während des Spiels zu drehen. Die Drehung und die Skalierung, die für uns interessant sind, müssen also sehr einfach über den Code zu erreichen und einzustellen sein.

Dazu fügen wir zunächst eine **Dummy-Box** hinzu, die eine Schachfigur darstellt.

Ich habe sie `DebugBoxToBeHidden` genannt und die Größe der Box auf `2 3 2` gesetzt, um große (*große Y-Achse*) Schachfiguren zu berücksichtigen. Später werden wir die Box ausblenden, indem wir ihre Eigenschaft `Exists` auf `false` setzen.

Sobald Sie eine Dummy-Box haben, fügen Sie zwischengeschaltete `TCastleTransform` Komponenten hinzu, um:

- den Pfeil (*Kegel und Zylinder*) in die Horizontale zu drehen,
- den Pfeil von der Box wegzubewegen,
- den Pfeil um die Box herum zu drehen,
- den Pfeil zu skalieren."

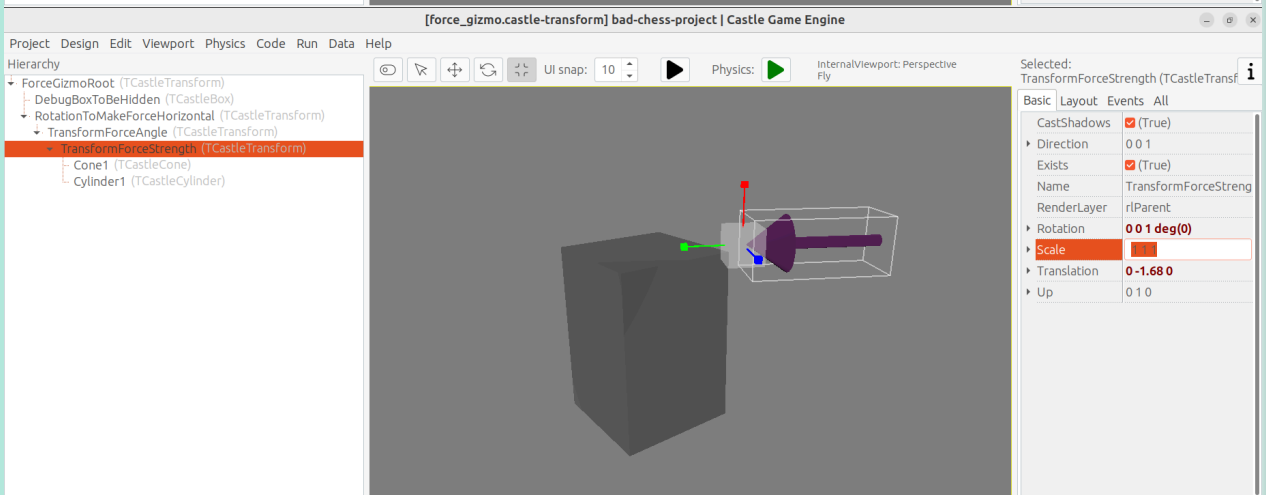
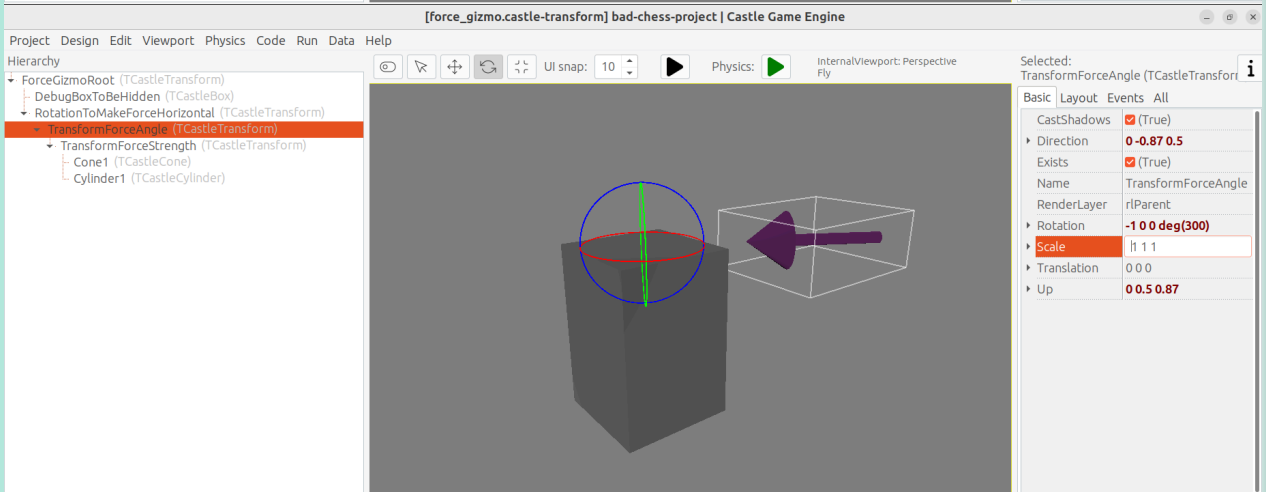
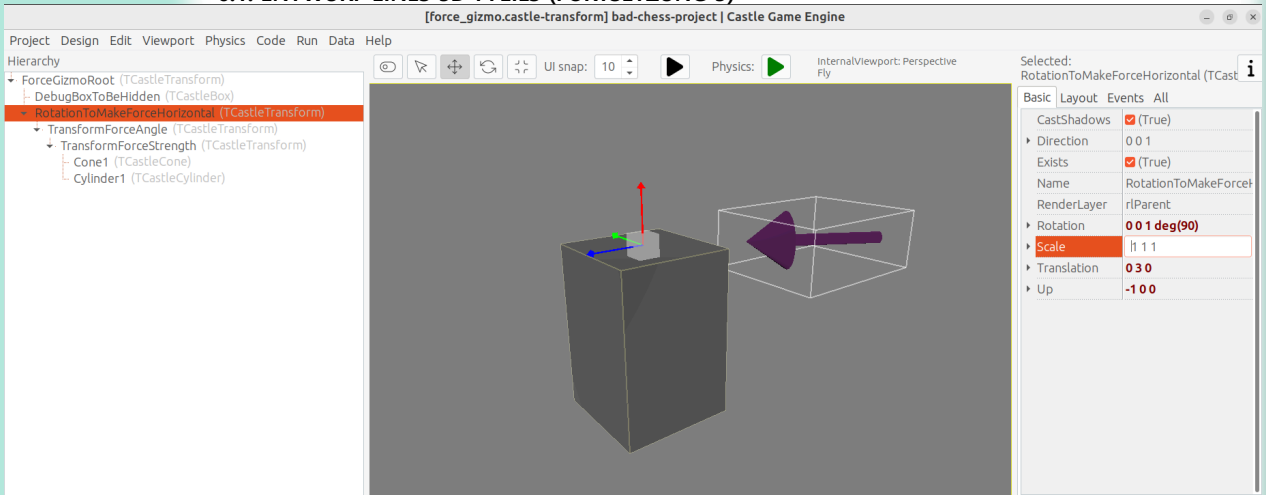
Es gibt mehrere gültige Möglichkeiten, dies zu erreichen. Der wichtigste Ratschlag ist, nicht zu zögern, eine verschachtelte Komposition zu machen, d.h. eine `TCastleTransform` innerhalb einer anderen `TCastleTransform` innerhalb einer anderen `TCastleTransform` zu platzieren und so weiter. Lassen Sie jede `TCastleTransform` eine einzelne Funktion ausführen. Gehen Sie Schritt für Schritt vor und Sie werden zu einer gültigen Lösung kommen (*und es gibt wirklich eine Reihe von Möglichkeiten, dies zu arrangieren*).

Sehen Sie sich meine Anordnung auf den Screenshots unten an. Wenn Sie nicht weiterkommen, verwenden Sie einfach den Entwurf aus unserem resultierenden Projekt in <https://github.com/castle-engine/bad-chess/> (*im Unterverzeichnis project/version_2_with_code*).





6.1. ENTWURF EINES 3D-PFEILS (FORTSETZUNG 3)





6.1 ENTWURF VON 3DARROW (FORTSETZUNG 4)

Das Ergebnis meines Entwurfs ist, dass ich per Code folgendes erreichen kann:
Die Eigenschaft `Rotation` der Komponente `TransformForceAngle` so anzupassen, dass sie eine einfache Drehung um die X-Achse erzielt. Der Winkel dieser Drehung kann vom Benutzer gewählt werden und der Pfeil umkreist effektiv die `Debug-Box` (*Schachfigur*).
Passen Sie die Eigenschaft `Y` der Komponente `TransformForceStrength` an.
Die Größe dieser Skala kann vom Benutzer gewählt werden, um die Stärke zu visualisieren.
Vergessen Sie nicht, `Exists` der Komponente `DebugBoxToBeHidden` auf `false` zu setzen, sobald dies geschehen ist.

6.2. HINZUFÜGEN DES PFEILS ZUM HAUPTENTWURF

Um zu testen, ob es funktioniert, fügen Sie den Pfeilentwurf mit dem Editor zum Hauptentwurf hinzu.

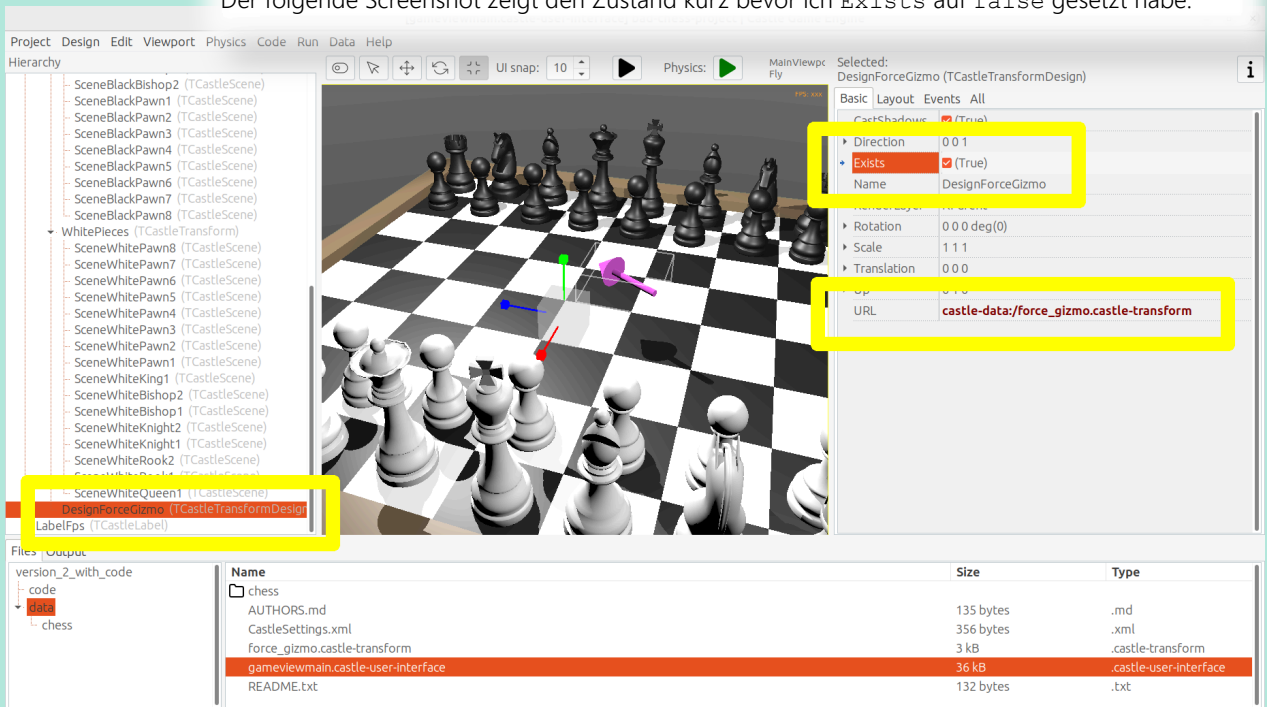
Speichern Sie den Entwurf `force_gizmo.castle-transform`, öffnen Sie unseren Hauptentwurf in `gameviewmain.castle-user-interface`, wählen Sie die Komponente `Items` im `MainViewport` aus und ziehen Sie die Datei `force_gizmo.castle-transform` (aus dem Panel "Dateien" unten) auf die Hierarchie.

Das Ergebnis sollte sein, dass eine neue Komponente namens `DesignForceGizmo1` erstellt und als untergeordnetes Element von `Items` platziert wird. Die Komponentenkategorie ist `TCastleTransformDesign`, was bedeutet, dass es sich um eine Instanz von `TCastleTransform` handelt, die aus einer anderen Datei mit der Erweiterung `.castle-transform` geladen wurde. Die `URL`-Eigenschaft dieser Komponente sollte automatisch auf unsere `force_gizmo.castle-transform`-Datei gesetzt werden.

Benennen Sie diese Komponente einfach in `DesignForceGizmo` um (das bleibt Ihnen überlassen, aber ich denke, das macht die Sache klarer - wir werden immer nur ein solches `Gizmo` brauchen).

Ändern Sie außerdem die `Exists`-Eigenschaft dieser Komponente auf `false`, denn zunächst wollen wir nicht, dass dieses Component sichtbar ist oder mit der Maus ausgewählt werden kann.

Der folgende Screenshot zeigt den Zustand kurz bevor ich `Exists` auf `false` gesetzt habe.





6.3. STEUERUNG DES PFEILS DURCH DEN BENUTZER

Wir müssen die Felder, die den aktuellen Winkel und die Stärke beschreiben, deklarieren und initialisieren.

Fügen Sie dies dem privaten Abschnitt der Klasse `TViewMain` hinzu:

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

Dann lassen Sie uns einige Konstanten setzen. Sie können sie am Anfang der `GameViewMain`-Implementierung der Einheit deklarieren:

```
const  
    MinStrength = 1;  
    MaxStrength = 1000;  
  
    MinStrengthScale = 1;  
    MaxStrengthScale = 3;  
  
    StrengthChangeSpeed = 30;  
    AngleAChangeSpeed = 10;
```

Fügen Sie der `uses`-Klausel neue notwendige Einheiten hinzu: `Math`, `CastleUtils`.

Schließlich fügen Sie der Datei `TViewMain.Start` ein zusätzliches Stück Code hinzu, um alles zu initialisieren:

```
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);
```

HINWEIS: Wir initialisieren die Komponenten innerhalb unseres `DesignForceGizmo`-Designs mit dem Aufruf `DesignForceGizmo.DesignedComponent(...)`. Dies ist notwendig, da Sie im Allgemeinen mehrere Instanzen des Entwurfs `force_gizmo.castle-transform` in Ihrer Ansicht platziert haben können. Daher können die veröffentlichten Felder der Ansicht nicht automatisch mit Komponenten in verschachtelten Designs verknüpft werden.

Darüber hinaus synchronisieren wir die `Single`-Field `ForceStrength` und `ForceAngle` mit ihren Gegenstücken `TCastleTransform`-Instanzen. `Single` in **Pascal** ist eine einfache Gleitkommazahl, die super-einfach zu manipulieren ist. Wir behandeln zwei `TCastleTransform` Instanzen oben als eine ausgefallene Art, diese Zahlen als 3D Rotation und Skalierung zu visualisieren.

Sie können nachschlagen, was die `MapRange`-Funktion in der **Castle Game Engine** API-Referenz tut. Kurz gesagt, es ist eine bequeme Art, eine lineare Interpolation durchzuführen und von einem Bereich in einen anderen zu konvertieren.

Nachdem wir nun alles initialisiert haben, wollen wir den `DesignForceGizmo` anzeigen, wenn der Benutzer eine Schachfigur auswählt. Wir haben bereits einen Code, um die Schachfigur beim Mausklick auszuwählen. Erweitern Sie ihn einfach, um das `DesignForceGizmo` anzuzeigen und es an der ausgewählten Schachfigur neu zu positionieren.





6.3. STEUERUNG DES PFEILS DURCH DEN BENUTZER (FORTSETZUNG)

```
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;
```

HINWEIS: Sie werden sich vielleicht fragen, ob ein alternativen Ansatz möglich wäre, bei dem wir DesignForceGizmo nicht neu positionieren, sondern stattdessen dynamisch das übergeordnete Element ändern, wie `DesignForceGizmo.Parent := ChessPieceSelected.Parent`. Das würde auch funktionieren, leider mit einigen zusätzlichen Komplikationen: die Drehung des ausgewählten Objekts, sobald wir es anklicken, würde auch das **Gizmo** drehen. Das würde die Berechnung der "gewünschten Schnipprichtung" später komplizierter machen. Daher habe ich mich für den einfacheren Ansatz entschieden, den DesignForceGizmo einfach neu zu positionieren.

Wenn Sie mit dem alternativen, komplizierten Ansatz experimentieren wollen, nur zu. Eine Lösung wäre, DesignForceGizmo so zu gestalten, dass Sie später `TransformForceAngle.GetWorldView(WorldPos, WorldDir, WorldUp)` ausführen und das resultierende `WorldDir` als Krafrichtung verwenden können.

Aber da wir die Dinge einfach halten ... sind wir fast fertig.

Sie können das Spiel starten und sehen, dass die Auswahl einer Schachfigur das Pfeil-Gizmo richtig anzeigt.

Jetzt muss der Benutzer noch die Richtung und die Stärke des Pfeils ändern können.

Wir können dies tun, indem wir die Tasten auswerten, die der Benutzer in der Update-Methode drückt.

Der folgende Code ermöglicht es, den Pfeil zu drehen (*ihn um die Schachfigur kreisen zu lassen*), indem man die linke und rechte Pfeiltaste drückt, und die Kraftstärke zu ändern (*den Pfeil zu skalieren*), indem man die Pfeiltasten nach oben und unten drückt. Fügen Sie diesen Code zu Ihrer bestehenden Update-Methode hinzu:

```
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;
```





🔗 SCHNIPPEN SIE DIE SCHACHFIGUR!

Es sieht so aus, als hätten wir alles Wissen, das wir brauchen.

- Wir wissen, wie man die Schachfigur schnippt,
- wir wissen, welche Schachfigur geschnipst werden muss,
- wir kennen die Richtung und die Stärke des Schnipsens.

Sie können sich den Code ansehen, den wir ein paar Abschnitte zuvor in der Übung erstellt haben "Schieben Sie die Schachfigur mit Hilfe der Physik".

Unser neuer Code wird ähnlich sein.

Fügen Sie ihn zur Implementierung der Methode Press hinzu:

```
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;
```

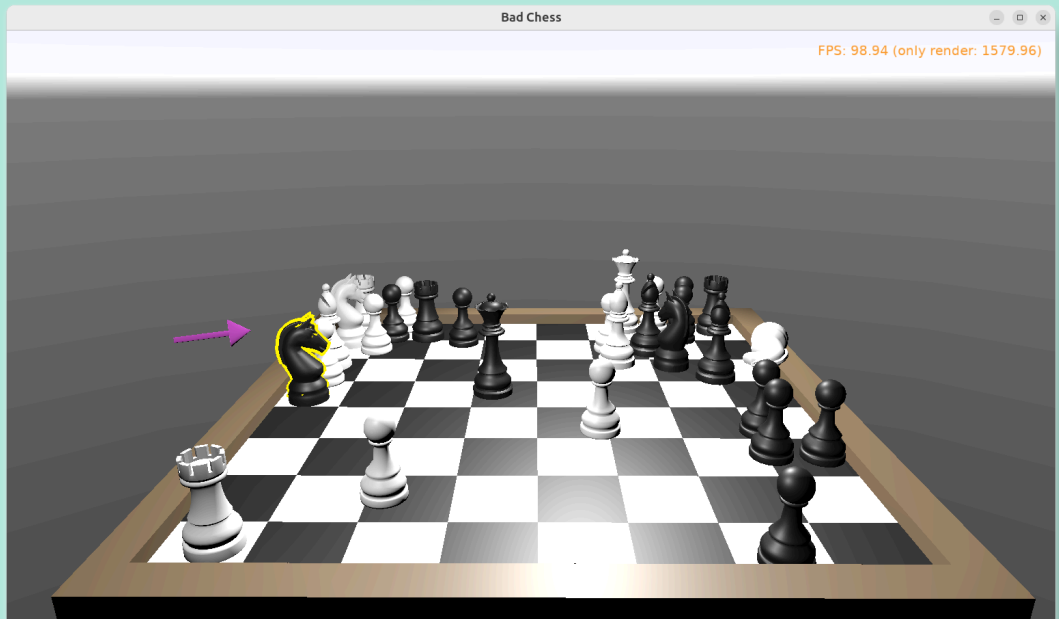
Je nachdem, wie Sie den `force_gizmo.castle-transform-Entwurf` gestaltet haben, müssen Sie möglicherweise die `ForceDirection`-Berechnung anpassen, insbesondere den zweiten Parameter von `RotatePointAroundAxis`, der eine Richtung angibt, die verwendet wird, wenn der Winkel Null ist.

Unser Wert `Vector3(-1, 0, 0)` hat nichts Magisches an sich, er folgt einfach unserem `force_gizmo.castle-transform-Design`.

Starten Sie das Spiel und sehen Sie, dass Sie jetzt die Schachfiguren schnippen können!

- Wähle die Schachfigur mit einem Mausklick aus.
- Drehen Sie die Kraft mit den Pfeiltasten links und rechts.
- Ändern Sie die Stärke der Kraft mit den Pfeiltasten nach oben und unten.
- Schnippe die Schachfigur, indem Sie Enter drücken.
- Wiederholen Sie das.





8 SCHLUSSFOLGERUNG UND ZUKÜNFTIGE IDEEN

Laden Sie einen Freund ein, mit Ihnen zu spielen. Benutze einfach abwechselnd die Maus, um ihre Schachfiguren zu schnippen, und haben Sie Spaß.

Ich bin mir sicher, dass Sie jetzt mehrere Möglichkeiten erfinden können, um dies zu verbessern.

- **Vielleicht** sollte jeder Spieler nur seine eigenen Schachfiguren schnippen können?
Wir wissen bereits, welche Schachfigur schwarz oder weiß ist (das boolesche `Field Black` in `TChessPieceBehavior`), obwohl wir es oben nicht für irgendetwas verwendet haben. Man sollte verfolgen, welcher Spieler das Objekt zuletzt geschnipst hat (*schwarz oder weiß*), und beim nächsten Mal nur die andere Seite zu zulassen.
- **Vielleicht** möchten Sie eine Benutzeroberfläche anzeigen, z. B. ein Etikett, um anzuzeigen, wer an der Reihe ist? Legen Sie einfach eine `TCastleLabel`-Komponente in der Ansicht ab, und ändern Sie die Beschriftung, wann immer Sie wollen.
- **Vielleicht** möchten Sie den aktuellen `Kraftwinkel` und die `Kraftstärke` anzeigen - entweder als Zahlen, oder als bunte Balken? Verwenden Sie `TCastleRectangleColor` für ein triviales Rechteck mit optionalem Rand und optional mit einer Farbe gefüllt.
- **Vielleicht** möchten Sie ein richtiges Schachspiel implementieren?
Sicher, verfolgen Sie einfach im Code alle Schachfiguren und die Schachbrettkacheln - was ist wo. Dann füge eine Logik hinzu, mit der der Spieler auswählen kann welche Figur wohin ziehen soll. Fügen Sie eine Validierung hinzu.
Es gibt standardisierte Protokolle für die Kommunikation mit "Schachengines". Sie müssen also nicht Ihre eigene Schach-KI von Grund auf implementieren.
- **Vielleicht** möchten Sie Netzwerke nutzen? Sie können eine Reihe von Netzwerklösungen (*jede Pascal-Bibliothek*) zusammen mit der **Castle Game Engine** verwenden.
Siehe https://castle-engine.io/manual_network.php .
Wir haben die Engine mit `Indy` und **RNL (Realtime Network Library)** verwendet.
Für die Zukunft planen wir die Integration der Engine mit **Nakama**, einem **Open-Source Server** und **Client-Framework** für Multiplayer-Spiele.





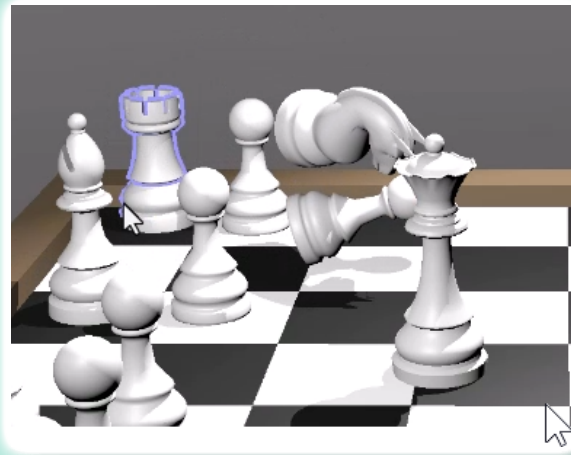
8 SCHLUSSFOLGERUNG UND ZUKÜNFTIGE IDEEN (FORTSETZUNG)

- Vielleicht möchten Sie dieses Spiel auch auf anderen Plattformen, insbesondere auf Handys einsetzen? Nur zu.
Der Code, den wir oben geschrieben haben, ist bereits plattformübergreifend und kann mit der Castle Game Engine für jedes Android oder iOS kompiliert werden.
Unser Build-Tool erledigt alles für Sie. Sie erhalten eine fertige **APK-**, **AAB-** oder **IPA-Datei** zur Installation auf Ihrem Telefon. Siehe die Dokumentation der Engine auf **https://castle-engine.io/manual_cross_platform.php** .
Allerdings funktionieren Tastatureingaben auf dem Handy nicht.
Sie müssen eine neue Benutzeroberfläche erfinden und implementieren, um die Kraft zu drehen, die Stärke zu ändern und die Schachfigur tatsächlich zu werfen.
Am einfachsten ist es, einfach anklickbare Buttons anzuzeigen, um die entsprechenden Aktionen auszuführen. Die `TCastleButton`-Klasse der Engine ist ein Button mit einem frei anpassbaren Aussehen.

Wenn Sie mehr über die Engine erfahren wollen, lesen Sie die Dokumentation auf **<https://castle-engine.io/>** und treten Sie unserer Community im Forum und auf Discord bei: **<https://castle-engine.io/talk.php>**

Zu guter Letzt, wenn Ihnen dieser Artikel und die Engine gefallen, würden wir uns freuen, wenn Sie uns auf **Patreon** unterstützen **<https://www.patreon.com/castleengine>** .
Wir zählen wirklich auf Ihre Unterstützung.

Und schließlich, vor allem, viel Spaß! Spiele zu entwickeln ist ein wilder Prozess und das Experimentieren mit "was sich gut anfühlt" ist der richtige Weg, es zu tun.
Ich hoffe, Sie werden es genießen.

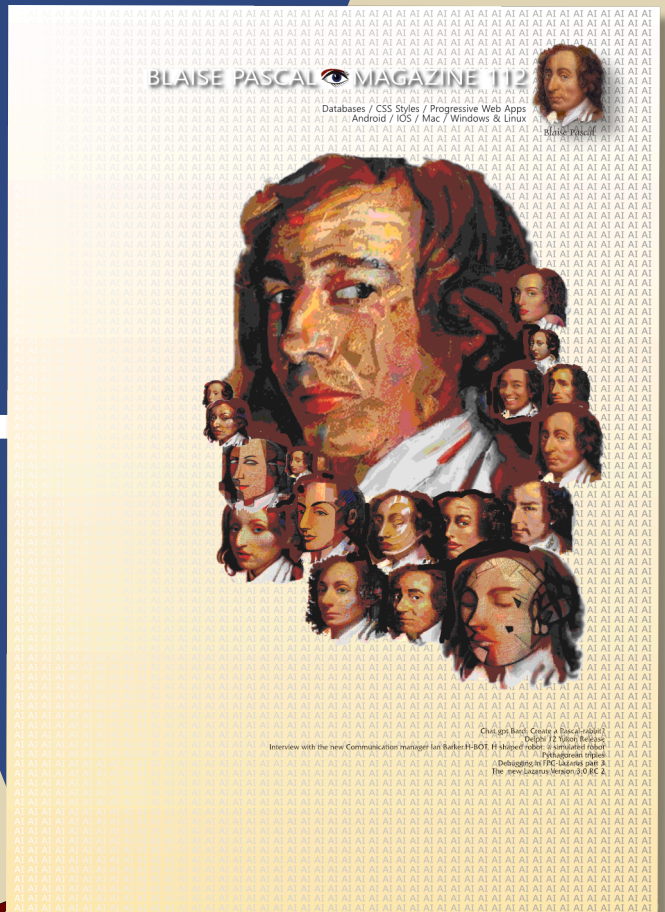


LAZARUS HANDBOOK (PDF) + SUBSCRIPTION 1 YEAR

- 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



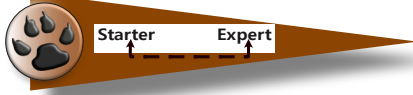
POCKET PACKAGE (2BOOKS)

EXCLUDING VAT AND SHIPPING

LAZARUS HANDBOOK PRICE: € 25,00



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



ES GEHT UM DIE PRÄSENTATION

Wir haben bereits das Fenster "Watches and Locals" verwendet, um Daten während der Fehlersuche zu betrachten. Aber nicht alle Daten sind gleich und es gibt viele Möglichkeiten, sie zu betrachten.

HINWEIS: Dieser Artikel basiert auf dem **FpDebug Debugger in Lazarus 2.2.6** und dem Projekt mit **DWARF 3**. Die Verwendung eines anderen Debugger-Backends oder anderer Einstellungen kann zu einer anderen Anzeige der Werte führen. Einige der in diesem Artikel gezeigten Funktionen erfordern **Lazarus 3.0**

OUR SAMPLE CODE FOR THIS ARTICLE:

```
1. program project1;
2.
3. uses Classes, StrUtils;
4.
5. const
6.   FORMAT_HEX = 0;
7.   FORMAT_OCT = 1;
8.   FORMAT_BIN = 2;
9.
10. type
11.
12.   TApiData = LongWord;
13.
14.   (* This API expects a LongWord made up from
15.    1 Byte (Bits 24..31): A Digit Count (Size)
16.    1 Byte (Bits 16..23): A Format
17.    1 Word (Bits 0..15): A Number
18.   *)
19.
20.   { TMyAPIBase }
21.
22.   TMyAPIBase = class
23.   private
24.     FList: TStringList;
25.   public
26.     constructor Create;
27.     destructor Destroy; override;
28.     procedure Add(ASize: Byte; AFormat: Byte; ANumber: Word);
29.     function GetText: String;
30.   end;
31.
32.   TMyAPI = class(TMyAPIBase)
33.   type
34.     TApiDataStruct = packed record
35.       Size: Byte;
36.       Format: Byte;
37.       Number: Word;
38.     end;
39.   public
40.     procedure ApiStore(AData: TApiData);
41.     procedure Print;
42.   end;
```





```
43.
44. constructor TMyAPIBase.Create;
45. begin
46.   inherited Create;
47.   FList := TStringList.Create;
48. end;
49.
50. destructor TMyAPIBase.Destroy;
51. begin
52.   FList.Destroy;
53. inherited Destroy;
54. end;
55.
56. procedure TMyAPIBase.Add(ASize: Byte; AFormat: Byte; ANumber: Word);
57. begin
58.   case AFormat of
59.     FORMAT_HEX:
60.       FList.Add(Dec2Numb(ANumber, ASize, 16));
61.     FORMAT_OCT:
62.       FList.Add(Dec2Numb(ANumber, ASize, 8));
63.     FORMAT_BIN:
64.       FList.Add(Dec2Numb(ANumber, ASize, 2));
65.     else
66.       FList.Add(Dec2Numb(ANumber, ASize, AFormat));
67.   end;
68. end;
69.
70. function TMyAPIBase.GetText: String;
71. begin
72.   Result := FList.Text;
73. end;
74.
75. procedure TMyAPI.ApiStore(AData: TApiData);
76. var
77.   d: TApiDataStruct;
78. begin
79.   d := TApiDataStruct(AData);
80.   with d do
81.     Add(Size, Format, Number);
82. end;
83.
84. procedure TMyAPI.Print;
85. begin
86.   WriteLn(GetText);
87. end;
88.
89. function GetApiValue(S, F, N: Integer): TApiData;
90. begin
91.   Result := S << 24 + F << 16 + N;
92. end;
93.
94. var
95.   Api: TMyAPI;
96.   Val: TApiData;
97.
98. begin
99.   Api := TMyAPI.Create;
100.
101.   Val := GetApiValue(4, FORMAT_HEX, 42);
102.   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.
```

Wir haben ein Objekt mit einer benutzerdefinierten **API**, die mehrere Werte in einem einzigen Parameter benötigt und zwar bit-packed. Wir rufen es auf, um den Wert 42 als **Hex**, **Oct**, **Bin** und Decimal formatiert auszugeben::

```
002A  
00000052  
0000000000101010  
042
```

Aber stattdessen bekommen wir:

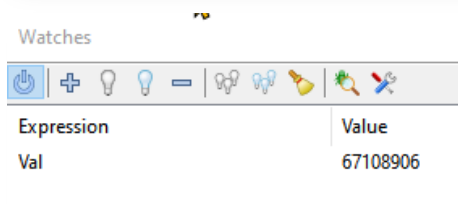
```
000000000000000000000000000000000000000000000000400  
0000000000000000000000000000000000000000000000801  
0000000000000000000000000000000000000000000001002  
00000000000000000000000000000000000000000000030A
```

Um zu debuggen lassen wir es mit **F9** bis zu einem Breakpoint laufen, den wir in Zeile 102 angelegt haben. Dies ist der erste Aufruf von


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

DAS ANZEIGEFORMAT

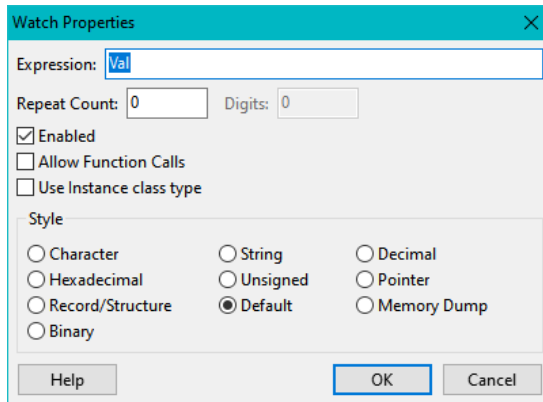
Wenn wir "Val" entweder im Fenster "**Locals**" oder im Fenster "**Watches**" betrachten, erhalten wir:



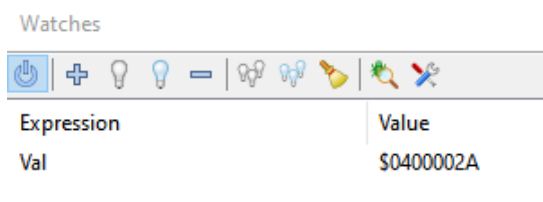
Leider sagt uns das nichts über die Bytes im Wert. Glücklicherweise bietet uns der **Debugger** einige Optionen, die wir im **Dialogfeld Eigenschaften der Watches** finden können.

Der schnellste Weg, ihn zu öffnen, ist ein Doppelklick auf die Watch. Alternativ können wir auch "Eigenschaften" aus dem Kontextmenü der **Watches** wählen oder den  Button drücken.





Im Moment interessieren wir uns für den "Stil" oder allgemeiner gesagt für das "Display format". Da "Val" eine Zahl ist, sind wir an den **Styles** interessiert, die sich auf Zahlen beziehen. Wir brauchen ein Format, das es uns erlaubt, die Grenzen der enthaltenen Bytes zu sehen. Eine gute Wahl dafür ist "**Hexadezimal**". Die Ausgabe ändert sich zu:



Val wurde mit den Werten initialisiert, die an:

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

Wir vergleichen den Hex-Wert mit der Beschreibung von "TApiData".

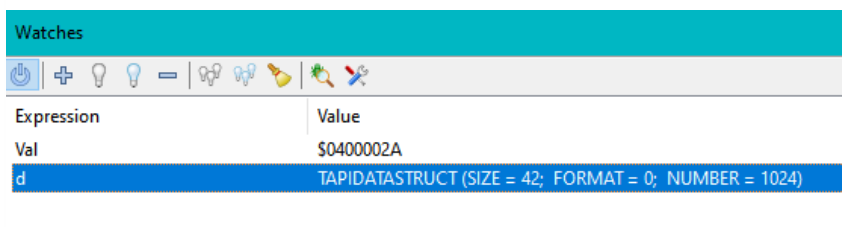
- Byte 1 = 04: Dies ist die gewünschte Größe
 - Byte 2 = 00: Dies ist FORMAT_HEX
 - Byte 3 und 4: = \$002a = 42: Der von uns angegebene Wert
- Das sieht also gut aus.

DAS DETAILFENSTER


Als Nächstes gehen wir in die Zeile 80 und machen eine Pause nachdem

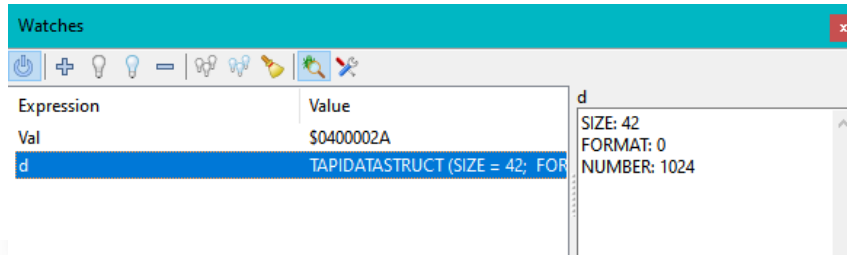
```
d := TApiDataStruct(AData);
```

ausgeführt worden ist. Jetzt können wir einen Blick auf "d" werfen.



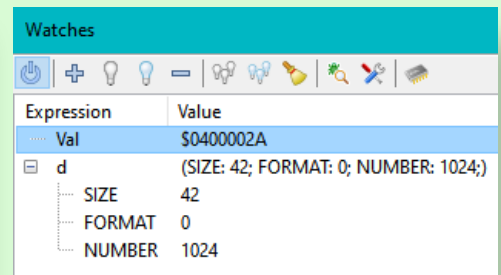
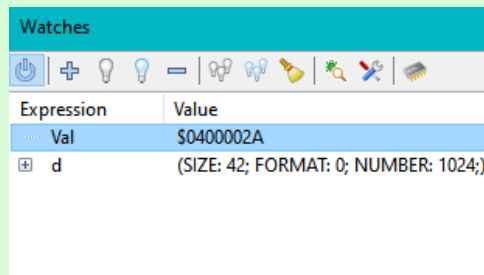


Bevor wir mit der Fehlersuche fortfahren, sollten wir uns noch ein wenig mit dem Beobachtungsfenster beschäftigen. Die obige Darstellung zeigt alle Felder in einer Zeile, das mag für die 3 Felder funktionieren, die wir haben, aber wenn es mehr Felder gibt, würden sie abgeschnitten werden. Hierfür gibt es das "Detailfenster". Es kann mit dem Button  umgeschaltet werden und zeigt den Inhalt der ausgewählten Überwachung in eine großzügigeren Form.



Diese grüne Farbe akzentuiert die Lazarus-Version 3.0

In **Lazarus 3.0** können **Watches** auch in der Hauptansicht aufgeklappt werden. Strukturierte Werte haben ein + Symbol, um sie zu entfalten.

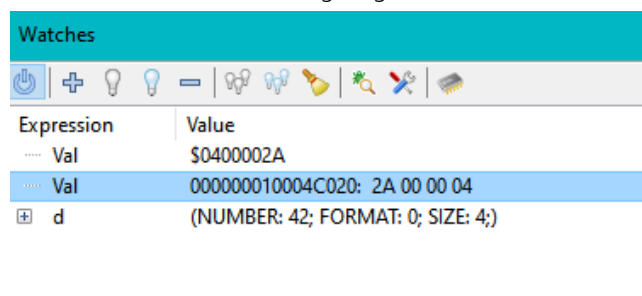


Um die Fehlersuche fortzusetzen: Während "Val" noch korrekt aussieht, entsprechen die Werte in "d: TAPIDataStruct" nicht unserer Erwartung entsprechen. Und das, obwohl "TAPIDataStruct" die Felder Größe, Format und Anzahl in der gleichen Reihenfolge enthält, in der die Bytes in "Val" vorhanden sind.

DUMPING SPEICHER

Es ist an der Zeit, einen weiteren Blick auf "Val" zu werfen. Gehen wir zurück zum Dialog der Überwachungseigenschaften und wählen wir "Memory Dump". Eigentlich werden wir eine zweite Überwachung für "Val" als Speicherauszug hinzufügen, so dass wir sie mit der hexadezimalen Darstellung von "Val" vergleichen können.

In **Lazarus 3.0** können wir dann die neue "Val"-Überwachung per Drag&Drop ziehen um direkt unter der existierenden "Val" angezeigt zu werden. In 2.2.6 wird die neue "Val" am Ende der Liste sein.





Jetzt haben wir einige interessante Informationen. Die Bytes im Speicher beginnen mit \$2A, dem niedrigsten Byte von "Val". Dies ist ein **Little-Endian-Rechner**. Wenn man also die Daten als eine einzige LongWord-Nummer betrachtet, steht die Größe ganz vorne, im Speicher jedoch ganz am Ende. Wir müssen die Reihenfolge, in der der Datensatz definiert ist, ändern, damit sie mit dem Layout im Speicher übereinstimmt.

```
33. type
34.   TApiDataStruct = packed record
35.     Number: Word;
36.     Format: Byte;
37.     Size: Byte;
38.   end;
```

Ein erneuter Start der Anwendung zeigt uns nun das richtige Ergebnis. Wir sind fertig mit der Debug-Sitzung, aber lassen Sie uns fortfahren und einen Blick auf einige weitere Funktionen werfen, die das Überwachungsfenster zu bieten hat.

VARIABLE ODER EXPRESSION

In den vorangegangenen Artikeln wurden bereits einige Beispiele für das Hinzufügen von Ausdrücken anstelle von Variablen in das **WatchesFenster** gegeben. **FpDebug** kann die meisten Ausdrücke interpretieren, die auch **Fpc** kann. Zugriff auf Felder, Array-Einträge, Arbeit mit Zeigern, Arithmetik, Type Casts, all das. Allerdings kann es noch nicht auf Eigenschaften zugreifen, wenn diese eine `Getter`-Funktion verwenden.

Mit Lazarus 3.0 im obigen Beispiel hätten wir `d := TApiDataStruct(AData);` nicht benötigt

Während **FpDebug** bereits viele `Typecasts` (*und Konvertierungen*) kannte, kann es nun auch beliebige Daten nach `Record` casten, solange die Daten die gleiche Größe wie der `Record` haben.

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

Mit **Lazarus 2.2.6** oder für Daten, die eine andere Größe haben als der Zieltyp des Casts, ist es erforderlich, die Adresse zu nehmen und in einen Zeiger zu casten. Glücklicherweise ist es möglich, "`^TApiDataStruct`" zu schreiben, um es als Zeiger auf den Typ zu verwenden.

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

HINWEIS: Bei der Verwendung von Debugger-Backends ohne **FpDebug**, wie dem reinen "**gdb**" oder dem reinen "**lldb**", funktionieren einige Ausdrücke möglicherweise nicht. Einige Operatoren existieren möglicherweise nicht oder sie haben eine andere Priorität, was zu einem anderen Ergebnis führt.





AUFRUFENDE FUNKTIONEN

Watch-Ausdrücke können Funktionsaufrufe enthalten. Diese Funktion muss in den globalen IDE-Optionen unter **Debugger** → **General** aktiviert werden. Und sie muss dann in den Eigenschaften jeder Überwachung aktiviert werden, die davon Gebrauch machen will.

Der Grund für diese explizite Aktivierung ist, dass die Auswertung eines Funktionsaufrufs unerwünschte Nebeneffekte haben kann.

Wenn die aufgerufene Funktion eine globale Variable oder ein Feld eines Objekts oder andere nicht lokale Daten ändert, bleibt diese Änderung nach der Auswertung bestehen.

Wenn Sie das Debugging fortsetzen, wird Ihre Anwendung mit diesen Änderungen ausgeführt und kann sich anders verhalten.

(Beachten Sie, dass auch die vorübergehende Zuweisung von Speicher den Zustand Ihrer Anwendung verändern kann, da es das zukünftige Speicherlayout verändern kann).

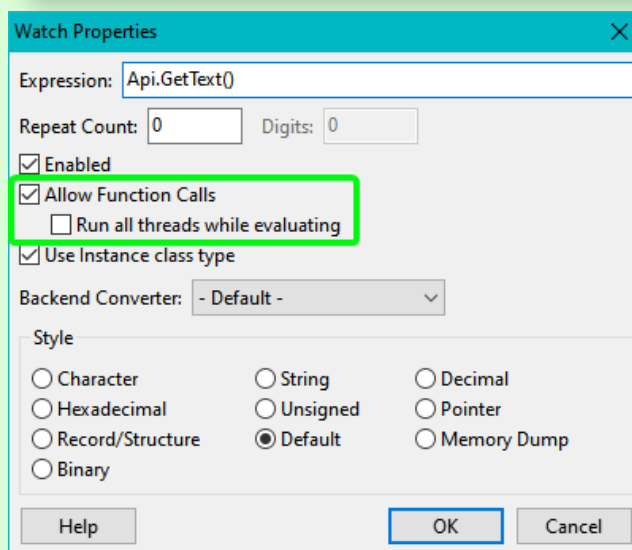
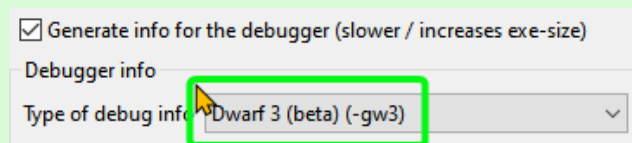
In **Lazarus 3.0** kann der Debugger den Funktionsaufruf ausführen, während alle anderen Threads angehalten werden. Dies ist der Standard, da ihre Anwendung pausiert ist und diese anderen Threads nicht laufen sollen. Andere Threads können jedoch laufen müssen, wenn die aufgerufene Funktion auf ein Ereignis von einem der anderen Threads warten kann.

Damit der Debugger eine Funktion aufrufen kann, muss das "()" angegeben werden, auch wenn es keine Parameter gibt.

HINWEIS: Werte einiger Datentypen können nicht als Parameter einer Funktion verwendet werden. Ebenso können Funktionen nur aufgerufen werden, wenn sie einen unterstützten Typ zurückgeben.

Laufen wir bis zur Zeile `writeln(Api.GetText);` durch und fügen die folgende Überwachung hinzu.

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





Watches	
Expression	Value
..... Api.GetText()	'002A'#\$0D#\$0A'00000052'#\$0D#\$0A'000000000101010'#\$0D#\$0A'042'#\$0D#\$0A

HINWEIS: Funktionen, die Strings (Long/Ansistring) zurückgeben oder Parameter dieses Typs haben, funktionieren nur mit **Lazarus 3.0**. Sie benötigen auch **DWARF 3 (oder höher)** als Debug-Information. Der Funktionsaufruf kann andernfalls abstürzen und ihre Debug-Sitzung beschädigt zurücklassen, d.h. ihre Anwendung wird nicht wie erwartet fortfahren können.

KLASSEN: DEKLARIERT ODER INSTANZIERT.

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

Standardmäßig zeigt der Debugger alle Daten entsprechend ihrer Deklaration an. Wenn Sie also "Sender" beobachten, werden Sie nur ein `TObject` sehen, das Ihnen sehr wenig Informationen liefert. Sie können natürlich die Variable selbst typisieren und eine Beobachtung wie `"TButton(Sender)"` hinzufügen. Aber wenn der Callback von verschiedenen Steuerelementen gemeinsam genutzt wird (*oder wenn Sie einfach etwas Bequemes wollen*), kann der Debugger das für Sie tun. Der Debugger kann die tatsächlich instanziierte Klasse des Objekts finden und die Watch entsprechend dieser Klasse anzeigen. Diese Funktion ist standardmäßig aktiviert. Sie können sie unter **Optionen → Debugger → General** global deaktivieren: **"Use instance class type' for new watches"**. Oder Sie können sie in den Eigenschaften jeder Watch durch Ändern von **"Use instance class type' for new watches"** umschalten. Dies funktioniert nur, wenn der deklarierte Typ eine Klasse ist (`TObject` oder ein Nachkomme). Wenn Sie eine Variable eines anderen Typs haben, wie `Data: PtrUInt`, dann müssen Sie sie zuerst in `"TObject(Data)"` typisieren. Dann wird der Debugger sie weiter in die Klasse der Instanz umwandeln.

ARRAYS UND "REPEAT COUNT"

Wenn ein **Array** deklariert (oder mittels `SetLength()`) ist, um eine bestimmte Anzahl von Einträgen zu haben, zeigt der Debugger die deklarierte Anzahl von Einträgen an (*oder die in den globalen Debugger-Optionen eingestellte Grenze*). Manchmal kann es vorkommen, dass Sie nach dem Ende des Arrays mehr Daten erwarten. In diesem Fall können Sie die **"repeat count"** in der watches-Eigenschaft einstellen und angeben, wie viele Elemente angezeigt werden sollen. Der **"repeat count"** kann auch die Anzahl der angezeigten Werte auf weniger als den Standardwert begrenzen. Damit **"repeat count"** funktioniert, muss die Watch für das gesamte Array gelten. Sie können keine "repeat count" für ein Array-Element `"MyArray[11]"` anwenden. Das bedeutet, dass Sie keinen Ausschnitt von z.B. 10 Elementen, beginnend bei 11, erhalten können.

In **Lazarus 3.0** können sie **Array Slices** erhalten, indem sie die `[11..20]` Syntax verwenden.

Die "Wiederholungszahl" kann auch auf Speicherabzüge angewendet werden. Standardmäßig geben sie den Speicher für die Größe des überwachten Wertes zurück (*z.B. 4 Bytes für einen LongInt*). Mit "repeat count" können Sie jedoch eine größere Menge an Speicher erhalten.





AKTIVIERT, DEAKTIVIERT ODER AUSGESCHALTET

Sie können alle oder einige der Watches deaktivieren. Dadurch wird verhindert, dass der Debugger sie auswertet. Dies kann nützlich sein, wenn eine Watch eine Funktion aufruft und Sie nicht wollen, dass diese aufgerufen wird, außer unter bestimmten Bedingungen. Es kann auch nützlich sein, wenn Sie einige Watches haben, die sehr große Datenmengen zurückgeben (*große Strukturen mit langen Arrays*). Das Abrufen solcher Daten kann einen Moment dauern, und vor allem, wenn Sie mehrere solcher Watches haben, können sie sich zu einer beachtlichen Zeit summieren. Mit dem "Power"-Button kann der aktuelle Zustand des Fensters erhalten werden. Alle Watches behalten ihren aktuell angezeigten Wert bei, solange die Power-Funktion aus ist.

Hinweis: Power wurde ursprünglich eingeführt, als der Debugger noch **gdb** verwendete und die Verarbeitung von Watches lange dauern konnte. Das Abschalten von Watches (**und stattdessen die Verwendung von Locals**) hat die Zeit zwischen den Schritten (**F7/F8**) unter **gdb** verbessert.

IN DAS CLIPBOARD

Im Kontextmenü gibt es mehrere Optionen, um den ausgewählten Wert in das ClipBoard zu kopieren.

- **„Name“ kopieren**
kopiert den Ausdruck (oder den Namen der lokalen Variable im Fenster "Lokale Variablen")
- **"Value" kopieren** kopiert den angezeigten Wert
- **"Rohwert kopieren" (Lokalfenster)** ist für Strings gedacht.
Es kopiert eine nicht-escaped Version des Strings in die Zwischenablage.

In **Lazarus 3.0** ist dies auch im Watch-Fenster verfügbar.

- "Datenadresse kopieren" (Lazarus 3.0) kopiert die Datenadresse (siehe Abschnitt unten).
- "Kopiere gesamten Eintrag" (Lazarus 3.0) kopiert Name, Wert und Adresse.

EINEN HINT NUTZEN

Variablen und Ausdrücke können auch direkt aus dem Quelltext-Editor heraus ausgewertet werden, indem man den Mauszeiger bewegt, bis die IDE einen Hinweis anzeigt.

Die IDE wird versuchen, die korrekten Grenzen des Ausdrucks zu ermitteln. Normalerweise das Wort unter der Maus, aber wenn es sich um ein Feld handelt, wird die IDE den Punkt finden und das vollständige "object.field" verwenden. Wenn sich die Maus am Ende einer Klammer befindet, wird die IDE den Anfang der Klammer finden und sogar eine Typumwandlung einbeziehen, falls vorhanden.

Die IDE wird keine Funktionen zur Auswertung von Hinweisen aufrufen. Aber sie kann (und tut es standardmäßig) jedes Objekt in die Klasse der Instanz umwandeln. Dies kann in den globalen Optionen umgeschaltet werden (Editor > Vervollständigung und Hinweise).

DATA ADDRESS (Lazarus 3.0)

Das Watches Fenster hat eine Spalte "Datenadresse".

Diese Spalte wird für Zeigertypen verwendet, einschließlich Typen, die einen internen Zeiger haben.

(T)Objects, AnsiStrings, dynamische Arrays haben alle einen internen Zeiger.

Dieser Zeiger wird in der Variablen gespeichert. Mit **@variable** erhält man die Adresse der Variablen, d. h. die Adresse, an der der Zeiger gespeichert ist. Die Daten selbst befinden sich an einer anderen Adresse, und diese Adresse wird in der Spalte "Datenadresse" angezeigt. Diese Adresse ist der Wert, den Sie erhalten würden, wenn Sie "Pointer(variable)" auswerten würden.

Watches		
Expression	Data-Address	Value
⊕ Api	\$0000000000CF970	TMyAPI(FList: TStringList(\$0000000000DF8D0); _vptr\$TOBJECT: SOC
..... @Api	\$000000010004C010	\$000000010004C010^: (FList: TStringList(\$0000000000DF8D0); _vpt
..... Val		67108906
..... @Val	\$000000010004C020	\$000000010004C020^: 67108906



Die Variable für das Objekt "Api" befindet sich an der Adresse \$10004C010. Die Daten für das Objekt befinden sich an der Adresse \$0000AE290. Die "Datenadresse" eines Zeigers ist derselbe wie der Wert des Zeigers. Ein Zeiger auf Adresse 123 hat seine Daten an dieser Adresse. Für "@Api" ist also die "Datenadresse" gleich dem Wert. Der Debugger zeigt zusätzlich den dereferenzierten Wert an. An der Adresse \$10004C010 enthält der Speicher die Daten \$0000AE290, die aber nicht angezeigt werden, weil der Debugger den Typ kennt und stattdessen das Array anzeigt.

"val" hat keine "Datenadresse" (*d.h. sie ist die gleiche wie die Adresse der Variablen*). Seine Daten sind direkt in der Variablen gespeichert. "Val" hat natürlich eine Adresse, die uns "@val" anzeigen wird. Hier ist die "Datenadresse" die Adresse, in der der Wert 67108906 (= \$0400002A) gespeichert ist. Die "Datenadresse" kann verwendet werden, um festzustellen, ob zwei **Strings/Arrays/Objekte** gleich sind oder nur den gleichen Inhalt haben.

STRUCTURES AND ARRAYS (LAZARUS 3.0)

Weiter oben in diesem Artikel haben wir gesehen, wie strukturierte Typen in der Überwachungsansicht erweitert werden können.

Das Watch-Fenster hat auch die Möglichkeit, die Einträge in einem Array zu erweitern. Nehmen wir an, wir hätten den folgenden Code:

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

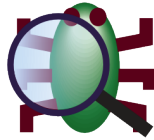
Dann könnten die **Watches** uns zeigen:

Expression	Value
ApiList	Len=25: ((FList: TStringList(\$00000000000BE1F0); _vptr\$TOBJECT: \$00000001000370E8^: Error: Unknown data;), nil, ...)
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

Der Inhalt des Arrays wird paginiert. Wenn das Array also 1000 Einträge hätte, müssten wir nicht durch eine lange Liste blättern. Es ermöglicht uns, die gewünschte Seite aufzurufen und die richtige Auswahl an Einträgen zu sehen. Wenn wir mehr als 10 Einträge benötigen, können wir im zweiten Eingabefeld eine andere Seitengröße angeben. Außerdem können wir in dem Fenster verschachtelte Werte aufklappen. Da das Feld Objekte enthält, können wir die Struktur dieser Objekte aufklappen. Wir können in jede beliebige Tiefe vordringen.

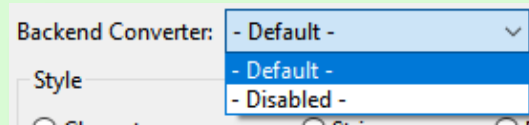
Wenn wir an einem bestimmten Wert in der Liste interessiert sind, **können wir diesen Wert per Drag & Drop an eine Position in der Hauptliste der Beobachtungen (außerhalb des aufgeklappten Bereichs)** ziehen. Dadurch wird automatisch eine neue Überwachung wie "ApiList[5]" erstellt. Dies funktioniert auch bei verschachtelten Einträgen, so dass wir nach dem Aufklappen von "ApiList[5]" "ApiList[5].FList" auswählen und per Drag&Drop ablegen können.





CONVERTER (LAZARUS 3.0)

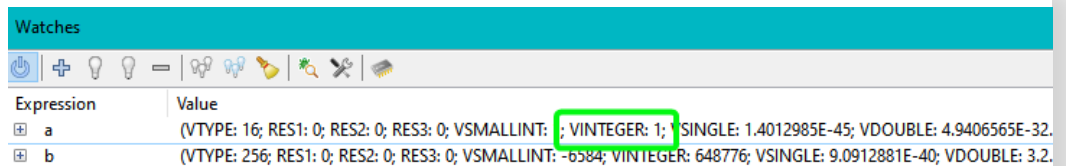
There is one more addition in the watches properties: Converter.



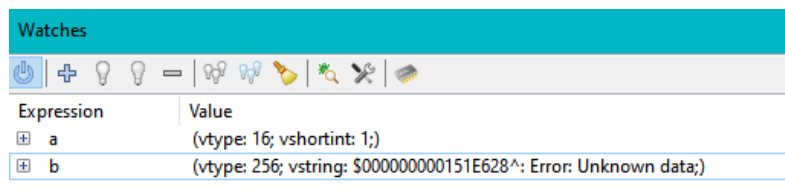
Standardmäßig bietet es nicht viel. Konverter müssen zunächst konfiguriert werden, entweder in den globalen Optionen oder in den Projektoptionen.
Wir werden uns einen Konverter als Beispiel ansehen, der ein Hilfsmittel zur Anzeige von "Varianten" ist. Zuvor sehen wir uns an, wie der Debugger eine Variable vom Typ Variante anzeigt. Betrachten wir den Code:

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

Mit **DWARF 2** zeigt der Debugger



Mit **DWARF 3** erhalten wir



In beiden Fällen wird der String-Wert für "b" nicht angezeigt. Dies liegt daran, dass eine Variante als `vtAnsiString : (VAnsiString: Pointer);` deklariert ist

Und der Compiler gibt in den Debug-Informationen nur "Pointer" an.

Um hier Abhilfe zu schaffen, verwenden wir einen Konverter.

Dieser kann entweder in den globalen Optionen oder für ein einzelnes Projekt in den Projektoptionen eingestellt werden.

In jedem Dialog gibt es eine Seite Debugger → Backend Converter.

Auf dieser Seite drücken wir auf "Add" und geben einen Namen unserer Wahl ein (z.B. "Variante").

Dies fügt einen Eintrag in der Checkliste oben hinzu. Dieser Eintrag muss angekreuzt werden.





In das Feld "Match types by name" müssen wir eine Zeile mit "variant" eingeben. Das bedeutet, dass dies für Variablen des Typs "Variant" gelten soll. Und wir müssen auswählen, was der Konverter tun soll. Dies ist die Dropdown-Liste "Aktion", und wir wählen "Variant in Werttyp umwandeln".

Was macht der Konverter? Nun, im Allgemeinen kann ein Konverter eine beliebige Übersetzung durchführen den Ergebniswert (*oder sogar Teile davon, wie Felder in einem Objekt*). Ein Konverter arbeitet im Debugger-Backend, so dass er auch zusätzliche Daten abrufen kann.

Die Funktion "**Variant in Werttyp konvertieren**" befasst sich speziell mit dem **FPC-Typ "variant"**. Er hat zusätzliches Wissen über diesen Typ, z.B. weiß er, dass "vtAnsiString" ein String ist. Es funktioniert nur mit dem **FPC-Typ "variant"**.

Da eine Anwendung aber auch Aliasnamen definieren kann, ist es immer noch notwendig, den/die Typennamen anzugeben, die abgeglichen werden sollen, wie wir es oben getan haben.

Es gibt auch einen Konverter "**CallSysVarToLStr**", der sich mit **Variant** befasst, aber auch benutzerdefinierte Variant einbezieht und Code in der debuggten Anwendung ausführt. Die Angabe von Typnamen, die übereinstimmen sollen, kann also auch zur Konfiguration verschiedener Konverter verwendet werden.

Nun, da wir einen Konverter konfiguriert haben, wird er standardmäßig auf alle Watches angewendet, die (*oder als Feld/Eintrag mit*) Variant Daten enthalten.

Einzelne Watches können ausgeschlossen werden, indem der den Konverter in ihren Eigenschaften deaktivieren.

Hätten wir den Konverter in der Checkliste deaktiviert, würde er nicht standardmäßig verwendet werden. Er würde nur von Watches verwendet werden, für die er in der Dropdown-Liste ihrer Eigenschaften ausgewählt ist.

(Der Typname wird weiterhin geprüft).

Jetzt (*mit beiden DWARF-Versionen*) zeigt uns der Debugger an:

Watches	
Expression	Value
a	1
b	'abc'

Der Konverter wird auch in einem Array auf jeden Eintrag angewendet. Oder in Objekten auf jedes Feld des Variantentyps.



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

Blaise Magazine Library

library.blaisepascalmagazine.eu

Issue 66 Open

Alan Turing Search

Search in PDF Dark mode Tester

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?
Dettef 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

NO ISSUE SELECTED
BLAISE PASCAL MAGAZINE

140

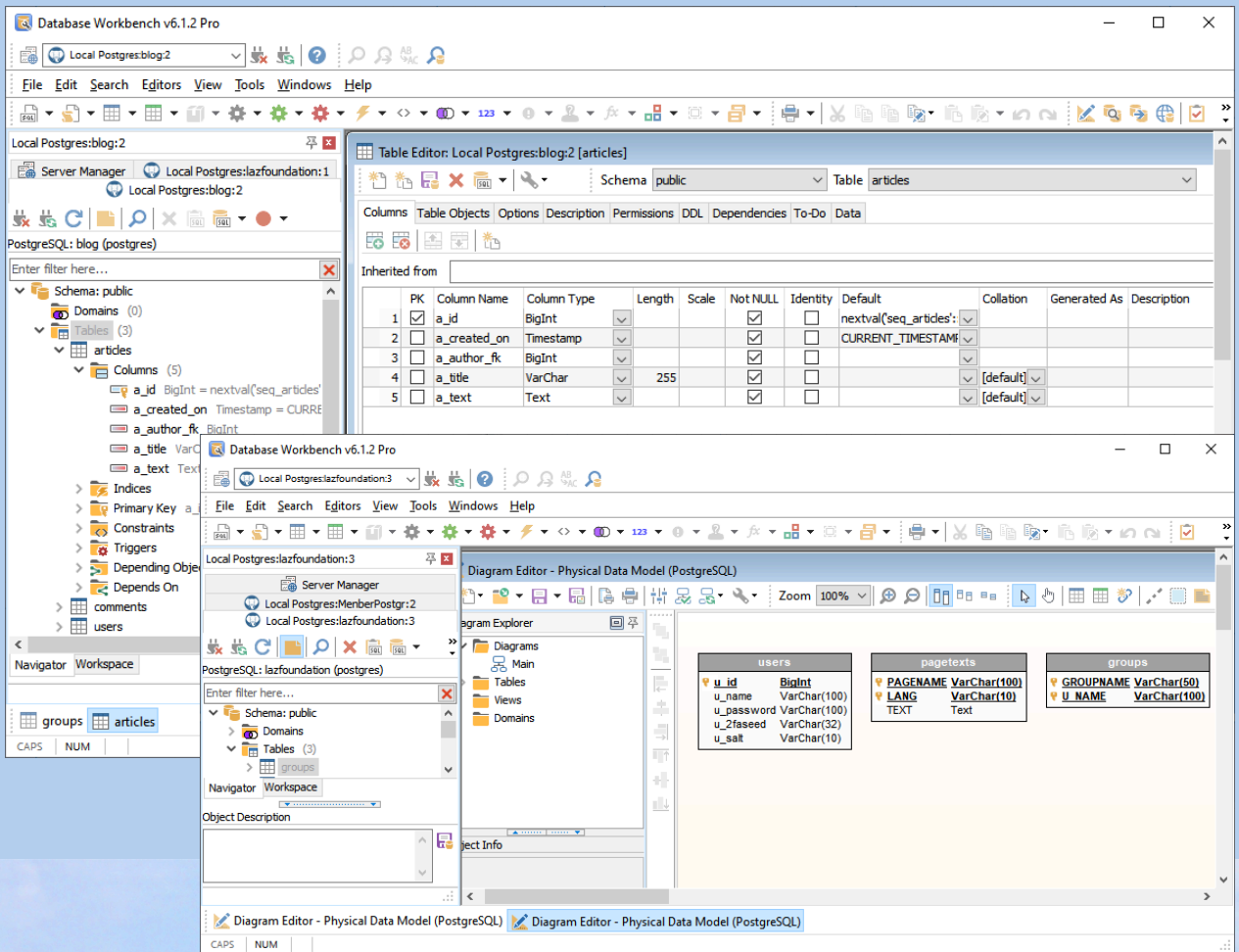
Load PDF...

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

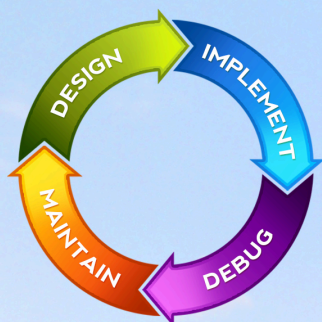
READ WHERE EVER THE INTERNET IS AVAILABLE



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



Database tools for developers

www.upsene.com



Starter

Expert

● KURZFASSUNG

In diesem Artikel zeigen wir, wie man einen gebrauchsfertigen Mechanismus zum **Senden** von Debug-Protokollen von einem **Pas2js**-Programm an eine in **Free Pascal** geschriebene HTTP-Server-Anwendung verwendet

Eine bessere Lösung zum Sammeln von Debug-Informationen besteht darin, sie direkt an den Webserver zu senden

① EINFÜHRUNG

In einer idealen Welt läuft die Anwendung reibungslos, und alle Eventualitäten während der Ausführung eines Programms werden angemessen behandelt. Wäre dies der Fall, könnten die Debug-Protokollierung und die Behandlung unerwarteter Fehler entfernt werden, sobald das Programm versandfertig ist.

In der Realität sind Programme oder ihre Ausführungsumgebung nicht perfekt und Benutzer tun unerwartete und unvorhergesehene Dinge: aus diesen beiden Gründen sind Debugging-Protokolle in ausgelieferten Anwendungen oft noch erforderlich.

Im Browser kann die **Pascal-Anweisung** `writeln()` verwendet werden, um in die Browserkonsole zu schreiben. Wenn gewünscht, kann das Ergebnis im HTML angezeigt werden. Sobald der Benutzer jedoch das Browserfenster schließt, geht diese Information verloren. Für die meisten Benutzer ist es eine schwierige, um nicht zu sagen unmögliche Aufgabe, die Informationen in der Browserkonsole auf Anfrage eines Support-Teams zu finden und zu übermitteln.

Eine bessere Lösung zum Sammeln von Debug-Informationen ist daher, sie direkt an den Webserver zu senden wo die Protokolle sofort untersucht oder für eine spätere Untersuchung gespeichert werden können. In diesem Artikel demonstrieren wir einen Mechanismus zur Übertragung solcher Debug-Informationen. Dieser Mechanismus ist standardmäßig in **Free Pascal** und **Pas2js** enthalten: `debugcapture`.

② ARCHITEKTUR

Die `Debug-Capture`-Funktionalität besteht - natürlich - aus 2 Teilen: ein Teil ist in `fcl-web` enthalten, der andere ist Teil von **Pas2js** und wird in einem **Pas2js**-Client-Programm verwendet. Die Unit `fpDebugCaptureSvc` ist Teil von **Free Pascals** `fpweb`-Paket zur Erstellung von HTTP-Server-Anwendungen: Sie können sie in ein HTTP-Server-Programm einbinden und mit einer einzigen Zeile Code aktivieren. Sie wird standardmäßig von der **simpleserver**-Anwendung eingebunden. Das in **Pas2js** enthaltene Programm `compileserver` bietet diese Funktionalität ebenfalls.

Die Funktionalität ist standardmäßig deaktiviert, der `command-line` muss `switch -u` angegeben werden, um die Debug-Aufzeichnung zu aktivieren: Wenn kein zusätzliches Argument angegeben wird, werden die aufgezeichneten Informationen auf der Konsole ausgegeben.

Wenn ein zusätzliches Argument für die Option `-u` angegeben wird, wird es als Dateiname interpretiert, unter dem die Ausgabe gespeichert werden soll. Die URL für den Dienst lautet standardmäßig `/debugcapture/`.

Der Client-Teil ist in der Unit `debugcapture` enthalten, die Teil von **Pas2JS** ist.

Sie enthält eine einfache Client-Komponente, die die Ausgabe an eine konfigurierbare URL sendet.

Im weiteren Verlauf dieses Artikels werden wir die Verwendung beider Seiten demonstrieren.

③ DER SERVERTEIL: TDEBUGCAPTURESERVICE

Die Unit `fpDebugCaptureSvc` enthält eine `TDebugCaptureService`-Komponente. Sie kann verwendet werden, um eine oder mehrere HTTP-Routen zu verarbeiten. Sie kann standardmäßig auf der Konsole oder in einer Datei protokollieren, aber es können zusätzliche Backends registriert werden. Diese Komponente hat die folgende Deklaration:



```
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;
```

Es gibt die folgenden Methoden:

- **HandleRequest**
Dies ist der Einstiegspunkt des Dienstes: Die Signatur dieser Methode ist so beschaffen, dass sie als der Handler einer Route im **HTTP**-Router des **fpWeb**-Servers. benutzt werden kann
- **RegisterHandler**
Sie können so viele Handler für eine `debug capture request` hinzufügen, wie Sie möchten. Sie registrieren einen Callback `aHandler` mit einem (eindeutigen) Namen `aName`. Der Name wird in Log Nachrichten verwendet und kann verwendet werden, um die Registrierung des Handlers aufzuheben.
- **UnregisterHandler**
kann verwendet werden, um einen Handler mit dem angegebenen Namen aus der Liste der **debug capture handler** zu entfernen.
- **JSONDataToString**
Diese Klassenmethode kann verwendet werden, um die JSON-Nutzdaten in eine Zeichenkette zu konvertieren. Sie übernimmt Sonderfälle wie `null` oder Objekte

Die folgenden Eigenschaften können verwendet werden:

- **Instance**
Dies ist eine globale Instanz der Komponente. Sie kann verwendet werden, um schnell eine Instanz des Debug-Erfassungsdienstes einzurichten.
- **LogFileName**
Wenn dieser Wert nicht leer ist, wird die Protokollierung der aufgezeichneten Debug-Output in eine Datei geschrieben.
- **LogToConsole**
Bei `True` wird eine nicht leere, aufgezeichnete Debug-Output an die Konsole gesendet.
- **CaptureToErrorLog**
Bei der Einstellung `True` wird die Ausgabe zusammen mit den Fehlermeldungen der Komponente an den `OnLog` Log Handler gesendet.
- **OnLog**
Dieses Ereignis wird verwendet, um Fehlermeldungen der Komponente zu protokollieren: Wenn ein Fehler während dieser beim Schreiben der Debug-Output in einen der Handler auftritt, wird er mit diesem Ereignis protokolliert. Wenn `CaptureToErrorLog` auf `true` gesetzt ist, werden alle erfassten Debug-Output ebenfalls an dieses Ereignis gesendet.
- **CORS**
Dies kann so konfiguriert werden, dass **CORS**-Preflight-Anfragen behandelt werden, so dass Sie den `debug capture service` unter einer anderen URL laufen lassen als die, unter der Ihre Anwendung bereitgestellt wird. Stellen Sie sicher, dass Sie **CORS** richtig konfigurieren, wenn Sie es aktivieren. Es ist keine gute Idee, allen möglichen Domänen die Nutzung dieses Dienstes zu erlauben.

Die 3 Standard-Protokollierungsmechanismen (`file`, `console`, `errorlog`) verwenden die `RegisterHandler`- und `UnregisterHandler`-Calls, so dass sie auf die gleiche Weise wie Ihre eigenen Handler aufgerufen werden.

Alle Fehler, die beim Schreiben in die Datei oder auf die Konsole auftreten, werden daher auch über den `Standard-Log`-Mechanismus gemeldet. Die Verwendung dieser Komponente ist sehr einfach. Das folgende kleine Programm ist ein vollständiger Webserver, der auch die `Debugcapture`-Ausgabe hat.

Es überschreibt 2 Methoden der Standardklasse `TCustomHTTPApplication`, um Logging zu ermöglichen und den Server zu konfigurieren:

```

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;
    
```

Hier haben wir noch nichts getan, außer unsere Klasse zu definieren und die Protokollierung zu implementieren. Die Überschreibung der `DoRun`-Methode ist der Ort, an dem die Magie geschieht: die Standardinstanz des `TDebugCaptureService` wird verwendet, um die Debug-Erfassungsfunktionalität bereitzustellen.

Sie wird so konfiguriert, dass die Debug-Ausgabe an die Konsole und in eine Datei namens `debug.log` gesendet wird, indem die Eigenschaften `LogToConsole` und `LogFileName` festgelegt werden:

```

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;
    
```

Nach der Registrierung der Route /debugcapture wird die Standardkomponente TSimpleFileModule verwendet, um die Standard-HTTP-Dateiübertragung aus dem aktuellen Verzeichnis bereitzustellen.

Beachten Sie, dass wir nicht den Standardmechanismus zur Protokollierung auf der Konsole verwenden, sondern einen eigenen Handler implementieren: HandleCaptureOutput, den wir mit dem Namen **Log** registrieren. *(die Namen für die internen Logging-Mechanismen beginnen alle mit \$, verwenden Sie dieses Zeichen nicht als Zeichen in Ihren eigenen Handlern)*

Die Methode HandleCaptureOutput verwendet die class method JSONDataToString, um eine Zeichenkette zu erstellen und protokolliert sie mit der Standardmethode DoLog der application class..

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

Dadurch werden die Debug-Informationen und die Informationen über bediente Seiten auf die gleiche einheitliche Weise angezeigt.

Damit ist die application class. fertig, sie muss nur noch gestartet werden:

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

So haben wir mit 20 Zeilen Code einen HTTP-Server geschaffen, der auch als Empfänger von Debug-Log-Informationen fungiert.

④ DER CLIENT-TEIL: TDEBUGCAPTURECLIENT

In **Pas2JS** stellt die Debugcapture Unit die Komponente TDebugCaptureClient zur Verfügung.

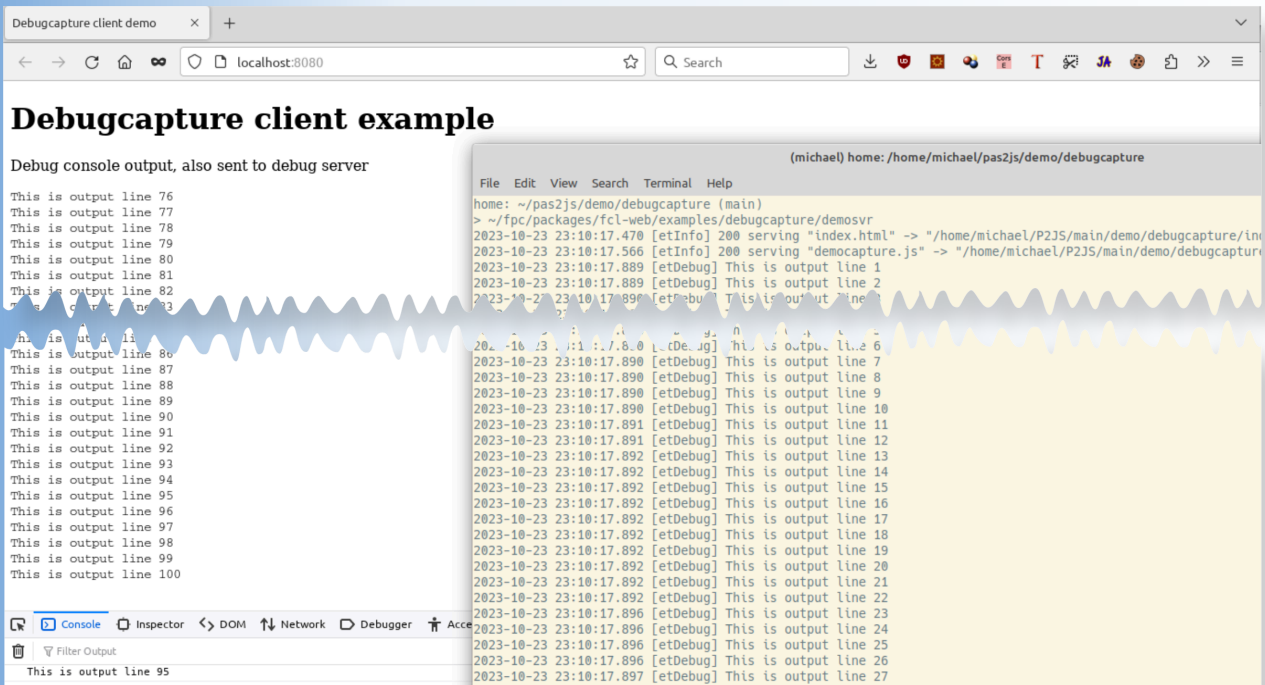
```
TDebugCaptureClient = class(TComponent)
Public
    Class property Instance : TDebugCaptureClient Read _Instance;
    Procedure Capture(const aLine : String; NewLine : Boolean = True); virtual;
    Procedure Flush;
    Procedure SetConsoleHook;
    Procedure ClearConsoleHook;
    Property URL : String;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

Es gibt die folgenden Methoden:

- **Capture**
Dies ist der zentrale Aufruf: Die Zeichenkette `aLine` wird an den Server gesendet. Wenn `NewLine` auf `True` gesetzt ist, wird ein Zeilenumbruchzeichen hinzugefügt.
- **Flush**
Wenn `BufferTimeout` auf eine positive Zahl gesetzt ist, werden die Zeilen gepuffert, bis die angegebene Zeitüberschreitung erreicht ist. Mit `Flush` wird der Puffer geleert und der Inhalt an den Server gesendet.
- **SetConsoleHook**
Beim Aufruf dieses Befehls wird der `ConsoleHook` installiert, was bedeutet, dass alle `Write(Ln)` Anweisungen auch in die „debug output“ geschrieben werden. Wenn ein vorheriger `ConsoleHook` vorhanden war, wird dieser ebenfalls aufgerufen.
- **ClearConsoleHook**
Setzt den `ConsoleHook` auf den Zustand vor dem Aufruf von `SetConsoleHook` zurück.
- **SetExceptionsHook**
Beim Aufruf dieser Funktion wird der `OnShowException`-Hook in `SysUtils` installiert, was bedeutet, dass alle Aufrufe von `Show-exceptions` auch in die „debug output“ geschrieben werden. Wenn ein vorheriger `ConsoleHook` vorhanden war, wird dieser ebenfalls aufgerufen.
- **ClearExceptionsHook**
Setzt den `Exceptions`-Hook auf den Zustand vor dem Aufruf von `SetExceptionsHook` zurück.

Darüber hinaus gibt es die folgenden Eigenschaften:

- **Instanz**
Diese Klasseneigenschaft stellt eine Standardinstanz zur Verfügung, die Sie konfigurieren und verwenden können.
- **URL**
Die URL, an die alle „debug output“ gesendet werden. Die Standard-URL ist `"/debugcapture"`.
- **BufferTimeout**
Eine Zeitspanne in Millisekunden, während der die Protokollausgabe lokal gepuffert wird, bevor sie an den Server gesendet wird. Bei einem Wert von 0 findet keine Pufferung statt, die gesamte Protokollierung wird sofort an den Server gesendet.
- **HookConsole**
Wenn auf `True` gesetzt, wird `SetConsoleHook` aufgerufen. Wenn auf `False` gesetzt, wird `ClearConsoleHook` aufgerufen.
- **HookExceptions**
Wenn auf `True` gesetzt, wird `SetExceptionsHook` aufgerufen. Wenn sie auf `False` gesetzt ist, wird `ClearExceptionsHook` aufgerufen.



Die Verwendung dieser Komponente ist wiederum recht einfach, wie das folgende Beispielprogramm zeigt:

```

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.

```

Das Ergebnis der beiden Programme zusammen ist in *Abbildung 1 auf Seite 6 dieses Artikels, Seite 96*, dargestellt. Im Hintergrund ist der Browser mit der Ausgabe der **WriteLn-Anweisungen** als HTML und in der **Debug-Console** des Browsers zu sehen. Im Vordergrund ist die Konsole zu sehen, auf der das HTTP-Serverprogramm gestartet wurde. Sie zeigt die URLs, die geladen wurden, und die **Debug-Capture-Ausgabe**.

5 SCHLUSSFOLGERUNG

Free Pascal und Pas2JS sind mit einfachen Werkzeugen ausgestattet, die es Ihnen ermöglichen, Debugging-Informationen von Produktanwendungen zu sammeln. Wie hier gezeigt wurde, ist der Code, um dies zu erreichen, wirklich einfach. Die Klassen, in diesem Prozess verwendet werden, können leicht mit zusätzlichen Funktionen erweitert werden: Sie können einen threaded Mechanismus auf dem Server hinzufügen, um die Leistung zu verbessern, Sie können die Protokolle in einer Datenbank speichern, sie an Logstash senden und das alles mit einem einzigen Mechanismus, der auch ohne zusätzlichen Code sofort funktioniert.



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

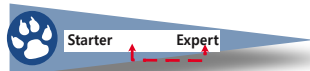
SUBSCRIPTION FOR 2 YEAR
BLAISE PASCAL MAGAZINE
€128,44 ex Vat

DEUTSCHSPRACHIGE AUSGABE : LAZARUS HANDBUCH

VORBESTELLUNG



blaisepascalmagazine.eu



ENDLICH IST ES DA: JETZT KÖNNEN WIR ES NUTZEN

KURZFASSUNG

Webassembly wurde für die Ausführung im Browser entwickelt. Sein Design ist auf Einfachheit und Sicherheit ausgerichtet, was es ideal für **Sandboxing** macht. Infolgedessen findet es mehr und mehr seinen Weg in Anwendungen, die außerhalb des Browsers laufen. In diesem Artikel zeigen wir, wie man ein Webassembly Modul in ein Free Pascal Modul einbettet.

Webassembly wurde in anderen Artikeln erwähnt
Heft 77 Seite 43 / Heft 83 /17 und Heft 101/82

1 EINFÜHRUNG

WebAssembly ist ein offenes Bytecode-Format, das ähnliche Zielsetzung wie die Java- und C#-Bytecode-Formate hat: Ein WebAssembly-Bytecode kann im Browser rechenintensive Aufgaben mit einer Geschwindigkeit ausführen, die der von einfachem Javascript weit überlegen ist.

Heute kann man aus jeder Programmiersprache (insbesondere **C**, **C++**, **Rust** und natürlich **Pascal**) in das **Webassembly**-Format kompilieren: Der **LLVM**-Compiler unterstützt **WebAssembly** als Ausgabeformat.

Die Spezifikation dieses Bytecode-Formats ist offen und wird vom **W3C**-Konsortium verwaltet:
<https://www.w3.org/TR/wasm-core-2/>.

Die Spezifikation wird auf github gepflegt:
<https://github.com/WebAssembly/design>

Neben der Kernspezifikation, die das grundlegende Bytecode-Format und die unterstützten **Assembler**-Anweisungen beschreibt, gibt es auch verschiedene Erweiterungen.

Eine Liste der Erweiterungen und ihrer verschiedenen Implementierungsstadien ist ebenfalls auf github zu finden:

<https://github.com/WebAssembly/proposals>

Einige der interessanteren Erweiterungen sind **Threading** und Exception Unterstützung.

Der offene Charakter des Formats bedeutet, dass jeder eine Laufzeitumgebung implementieren kann, die das Format lädt und ausführt. Tatsächlich gibt es außerhalb des Browsers mehrere Bytecode (wasm)-Ausführungsumgebungen:

WasmTime Diese Engine wird von der **Bytecode Alliance** gepflegt, die die Entwicklung von **WebAssembly** aktiv unterstützt.

Es handelt sich um eine konservative Implementierung. Das bedeutet, dass sie nur etablierte Vorschläge der **WebAssembly**-Spezifikation unterstützt.

<https://wasmtime.dev/>

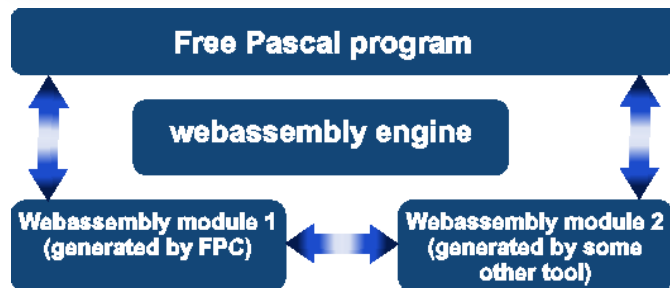
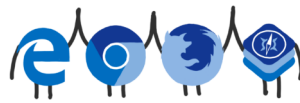


Abbildung 1: Verwendung einer Webassembly-Engine zur Ausführung eines WebAssembly-Programms in einem FPC-Programm





WasmEdge

Diese Engine wird von einer unabhängigen Gemeinschaft von Entwicklern gepflegt und wurde kürzlich unter das Dach der **Cloud Native Computing Foundation** gebracht <https://cncf.io/>, die ihrerseits Teil der **Linux**-Stiftung ist.

Diese Implementierung ist innovativer, sie unterstützt viele der experimentellen **WebAssembly**-Erweiterungen. <https://wasmedge.org/>

Wasmer ist eine unabhängige Implementierung einer **WebAssembly** Bytecode-Engine. Wie **WasmEdge** ist sie gegenüber neuen Vorschlägen aufgeschlossener: <https://wasmer.io/>

Es hat einen ähnlichen Ansatz wie **npm** (*ehemals Node Package Manager*) gewählt: Es verfügt über ein Paketsystem mit sofort einsetzbaren Webassembly-Modulen.

WAMR ist eine weitere **Bytecode-Allianz**-Implementierung der Webassembly-Laufzeit, die sich auf geringen Speicherbedarf und schnelle Ausführung konzentriert: <https://github.com/bytecodealliance/-micro-runtime>

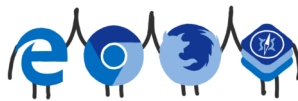
Alle diese Implementierungen verfügen über eine Bibliothek, die Sie verwenden können, um die Engine in Ihre Anwendung einzubetten: Das bedeutet, dass Sie ein **WebAssembly**-Programm (*das aus mehreren miteinander verknüpften WebAssembly-Modulen bestehen kann*) eingebettet in Ihrem **Free Pascal**-Programm ausführen können.

Dieses eingebettete **WebAssembly**-Programm kann auch von **Free Pascal** oder von einem anderen Programmierwerkzeug oder - höchstwahrscheinlich - einer Kombination aus beidem erzeugt werden:

- Alle Engines unterstützen das Verknüpfen verschiedener **WebAssembly**-Module, unabhängig davon, in welcher Sprache sie ursprünglich programmiert wurden.
- Das von **WebAssembly** verwendete Format stellt sicher, dass alle Module das gleiche Format für den Austausch von Daten und Code verwenden. Dies ist in *Abbildung 1 auf Seite 1* dieses Artikels schematisch dargestellt: ein natives **Free Pascal-Host Programm** lädt 2 **WebAssembly**-Module: eines in **Free Pascal** und eines in einer anderen Sprache geschrieben und kann Funktionen in beiden Modulen ausführen.
- **Die Module selbst können auch Funktionen im jeweils anderen Modul aufrufen.**

Von diesen Modulen haben die **Wasmtime**- und **WasmEdge** -Engines zumindest eine ausführliche Dokumentation, und deshalb wurden Import-Einheiten für diese Bibliotheken für **Free Pascal** erstellt, die wir hier demonstrieren werden.





2 DIE WASI-SPEZIFIKATION

Die **WebAssembly**-Spezifikation selbst legt nicht fest, wie mit der Umgebung zu interagieren ist: Das Format beschreibt **nicht**, wie man Dateien liest und schreibt, die Zeit abrufen und so weiter. Es wird lediglich ein Mechanismus beschrieben, wie Funktionen aus der Laufzeitumgebung importiert werden können. Natürlich wird auch beschrieben, wie Funktionen in der **Webassembly** ausgeführt werden können. Dies ermöglicht die Verketten von Modulen, genau wie bei dynamisch ladbaren Bibliotheken. Natürlich ist ein **Bytecode**-Format, das nicht mit der Umgebung interagieren kann, von geringem Nutzen. Daher wurde eine separate Spezifikation entwickelt, die eine minimale Liste von Funktionen enthält, die für die Interaktion mit der Außenwelt erforderlich sind, **WASI**: die **WebAssembly**-Systemschnittstelle.

<https://github.com/WebAssembly/WASI>

Alle oben genannten Engines unterstützen diese Schnittstelle. Das bedeutet, dass beim Laden eines **Webassembly**-Moduls in eine dieser Engines die in **WASI** aufgeführten Funktionen verfügbar sind. Sie dient als Grundlage für eine **LibC**-Implementierung, die in einer **Webassembly**-Umgebung läuft. Die aktuelle **WASI**-Spezifikation bietet nur grundlegende Unterstützung für die Interaktion mit dem Betriebssystem: nur grundlegende Dateieingabe/Ausgabe und das Abrufen der Zeit- und Umgebungsvariablen.

Das bedeutet **keine grafische Umgebung**, keine **TCP/IP**- oder **HTTP**-Umgebung usw.

(Letztere werden jedoch voraussichtlich in Version 2 der Spezifikation erscheinen).

Im Wesentlichen bietet die Spezifikation genügend Aufrufe, um die SysUtils Unit in Free Pascal zu implementieren und das ist es, was für die Entwicklung des Free Pascal WebAssembly Ziels verwendet wurde. Zusammengefasst bedeutet dies, dass grundlegende Free Pascal Programme in eine der oben erwähnten Engines geladen werden. Das heißt, dass die **WebAssembly-WASI-Spezifikation** alle Elemente enthält, die zur Implementierung von sysutils erforderlich sind.

Bedeutet dies, dass man keinen fortgeschrittenen Code (*der Sockets, UI etc. benötigt*) in diesen Runtimes laufenlassen kann?

Nein, natürlich nicht: die Engines unterstützen die Bereitstellung eigener Funktionen für das **Webassembly**. Das heißt, wenn Sie Funktionen zur Ausführung einer **HTTP**-Anfrage bereitstellen, können diese Funktionen innerhalb der Laufzeitumgebung ausgeführt werden, um **HTML**-Seiten herunterzuladen. Es sei darauf hingewiesen, dass Sie mit der Funktionalität, die Sie der **Webassembly** zur Verfügung stellen, vorsichtig sein müssen: Funktionen öffnen Türen zur Umgebung, die möglicherweise ausgenutzt werden können.

3 WASMTIME VERWENDEN

Wasmtime ist als Kommandozeilenwerkzeug verfügbar, mit dem Sie ein **Webassembly**-Modul von der Kommandozeile aus starten können. Dieses Kommandozeilen-Tool selbst ist einfach eine Shell um die dynamisch ladbare **Wasmtime**-Bibliothek.

Eine Anleitung zum Herunterladen und Installieren von wasmtime finden Sie hier

<https://docs.wasmtime.dev/cli-install.html>

Binärdateien der Versionen für alle wichtigen Plattformen finden Sie hier:

<https://github.com/bytecodealliance/wasmtime/releases>

Free Pascal enthält eine Unit namens `wasmtime`, die verwendet werden kann, um auf die Funktionalität der wasmtime Bibliothek zuzugreifen. Die Bibliothek wird zur Laufzeit mit dem Aufruf `LoadWasmTime` geladen:

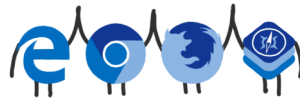
```
Procedure LoadWasmTime(const Lib : string);
```

Das Argument ist der Name der zu ladenden Bibliotheksdatei. Der Name der Bibliothek, die verteilt wird, ist in der Konstante `libwasmtime` verfügbar. Wenn das Laden fehlschlägt, wird eine `Exception` ausgelöst. Die Bibliothek stellt weit über 100 Typen und 500 Funktionen zur Verfügung, das ist deutlich mehr, als im Rahmen eines einzelnen Artikels erklärt werden kann.

Daher werden wir ein einfaches Beispiel beschreiben, das zeigt, wie eine Bibliothek geladen wird, wie eine vom Host bereitgestellte Funktion dem `wasm`-Modul zur Verfügung gestellt wird und wie diese Funktion aufgerufen wird.

Das **Webassembly**-Programm, das wir laden und ausführen werden, ist recht einfach:





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

Ohne auf die Einzelheiten des **Webassembly**-Textformats einzugehen, ist aus dem Text ersichtlich, dass dieses Modul eine Funktion namens `hello` (*ohne Parameter und Rückgabewert*)

importiert und eine Funktion namens `run` exportiert, die einfach die importierte Funktion "hello" aufruft.

Die `run`-Funktion hat wiederum keine Parameter und keinen Rückgabewert. Beachten Sie, dass keine **Datei-IO**- oder andere externe Funktionen verwendet werden: nur eine **importierte** Funktion und eine **exportierte** Funktion. Es gibt auch keinen Initialisierungs- oder Finalisierungscode. Um dieses **Webassembly**-Programm auszuführen, müssen wir also diese Datei laden, sie in Bytecode umwandeln, sie mit einer `Hallo`-Funktion versehen und dann die `Run`-Funktion aufrufen. Die Erwartung ist, dass die `Hallo`-Funktion aufgerufen wird und die `Run`-Funktion unmittelbar danach zurückkehrt. Das Hauptprogramm beginnt mit der Deklaration einer Vielzahl von Variablen:

```
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_func_type_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;
```

Die Bedeutung dieser Variablen wird erklärt, wenn wir sie im Programmcode finden.

Alle in **WasmTime** verwendeten Typen sind opaque record typen: die genauen Details des Datensatzes sind nicht offengelegt. Die meisten dieser Datensätze werden dynamisch mit einer Funktion erzeugt/erstellt, die einen Zeiger auf einen solchen opaque record Type zurückgibt, und in der Regel endet der Funktionsname auf (*oder enthält*). Wenn Sie mit einer bestimmten Variablen fertig sind, müssen Sie den von der Variable belegten Speicher mit einer Funktion freigeben, deren Name mit `Delete` endet

Das Programm beginnt natürlich mit dem Laden der `wasmtime`-Bibliothek. Wenn dies erfolgreich war, wird mit der Funktion `wasmtime engine new` eine **Webassembly-Engine** erstellt.

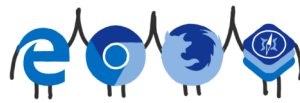
```
begin
  Writeln('Loading wasm library');
  Loadwasmtime('./'+libwasmtime);
  Writeln('Initializing...');
  engine := wasmtime_engine_new();
  store  := wasmtime_store_new(engine, nil, nil);
  context := wasmtime_store_context(store);
```

Der `Store` ist ein Allzweck-Speicherbereich für die Engine. Er kann verwendet werden, um Benutzerdaten hinzuzufügen, aber er wird auch von der Engine verwendet.

Der `Context` ist ein Zeiger, der von der Engine zum Hinzufügen/Entfernen von Daten zum Speicher verwendet wird.

Das folgende Codestück lädt eine Datei mit einem **Webassembly**-Modul in Textdarstellung (*eine Art Assembler*). Er weist einen Speicherbereich (`wat`) zu, indem er die `twasm_byte_vec_t_type` (*der einen Speicherblock darstellt*), der von der Engine benötigt wird, und verschiebt den Inhalt der Datei in diesen Bereich:





```
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;
```

Im folgenden Schritt wird die Textrepräsentation des **Webassembly**-Moduls mit mittels `wasmtime_wat2wasm` in Bytecode umgewandelt und in einem Speicherblock **wasm** gespeichert. (wiederum vom Typ `twasm_byte_vec` t). Die Textrepräsentation des Moduls (`wat`) wird entsorgt.

```
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);
```

Nach der Kompilierung der Webassembly wird der Bytecode mit der Funktion `wasmtime_module_new` in ein Modul (*Modul vom Typ* `Pwasmtime_module_t`) geladen und die Bytecode-Darstellung wird verworfen.

Das Modul wird bei der Ausführung der **Webassembly** verwendet.

Die Funktion `exit_with_error` ist eine Hilfsfunktion, die an mehreren Stellen im Programm verwendet wird. Wir werden später darauf zurückkommen.

An diesem Punkt haben wir ein Modul, das zur Ausführung bereit ist. Wir haben den Kontext, den wir am Anfang erstellt haben, noch nicht verwendet. Jetzt kommen wir zu dem Punkt, an dem dieser Kontext verwendet werden soll: Wir werden dem `webassembly`-Modul eine Pascal-'Hallo'-Funktion zur Verfügung stellen. Funktionen, die dem `webassembly`-Modul zur Verfügung gestellt werden, müssen den entsprechenden Funktionstyp haben

```
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;
```

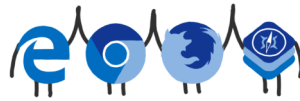
Das `env`-Argument kann verwendet werden, um Informationen weiterzugeben, zum Beispiel den `self`-Zeiger eines Objekts. Der Aufrufer enthält Informationen über die aufrufende Umgebung, und der `args`-Zeiger verweist auf die beim Aufruf übergebenen Argumente. Der Parameter `nargs` enthält die Anzahl der Argumente. In ähnlicher Weise werden die Argumente `results` und `nresults` zur Angabe von Rückgabewerten verwendet.

Der Rückgabewert ist ein `Trap` (vom Typ `Pwasm_trap_t`): Wenn er nicht null ist, signalisiert er der **Webassembly**-Engine einen Fehlerzustand. Mit diesem Wissen sieht unsere "Hello"-Funktion wie folgt aus:

```
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;
```





Der nächste Teil unseres Programms besteht darin, diese Funktion im **Webassembly**-Modul zu definieren, damit sie aufgerufen werden kann. Dies beginnt mit der Erstellung eines Funktionstyps (`hello_ty`, vom Typ `Pwasm_func_t`), der dann mit `wasmtime_func_new` als Funktion registriert wird.

Ein Funktionstyp wird durch `Pwasm_func_t` repräsentiert. Dies entspricht einem prozeduralen Typ in **Pascal**. Das **WebAssembly**-Format definiert einen Funktionstyp für alle Funktionen und Prozeduren.

Sowohl für interne als auch für externe Funktionen muss ein Funktionstyp definiert werden.

Für externe (importierte/exportierte) Funktionen ist dies logisch: Die Laufzeit-Engine muss wissen, welche Daten sie bereitstellen oder welches Datum sie extrahieren muss, wenn die Grenze zwischen

Webassembly und der Host-Umgebung überschritten wird: sowohl beim Aufruf einer **Webassembly**-Funktion in einem **Webassembly**-Modul als auch wenn eine externe Funktion durch das **Webassembly**-Modul aufgerufen wird.

Um eine aufrufbare Funktion zu registrieren, wird die Funktion `wasmtime_func_new` verwendet:

```
procedure wasmtime_func_new (
    store: Pwasmtime_context_t;
    _type: Pwasm_func_t;
    callback: Twasmtime_func_callback_t;
    env: pointer;
    finalizer: Tfinalizer;
    ret: Pwasmtime_func_t)
```

Das zweite Argument (`_type`) ist der Funktionstyp, und das dritte (`callback`) ist die eigentliche Funktion, die aufgerufen werden soll.

Das `env`-Argument kann mit einem beliebigen Wert gefüllt werden, es wird beim Aufruf des `Callbacks` unverändert übergeben. Dies kann zum Beispiel zum Speichern eines Objektzeigers verwendet werden.

Schließlich kann ein `Finalizer` für `Env` angegeben werden, eine Funktion, die normalerweise aufgerufen wird, um das `env`-Objekt freizugeben, wenn das **Webassembly**-Modul zerstört wird.

Das Argument `ret` verweist auf einen Ort, der mit einer Funktionsdefinition gefüllt wird.

Da unsere Funktion keine Argumente akzeptiert und keine Ergebnisse zurückgibt, ist der Funktionstyp und die Registrierung des Rückrufs recht einfach:

```
Writeln('Creating callback...\n');
hello_ty:=wasm_func_new_0_0();
wasmtime_func_new(context, hello_ty, @hello_callback, Nil, Nil, @hello);
```

BEACHTEN Sie das Kontextargument und die Variable `hello`, die die von der **Wasm**-Laufzeit erstellte Funktionsdefinition enthalten wird.

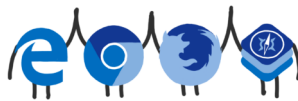
Was wir bisher getan haben, ist die Definition des **Webassembly**-Moduls und der Funktionen, die wir ihm zur Verfügung stellen werden. Es ist bereit zur Ausführung. Um ein **Webassembly** tatsächlich auszuführen, müssen wir eine Instanz des Moduls erstellen (*es ist möglich, mehrere Instanzen eines einzelnen Moduls zu erstellen und sie parallel auszuführen*).

Aus dieser Instanz können wir dann die Adresse der exportierten Funktion (`'run'`) extrahieren und sie aufrufen.

Um eine Instanz eines **Webassembly**-Moduls zu erstellen, wird die Funktion `wasmtime_instance_new` verwendet

```
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
```





Der Store ist der Kontext, den wir verwenden, das Modul ist das Modul, das wir gerade definiert haben. Wie aus dem Argument `imports` ersichtlich ist, müssen wir ihm alle Funktionen zur Verfügung stellen, die verwendet werden können.

Wenn mehrere Instanzen erstellt und ausgeführt werden können, ist es sinnvoll, dass die Funktionen der Instanz und nicht dem Modul zur Verfügung gestellt werden:

der `env`-Zeiger für die aufrufbaren Funktionen ist **in der Regel für jede Instanz unterschiedlich**.

Bei erfolgreicher Rückkehr wird `instance` mit einer lauffähigen Instanz gefüllt. `trap` wird mit einem Fehlerbericht gefüllt, wenn ein Fehler aufgetreten ist.

Fehler können zum Beispiel auftreten, wenn das Modul erwartet, die Funktionen `foo` und `bar` importieren zu können, aber nur `bar` geliefert wird.

In unserem Fall müssen wir die Funktion `hello` bereitstellen, die wir gerade erstellt haben:

```
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);
```

Die Argumente für die Funktion `wasmtime_instance_new` sind der Kontext, das Modul, **1 Importdefinition** und **2 Variablen** für die Rückgabewerte `error` und `trap`, die wir bei der Rückkehr untersuchen.

Die Funktion, die aus dem **Webassembly**-Modul exportiert wird, heißt "run".

Wir extrahieren die Funktionsdefinition aus der Instanz mit der Methode `wasmtime_instance_export_get` Funktion:

```
function wasmtime_instance_export_get(
    store      : Pwasmtime_context_t;
    instance   : Pwasmtime_instance_t;
    name       : PAnsiChar;
    name_len   : Tsize_t;
    item       : Pwasmtime_extern_t):T_Boolean;
```

Die Argumente `store` und `instance` sind natürlich der von uns verwendete Kontext und die Instanz, die wir gerade erstellt haben. Die Funktionen `name` und `name_len` werden verwendet, um den Namen der gewünschten Funktion ("run" in unserem Fall) und das Element wird mit die Funktionsdefinition bei der Rückgabe: Wenn die Funktion existiert, gibt die Funktion einen Rückgabewert ungleich Null.

Unser Code, um die Definition der Funktion "run" zu erhalten, lautet also:

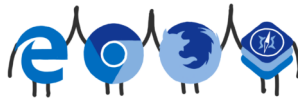
```
Writeln('Extracting export...\n');
ok:=wasmtime_instance_export_get(context, @instance, PAnsiChar('run'), 3,
@run);
if OK=0 then
    exit_with_error('failed to get run export', nil, nil);
if run.kind<>WASMTIME_EXTERN_FUNC then
    exit_with_error('run is not a function', nil, nil);
```

Die Variable `run` enthält einen Verweis auf die exportierte Funktion.

Jetzt können wir die Funktion tatsächlich ausführen.

Dies geschieht mit dem Aufruf `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;
```

Die ersten beiden Argumente sind der Speicherkontext und die Funktionsdefinition, die wir gerade extrahiert haben. Beachten Sie, dass das **Wasm-Modul** oder die **Wasm-Instanz** nicht angegeben werden müssen:

Sie sind implizit in der Funktionsdefinition enthalten. Die Argumente, die der aufgerufenen Funktion übergeben werden und die von ihr zurückgegebenen Ergebnisse werden in den nächsten 4 Argumenten angegeben. Das letzte Argument (`trap`) wird verwendet, um eine Fehlerbedingung festzuhalten, wenn etwas schief läuft. Da die Funktion 'run' keine Argumente entgegennimmt und keine Ergebnisse liefert, müssen wir nichts einrichten, um sie zu spezifizieren, so dass wir bereit sind, unsere Funktion aufzurufen:

```
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);
```

Als erstes wird geprüft, ob ein Fehler zurückgegeben wurde.

Nach all dem wurde die Funktion "run" aufgerufen, und die Instanz und das Modul können bereinigt werden.

Zum Bereinigen werden das Modul und der Kontext des Speichers bereinigt: alles, was mit dem Speicher verbunden ist, wird ebenfalls bereinigt:

```
Writeln('All finished!\n');
wasmtime_module_delete(module);
wasmtime_store_delete(store);
wasm_engine_delete(engine);
end.
```

Nun muss nur noch die Prozedur `exit_with_error` gezeigt werden:

Diese Prozedur zeigt die von der **Wasmtime-Engine** zurückgegebenen Fehlerinformationen und demonstriert, dass Sie die `Trap`- und `Error`-Laufzeitfehlerberichte freigeben müssen, wenn sie auftreten. Andernfalls kommt es zu **Speicherlecks**:

```
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;
```





Jetzt können wir das Binärprogramm ausführen. Wenn alles gut geht, bekommen Sie eine Ausgabe ähnlich der in *Abbildung 2 auf Seite 10* dargestellten.

```
(michael) home: /home/michael/source/articles/embedding/wasmtime - [x]
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
>
```

Abbildung 2: Das erste wasmtime Beispielprogramm in Aktion

④ BEREITSTELLUNG EINER WASI-UMGEBUNG UM EIN FPC-GENERIERTES PROGRAMM AUSZUFÜHREN

Das vorherige Demonstrationsprogramm benutzte nur eine importierte Funktion (`hallo`) und eine exportierte Funktion (`run`), um mit der Außenwelt zu kommunizieren. Es nutzte insbesondere keine **WASI-Funktionalität**. Die **Webassembly-RTL** von **Free Pascal** verwendet die **WASI-Funktionalität**. **wasmtime** stellt die **WASI-Schnittstelle** einem **Webassembly-Modul** nur dann zur Verfügung, wenn Sie es dazu anweisen. Da die **FPC-RTL** für **Webassembly** auf die **WASI-Schnittstelle** angewiesen ist, werden wir ein von **FPC** generiertes Programm ausführen, um zu demonstrieren, wie die **WASI-Funktionalität** einem **Webassembly-Modul** zur Verfügung gestellt werden kann. Das **FPC-Programm** ist ein sehr einfaches 'Hello, world' :

```
begin
  Writeln("Hello, World!" from FPC webassembly');
end.
```

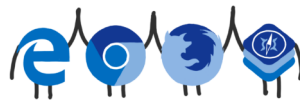
Wenn der **Free Pascal Webassembly** Compiler und RTL installiert sind, dann kann dieses Programm so kompiliert werden:

```
ppcrosswasm32 hello.pp
```

Wenn alles gut gegangen ist, entsteht ein **Webassembly-Modul** `hello.wasm`. Das folgende **Free Pascal**-Programm lädt das **Webassembly-Modul** und stellt die **WASI-Umgebung** bereit. Der Variablendeklarationsblock ähnelt dem des vorherigen Beispiels sehr, wir führen nur die zusätzlichen Variablen auf, die im vorherigen Programm nicht vorhanden waren:

```
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);
```





Hier erstellen wir einen **Webassembly**-Linker und verwenden ihn, um die **WASI**-Funktionalität mit unserem **Webassembly**-Modul zu verknüpfen: Die Funktion `wasmtime_linker_define_wasi` macht die **WASI**-Standardfunktionen im **Webassembly**-Modul verfügbar. Die **WASI**-Funktionen müssen jedoch konfiguriert werden: Welche Dateisystemverzeichnisse sind verfügbar, wie lauten die Umgebungsvariablen, Befehlszeilenparameter? Was soll mit den Dateideskriptoren für die Standardeingabe, -ausgabe und -fehlerausgabe geschehen? All dies kann durch die Erstellung einer **WASI**-Konfiguration unter Verwendung der Funktion `wasi_config_new` festgelegt werden:

```
wasi_config:=wasi_config_new();
if (wasi_config=nil) then
  exit_with_error('failed to create wasi config', Nil, nil);
```

Die **wasi-Konfiguration** muss mit einer oder mehreren Funktionen konfiguriert werden:

wasi_config_set_argv

Setzt Werte für die Kommandozeilenparameter des Moduls `wasm`.

wasi_config_inherit_argv

Verwendet die Werte des Host-Programms für die Kommandozeilenparameter des `wasm`-Moduls.

wasi_config_set_env

Setzt die Werte für die Umgebungsvariablen des Moduls `wasm`.

wasi_config_inherit_env

Verwendet die Werte der Umgebungsvariablen des Hostprogramms für das `wasm`-Modul.

wasi_config_set_stdin_file

Gibt eine Datei an, die als Standardeingabe für das **Webassembly**-Programm verwendet werden soll.

wasi_config_set_stdin_bytes

Gibt einen Speicherblock an, der als Standardeingabe für das **Webassembly**-Programm verwendet werden soll.

wasi_config_inherit_stdin

Setzt die Standardeingabe des Hostprogramms als Standardeingabe für das **Webassembly**-Programm.

wasi_config_set_stdout_file

Gibt eine Datei an, die als Standardausgabe für das **Webassembly**-Programm verwendet werden soll.

wasi_config_inherit_stdout

Setzt die Standardausgabe des Hostprogramms als Standardausgabe für das **Webassembly**-Programm.

wasi_config_set_stderr_file

Gibt eine Datei an, die als Standardfehlerausgabe für das **Webassembly**-Programm verwendet werden soll.

wasi_config_inherit_stderr

Setzt die Standardfehlerausgabe des Hostprogramms als Standardfehlerausgabe für das **Webassembly**-Programm.

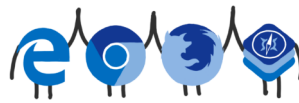
wasi_config_preopen_dir

Konfiguriert ein "vorgeöffnetes Verzeichnis" als Basisverzeichnis für **WASI**-Datei-APIs.

Für unsere einfache Demonstration erben wir einfach alles von der Host-Umgebung, und stellen das aktuelle Verzeichnis zur Verfügung:

```
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);
```





Die Funktion `wasmtime context_set_wasi` koppelt die **WASI**-Konfiguration an die **WASI**-Umgebung des **Webassembly**-Moduls.

Jetzt können wir das Modul laden und ausführen.

Das Laden des **Webassembly**-Moduls unterscheidet sich etwas von unserem vorherigen Programm: Anstatt ein **Webassembly**-Textformat zu laden und es zu kompilieren, laden wir ein bereits kompiliertes `.wasm`-Modul:

```
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);
```

Dieses Mal verwenden wir den **Webassembly-Linker**, um das Modul zu instanziiieren, da der Linker die **WASI-Funktionalität** für das **Webassembly-Modul** bereitstellen muss.

Die Funktion die dies tut, heißt `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;
```

Der Name des Moduls kann in den Variablen `name` und `name_len` angegeben werden.

Wir verwenden sie hier nicht. Sie werden nur benötigt, wenn verschiedene Module miteinander verknüpft werden müssen, da der Linker Importe eines Moduls mit Exporten eines anderen Moduls unter Verwendung des Modulnamens verknüpft.

Da wir nur ein Modul laden, ist es nicht notwendig, einen Namen anzugeben:

```
// 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);
```

Ein Modul kann eine exportierte Standardfunktion haben. Dies ist das `'_start'` Symbol, das das **Free Pascal Programm** startet. Wir extrahieren den Wert dieser Funktion mit `wasmtime_linker_get_default` und rufen sie auf, um das von **FPC** generierte Programm zu starten:

```
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);
```





Die `exit_proc`-Routine in der **WASI-Spezifikation** beendet das **Webassembly-Programm**. Die **Free Pascal-Laufzeitumgebung** für **Webassembly** ruft sie auf, wenn das Programm angehalten wird. "In `WasmTime*`, löst die `exit_proc`-Routine einen Fehler aus, um das Programm anzuhalten, was seltsamerweise das Ergebnis der Ausführung der Startfunktion ist, also eine Fehlerbedingung! Glücklicherweise kann die Funktion `wasmtime_error_exit_status` verwendet werden, um diesen speziellen Fall zu überprüfen. Wenn das Programm beendet ist, muss es nur noch aufgeräumt werden, genau wie im vorherigen Beispielprogramm:

***Ergänzender Kommentar von Michael van Canneyt.**

Die Ausnahme wird ausgelöst, wenn das Programm angehalten (geschlossen) wird und die Startprozedur noch läuft. Die Startprozedur läuft natürlich immer, da sie der Einstiegspunkt des Programms ist...Das ist eine Eigenart von `wasmtime` (sie könnten dies auch einfach intern behandeln)

```
// Clean up after ourselves at this point
Writeln('All finished!\n');
wasmtime_module_delete(module);
wasmtime_store_delete(store);
wasm_engine_delete(engine);
end.
```

```
(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
> |
```

Abbildung 3: Der Free Pascal WASI-basierte RTL in Aktion

5 WASMEDGE VERWENDEN

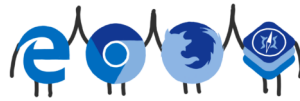
Eine zweite Bibliothek, die zum Einbetten von **WebAssembly**-Programmen verwendet werden kann, ist `wasmedge`. Installationsanweisungen finden Sie unter <https://wasmedge.org/docs/start/install/>

Die Unit, die diese Bibliothek importiert, heißt **libwasmedge**.

Die Bibliothek funktioniert weitgehend ähnlich wie die `wasmtime`-Bibliothek, unterscheidet sich aber in Details. In mancher Hinsicht ist sie einfacher als die `wasmtime`-Bibliothek.

Sie stellt nur 300 Funktionen zur Verfügung - immer noch eine beträchtliche Anzahl, aber weniger als die `wasmtime`-Bibliothek.

Das Beispielprogramm, das wir demonstrieren werden, lädt und führt die folgende **Webassembly**-Funktion aus, die die **Fibonacci-Reihe** berechnet:



```
(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
  )
)
```

Ohne auf die Einzelheiten des **Webassembly-Formats** einzugehen, können Sie in der zweiten Zeile sehen, dass eine Funktion "fib" definiert wird, die eine 32-Bit-Ganzzahl als Parameter akzeptiert

Das Programm zum Aufrufen dieser Funktion ist relativ einfach:

```
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  : pchar;
begin
  Writeln("Loading library...");
  Loadlibwasmedge('./'+libwasmedge);
  ConfCxt:=Wasmedge_ConfigureCreate();
  Writeln("Adding WASI environment...");
  Wasmedge_ConfigureAddHostRegistration(ConfCxt, Wasmedge_
                                         HostRegistration_Wasi);

  Writeln("Creating engine...");
  VMCxt:=Wasmedge_VMCreate(ConfCxt,Nil);
```

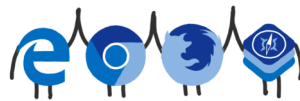
Nach dem Laden der Bibliothek wird ein Konfigurationskontext mit `Wasmedge_ConfigureCreate` erstellt. Die **WASI**-Umgebung wird der Konfiguration mit der Routine `Wasmedge_configureAddHostRegistration`-Routine hinzugefügt.

In diesem Kontext wird eine "virtuelle Maschine" erstellt, auf der das **Webassembly** Modul ausführt.

Die Funktion `Wasmedge_VMCreate` wird zur Erstellung dieser virtuellen Maschine verwendet:

```
function Wasmedge_VMCreate(
  ConfCxt: PWasmedge_ConfigureContext;
  StoreCxt: PWasmedge_StoreContext):PWasmedge_VMContext;
```





Die Parameter sind die Konfiguration und ein Speicher (*ähnlich dem, der in wasmtime verwendet wird*). Der Speicher wird für dieses Beispiel nicht benötigt.

Um die **Fibonacci**-Funktion aufzurufen, wird ein Parameter benötigt. Dieser Parameter wird erzeugt durch der Funktion `WasmEdge_ValueGenI32` erzeugt. Webassembly kennt nur wenige Grundtypen (`Integer`, `Float`), so dass die Anzahl der Funktionen, die Sie zur Erzeugung von Werten verwenden müssen, begrenzt ist: Es gibt nur 8 Funktionen, von denen Sie in der Praxis 4 verwenden werden.

```
Params[0] := WasmEdge_ValueGenI32(32);
FuncName := WasmEdge_StringCreateByCString(Pcchar(PAnsiChar('fib')));
```

Die zweite Zeile erzeugt eine Zeichenkette 'fib', die von der **wasmedge**-Bibliothek verwendet werden kann. Diese String wird zum Aufruf von **Fibonacci** verwendet.

Der Aufruf einer Funktion in einem **Webassembly-Modul** kann in einem einzigen Aufruf mit der Komfortfunktion `WasmEdge_VMRunWasmFromFile` erfolgen:

```
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;
```

Die Argumente für diese Funktion sind ziemlich einfach: **path** ist der Dateiname des zu ladenden Moduls. **FuncName** ist die zu ladende Funktion, **Params** und **ParamLen** geben die Parameter an, die an die Funktion übergeben werden müssen, und **Returns** und **ReturnLen** geben den Ort an, an dem die Rückgabewerte der Funktion gespeichert werden müssen. Im Falle der **Fibonacci**-Funktion wird die Funktion wie folgt verwendet:

```
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
> |
```

Abbildung 4: Die 32. Fibonacci-Zahl, berechnet mit einem Webassembly-Programm.





Nach der Überprüfung des Ergebnisses der Funktion 'run' mit `WasmEdge_ResultOK` wird der Rückgabewert aus dem ersten Element des Arrays `Returns` abgerufen. `WasmEdge_ValueGetI32` ist eine der acht Funktionen, die verwendet werden können, um einen Webassembly-Rückgabewert in einen nativen Pascal-Wert zu konvertieren, in diesem Fall eine 32-Bit-Ganzzahl. Nun müssen nur noch die verschiedenen zugewiesenen Ressourcen aufgeräumt werden:

```
Writeln('Cleaning up...');
WasmEdge_VMDelete(VMCxt);
WasmEdge_ConfigureDelete(ConfCxt);
WasmEdge_StringDelete(FuncName);
end.
```

Das Ergebnis dieses Programms ist in Abbildung 4 auf Seite 14 dieses Artikels zu sehen.

⑥ EIN MIT FPC GENERIERTES PROGRAMM MIT WASMEDGE EINBETTEN.

Das Einbetten eines mit FPC generierten Webassembly-Moduls unterscheidet sich nicht so sehr von dem vorherigen Programm. Der Start des Programms ist ähnlich, die Unterschiede liegen in der Art und Weise, wie die virtuelle **Webassembly-Maschine** erstellt wird.

```
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     : ppcchar;
  WasiModule  : PWasmEdge_ModuleInstanceContext;
  ModName     : TWasmEdge_String;
begin
  Writeln('Loading library...');
  Loadlibwasmedge('./'+libwasmedge);
  Writeln('Adding WASI environment...');
  ConfCxt:=WasmEdge_ConfigureCreate();
  WasmEdge_ConfigureAddHostRegistration(ConfCxt, WasmEdge_HostRegistration_Wasi);
  Writeln('Creating engine...');
  VMCxt:=WasmEdge_VMCreate(ConfCxt,Nil);
```

Bis hierhin gibt es keinen Unterschied. Wie im Fall der **Wasmtime**-Bibliothek besteht der nächste Schritt darin, eine Instanz des **WASI-Moduls** abzurufen und es zu konfigurieren.

Dies geschieht mit den `WasmEdge_VMGetImportModuleContext` und `WasmEdge_ModuleInstanceInitWASI`-Funktionen.

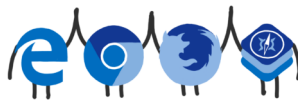
Die erste dieser beiden Funktionen gibt den Modulkontext des vordefinierten **WASI_moduls** zurück: Dieses vordefinierte Modul wurde mit der Funktion `WasmEdge_ConfigureAddHostRegistration` aktiviert. Bis hierhin gibt es keinen Unterschied.

Wie im Fall der **Wasmtime-Bibliothek** besteht der nächste Schritt darin, eine Instanz des **WASI-Moduls** abzurufen und es zu konfigurieren.

Dies geschieht mit den Befehlen `WasmEdge_VMGetImportModuleContext` und `WasmEdge_ModuleInstanceInitWASI`-Funktionen.

Die erste dieser beiden Funktionen gibt den Modulkontext des vordefinierten **WASI_moduls** zurück: Dieses vordefinierte Modul wurde mit der Funktion `WasmEdge_ConfigureAddHostRegistration` aktiviert und muss konfiguriert werden

```
procedure WasmEdge_ModuleInstanceInitWASI(
  Cxt: PWasmEdge_ModuleInstanceContext;
  Args: Ppcchar; ArgLen: Tuint32_t;
  Envs: Ppcchar; EnvLen: Tuint32_t;
  Preopens: Ppcchar; PreopenLen: Tuint32_t);
```



Wie Sie sehen, können die Liste der Befehlszeilenparameter, Umgebungsvariablen und erlaubte Verzeichnisse für den Dateizugriff konfiguriert werden. Standard-**Eingabe/Ausgabe/Fehler** können nicht wie bei **wasmtime** konfiguriert werden. In unserem Fall werden weder Befehlszeilenparameter noch Umgebungsvariablen benötigt, so dass die Konfiguration recht einfach ist:

```
WasiModule:=WasmEdge_VMGetImportModuleContext(VMCxt,WasmEdge_
HostRegistration_Wasi);
WasmEdge_ModuleInstanceInitWASI(WasiModule,Nil,0,Nil,0,Nil,0);
```

Damit können wir unser **Webassembly-Modul** laden und die Funktion ausführen. Wir können dies nicht mit der Funktion `WasmEdge_VMRunWasmFromFile` tun, sondern müssen stattdessen das Modul mit der Funktion `WasmEdge_VMRegisterModuleFromFile` laden. Damit können wir unser Webassembly-Modul laden und die Funktion ausführen.

```
function WasmEdge_VMRegisterModuleFromFile(Cxt: PWasmEdge_VMContext;
ModuleName: TWasmEdge_String; Path: pchar): TWasmEdge_Result;
```

Der Modulname muss angegeben werden und muss eindeutig sein. Wenn mehrere Module geladen werden, verknüpft die Engine die Module anhand ihres Namens.

Wenn **Modul** 'a' die Funktion 'b.proc1' importieren muss, dann muss Name 'b' übergeben werden, wenn das **webassembly-Modul** geladen wird, welches die Prozedur 'proc1' enthält. Module haben keinen mit ihnen verbundenen Namen und der Dateiname steht in keinem Zusammenhang mit dem Modulnamen.

Da der Modulname als `TWasmEdge_String`-Typ weitergegeben wird, müssen wir ihn mit `WasmEdge_StringCreateByCString` zuweisen, bevor wir das Modul laden:

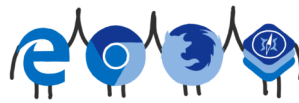
```
ModName:=WasmEdge_StringCreateByCString(Pcchar('prog'));
pmodule:=pchar(PAnsiChar('hello.wasm'));
Res:=WasmEdge_VMRegisterModuleFromFile(VMCxt, modname, pmodule);
if (WasmEdge_ResultOK(Res)) then
  Writeln('Loaded OK')
else
  Writeln('Error message: ', PAnsiChar(WasmEdge_ResultGetMessage(Res)));
```

Nun, da das Modul geladen ist, können wir die Startfunktion tatsächlich ausführen:

```
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)));
```

HINWEIS: Wir haben den Exit-Code des **FPC-Programms** mit der Funktion `WasmEdge_ModuleInstanceWASIGetExitCode` abgefragt. Im Unterschied zu **Wasmtime** verwendet die **Wasmedge**-Bibliothek keinen `Trap`, um den `Exit-Code` zu setzen. Damit bleibt nur noch das Aufräumen, ähnlich wie im vorherigen Beispielprogramm:





```
Writeln('Cleaning up...');  
WasmEdge_VMDelete(VMCxt);  
WasmEdge_ConfigureDelete(ConfCxt);  
WasmEdge_StringDelete(FuncName);  
end.
```

end.

Damit kann das Programm getestet werden, und die Ausgabe sollte wie in Abbildung 5 auf Seite 17 aussehen

```
(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  
> |
```

Abbildung 5: Wasmedge nutzen, um ein durch FPC generiertes WebAssembly-Module in einem nativen FPC-Programm laufen zu lassen

🔗 SCHLUSSEFOLGERUNG

In den vorangegangenen Artikeln haben wir gezeigt, dass Free Pascal verwendet werden kann, um Webassembly Module zu erzeugen. Und wie in diesem Artikel gezeigt, können native FPC-Programme mit Hilfe einiger externer Bibliotheken Programme Webassembly-Module laden, unabhängig davon, ob sie **VON** FPC oder von einem anderen Werkzeug.



Von Helmut Elsner



*1 USABILITY
*2 GUI
GRAPHICAL USER INTERFACE

EINLEITUNG

Usability ist eine besonders von Hobbyprogrammierern, aber auch leider oft bei Profis, zu wenig beachtete Eigenschaft von selbstgeschriebenen Programmen.

Der Programmierer weiß was das Programm tut, welche Möglichkeiten und Funktionalität es hat, aber weiß das auch der potenzielle Nutzer des Programms? Was für den Programmierer selbstverständlich ist, kann den Benutzer vor unlösbare Probleme stellen oder im besten Falle nur für Unverständnis sorgen.

Dem kann aber mit Hilfe der Lazarus IDE leicht abgeholfen werden.

Nicht behandelt werden in diesem Beitrag Skalierung und DPI-Anpassung. Das wäre ein eigenes Thema.

WAS MACHT USABILITY AUS?

Ohne Anspruch auf Vollständigkeit zu erheben, hier meine Gedanken dazu:

- Übersichtlichkeit
- Rückmeldung
- Struktur
- Konsistenz
- Unterstützung
- Hilfe



1 ÜBERSICHTLICHKEIT

Der Benutzer sollte nach dem Start des Programms sofort sehen, was die Hauptfunktionen des Programms sind und wie er sie erreicht. Hauptfunktionen oder sehr häufig benutzte Funktionen sollten prominent auf der Programmoberfläche angeordnet und als solche erkennbar sein. Dazu muss man sich die Priorisierung von Funktionalität bei der Bedienung des Programms vor der Erstellung der **GUI** überlegen. Hauptfunktionen kann man zum Beispiel mit `TSpeedButton` besser sichtbar machen. Macht man für alle Funktionen einen `SpeedButton`, dann ist dieser Effekt aber wieder verpufft.

Wenn es möglich ist, sollte der Workflow von links nach rechts bzw. von oben nach unten an der Oberfläche sichtbar sein. Dies erreicht man, indem man die Bedienelemente entsprechend anordnet und gruppiert (→**Struktur**). Gern vernachlässigt sind die Komponenten-Eigenschaften `TabStop` (`true` / `false`) und `TabOrder`. Die Reihenfolge `TabOrder` sollte dem Workflow folgen. Dies erfordert nur etwas Fleißarbeit.

Nicht zu viele Bedienelemente auf der Oberfläche anzeigen. Man sollte temporär nicht nutzbare Bedienelemente auf `Enabled:=false` setzen und `Visible:=false` nur verwenden, wenn es wirklich nötig ist. Es ist für den Benutzer meist besser, zu sehen, dass da etwas ist, was er aber im Moment nicht benutzen kann, als gar nichts zu sehen.

2 RÜCKMELDUNG

Der Programmierer sollte dem Benutzer immer das Gefühl geben, dass das Programm auf ihn sofort reagiert.

Wenn irgendetwas länger dauert, muss der Benutzer darüber informiert werden und es muss irgendetwas angezeigt werden, was Bewegung in der Sache vorspiegelt. Hier bietet sich `TProgressBar` aus `CommonControls` an. Auf jeden Fall aber sollte der Cursor dem Kontext angepasst werden. Bei länger andauernden Prozessen sollte zumindest der Cursor zeigen, dass da etwas läuft.





```

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;
    
```

Für kurze Textausgaben und Statusmeldungen gibt es `TStatusBar`. Diese sollen dem Nutzer stets verständlich mitteilen, was gerade passiert.
 Bei komplexeren Programmen kann man Hinweise, Status- und Fehlermeldungen auch in einem Textfeld (zum Beispiel `TMemo`) sammeln und bei Bedarf sichtbar und abspeicherbar machen.

③ STRUKTUR

Wie oben schon gesagt, sollte der Workflow auf der Programmoberfläche sichtbar sein (z.B. Datei laden → Datei bearbeiten → Datei speichern - von links nach rechts angeordnet).
 Zusammengehörige Bedienelemente sollten sichtbar gruppiert werden. Dazu gibt es `TGroupBox` bei den Standard-Komponenten.

Bei vielen Bedienelementen sollte man diese nach Funktionalität gruppieren. Dazu kann man mehrere Fenster verwenden, aber auch da sollte man den Benutzer nicht überfordern. Ich finde es besser, mit `TPageControl` und diesem untergeordneten `TTabSheet` zu arbeiten. Das ermöglicht trotz vieler Bedienmöglichkeiten eine übersichtliche und strukturierte Bedienoberfläche. Man kann zum Beispiel alle Einstellungen auf eine oder mehrere `TabSheets` legen. So sind die Einstellungen leicht zu erreichen, überfrachten aber die normale Bedienoberfläche nicht. Außerdem kann man mit `TabSheets` allgemeingültige und funktionspezifische Bedienelemente getrennt anordnen.

Allzu bunte und durch (Hintergrund-) Bilder verwirrende Bedienoberflächen sollte man vermeiden.

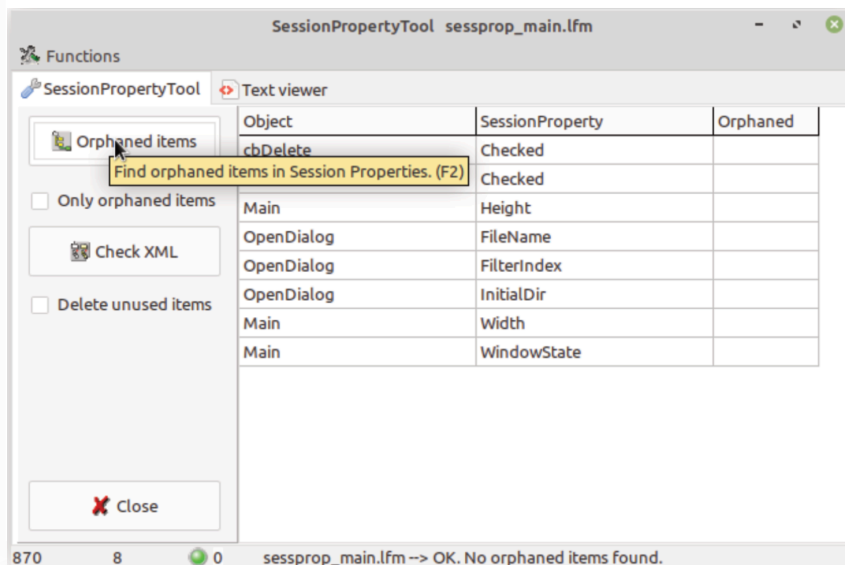


Figure 1: Screenshot from example project with two `TabSheets` to separate functionality.





4 KONSISTENZ

Die Bedienoberfläche sollte in sich konsistent sein, das heißt gleiche Funktionen sollten gleiche Namen bzw. Namensstämme, gleiche Bildchen (**Glyphs**) haben und gleich behandelt werden. Ein wesentliches Hilfsmittel Konsistenz zu erlangen und den Entwicklungs- und Wartungsaufwand zu reduzieren ist die `TActionList`. Die Funktionen werden in `TActions` gekapselt und dann den Bedienelementen zu geordnet, welche dann die Namen (`Caption`), Hints und Glyphs übernehmen. Änderungen werden in den Actions vorgenommen und die Bedienelemente (z.B: *Menüeinträge und Buttons*) übernehmen die GUI-relevanten Eigenschaften automatisch von ihnen zugewiesenen Actions. Ebenfalls ein Beitrag zur Konsistenz und leichter Wartbarkeit bietet `TImageList`. Alle verwendeten **Glyphs** werden in einer `ImageList` gesammelt und den **Actions** bzw. Bedienelementen ohne verbundene Actions zugewiesen. Ändert man ein Bildchen (Glyph), weil es besser zu einer konsistenten Oberfläche passt, wird es automatisch überall geändert, wo es eingesetzt wurde. `ActionLists` und `ImageLists` von **Lazarus** sind eine große Hilfe, einheitliches Look & Feel zu erhalten und zu pflegen.

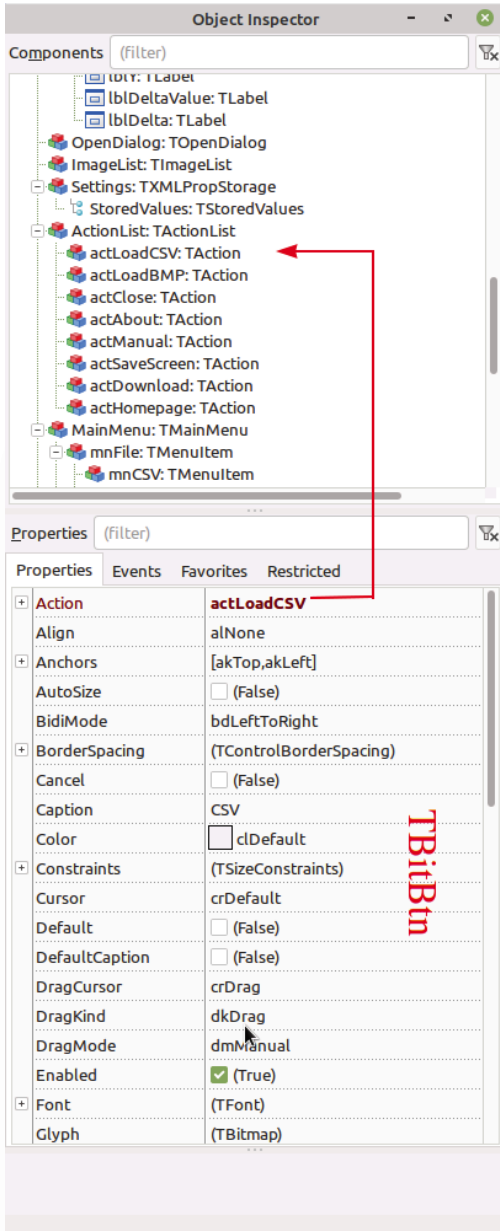


Abbildung 2:
 OnClick des Bedienelements der Action zuweisen

5 UNTERSTÜTZUNG

Dem Benutzer sollte die Arbeit erleichtert werden wo es nur geht. In den meisten Fällen ist sinnvoll, dass man das Programmfenster nach eigenem Gutdünken vergrößern oder verkleinern kann. Nur sehr selten ist eine fixe Größe sinnvoll. Bereiche, die viel Information beinhalten (z.B. *Protokollausgaben oder längere Eingaben erfordern*, z.B. *Pfadnamen*) sollten sich mit dem Programmfenster vergrößern. Besonders unfreundlich ist, wenn sich Textausgabefelder nicht vergrößern lassen, so dass der Benutzer unnötig zum Scrollen gezwungen wird. Dazu kann man die Komponenten-Eigenschaft `anchors` (`akTop`, `akBottom`, `akLeft`, `akRight`) entsprechend anpassen. Vergrößerbare Fenster sind bei Lazarus Standard. Daran sollte man ohne Not auch nichts ändern. Man kann und soll aber verhindern, dass die Fenster zu klein gezogen werden, so dass sich Bedienelemente überschneiden. Dazu gibt es die Eigenschaft `constraints` in `TForm`. Mit `MinHeight` und `MinWidth` stellt man die minimale Größe des Programmfensters benutzerfreundlich ein.

Wichtige Einstellungen des Programms und Verhalten des Programmfensters sollte man für den nächsten Aufruf speichern. Dazu stellt Lazarus drei Formate unter **Misc** zur Verfügung:

- **INI Datei** `TIniPropStorage`
- **JSON** `TJSONPropStorage`
- **XML** `TXMLPropStorage`

Eine dieser Komponenten plaziert man auf dem Formular. Welches Format man nimmt, bleibt dem persönlichen Geschmack des Programmierers überlassen. Die Datei zur Speicherung der Werte im jeweiligen Format werden unter Windows im Verzeichnis des Programms und unter **LINUX** als versteckte Datei im Home-Verzeichnis gespeichert.

Die Eigenschaft `sessionProperties` von `TForm` macht das Verwalten der zu speichernden Einstellungen leicht.





Wichtige Einstellungen des Programms und Verhalten des Programmfensters sollte man für den nächsten Aufruf speichern. Dazu stellt Lazarus drei Formate unter **Misc** zur Verfügung:

- **INI file** TIniPropStorage
- **JSON** TJSONPropStorage
- **XML** TXMLPropStorage

Eine dieser Komponenten plaziert man auf dem Formular. Welches Format man nimmt, bleibt dem persönlichen Geschmack des Programmierers überlassen. Die Datei zur Speicherung der Werte im jeweiligen Format werden unter Windows im Verzeichnis des Programms und unter **LINUX** als versteckte Datei im Home-Verzeichnis gespeichert.

Die Eigenschaft `SessionProperties` von `TForm` macht das Verwalten der zu speichernden Einstellungen leicht.

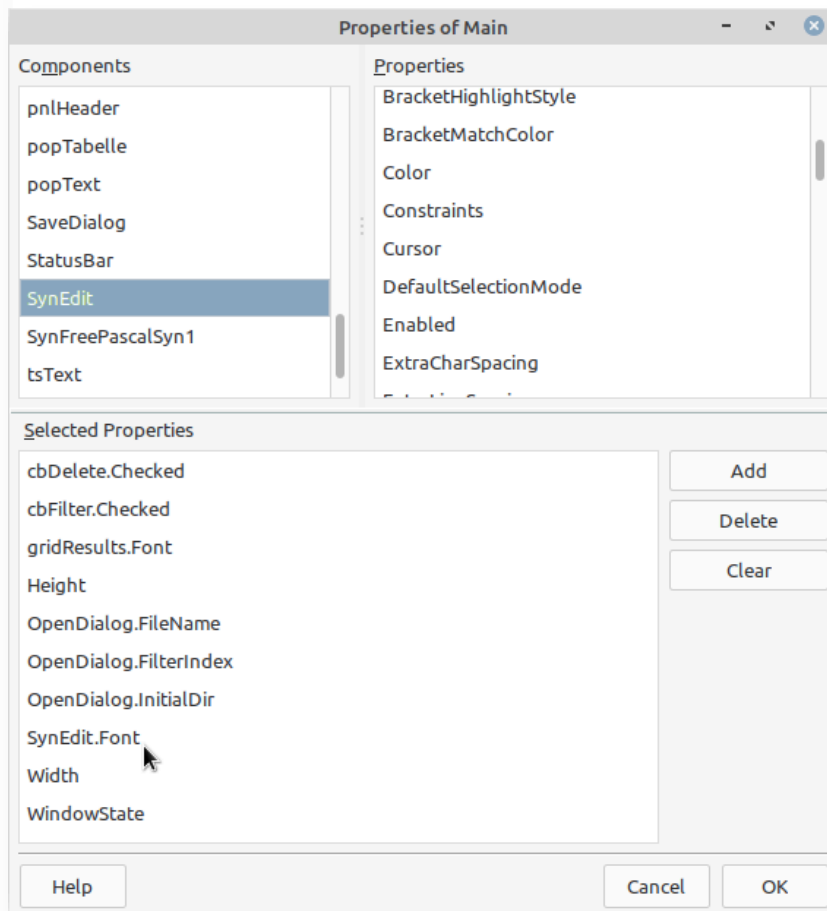


Bild3: Verwaltung der SessionProperties in der Lazarus-IDE





Bei Textausgaben sollte es möglich sein, die Größe der Buchstaben zu verändern, um bessere Lesbarkeit zu erzielen. Üblicherweise geht das mit **Shift + Scrollrad** an der Maus. Dazu benutzt man die Ereignisse `OnMouseWheelDown` und `OnMouseWheelUp`.

Example for TStringGrid:

```
procedure TForm1.StringGrid1MouseWheelDown(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;
```

6 HILFE

Es geht hier nicht um die berühmt-berüchtigten Hilfesysteme, die ein Handbuch ersetzen sollen oder im besten Fall auch können, sondern um die Hilfen, die das **GUI-Gestaltung** mitbringen kann. Hierzu zählen der oben erwähnte möglichst sichtbare **Workflow**, die **Übersichtlichkeit** und **Struktur**.

Aber es gibt ein weiteres mächtiges Mittel, um dem Benutzer zu helfen: Die **Hints**. Fast zu jedem Bedienelement, welches die **Lazarus Komponentenpalette** bietet, gibt es die Eigenschaft `Hint`. Um die **Hints** anzuzeigen, muss man zuerst im Formular die Eigenschaft `ShowHint` auf `true` setzen. Dann kann man den einzelnen Bedienelementen am Besten in einer `resourcestring` Sektion `Strings` mit möglichst aussagekräftigem Inhalt zuweisen. **Hints** können auch mehrzeilig sein. Diese Erklärungen tauchen dann überall auf, wo es Sinn macht, wenn der Benutzer den Mauszeiger darüber hält. Bei einfachen Programmen ist das eigentlich alles, was man als Dokumentation braucht. Das Programm ist selbsterklärend – der Idealfall.

Die Eigenschaft `Hint` bietet noch weitergehende Möglichkeiten, wenn `Hint`-Inhalte abhängig von bestimmten Bedingungen zugewiesen. Hier ein Beispiel, wie man jeder einzelnen Zelle in einem `TStringGrid` einen separaten `Hint` zuweisen kann. Das ist sinnvoll, wenn der Text in der Zelle breiter als die Zelle ist und der `Hint` den gesamten Text anzeigt oder wenn man weitergehende Informationen zum Inhalt der Zelle anzeigen will.

```
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;
```





Was man vermeiden sollte, sind blockierende Messagefenster wie `MessageDlg()` und sonstige modale Dialoge sowie Formulare die mit `ShowModal` aufgerufen werden. Diese immer nur da verwenden, wo es unumgänglich, die weitere Bedienung zu stoppen, um eine Entscheidung zu erzwingen.

Es gibt nichts Nervigeres als nach dem Close-Button die Meldung zu erhalten, „**Wollen Sie wirklich beenden?**“. Das macht wirklich nur dann Sinn, wenn eine wichtige Arbeit noch nicht gespeichert wurde und da sollte es bei der Meldung genau darum gehen. Auch eine **AboutBox** muss nicht unbedingt blockierend (`modal`) sein.

FAZIT

Alles in allem erfordert eine gute Usability eine Menge Planung und Überlegung, viel Schreibarbeit und zusätzlichen Aufwand, macht sich aber in der Akzeptanz des Programms bezahlt. Bei selten benutzten Programmen, selbst wenn ich sie selber geschrieben habe, bin ich oft dankbar, wenn ich einen Hint sehe und wieder weiß, was ich bezweckt hatte.

Beispiel Projekt: `SessionPropertyTool` in `SessionPropertyTool.zip`

EMPFEHLUNGEN

Design

- Mit `TActionList` Programmfunktionen verwalten
- `TImageList` verwenden
- `resourcestrings` Sektion für Captions oder Titles verwenden
- `TStatusBar` und `TProgressBar` für Zustandsmeldung einsetzen
- `Screen.Cursor` kontextabhängig einstellen
- `TGroupBox`, `TPageControl` mit `TTabSheet` zur Strukturierung der Bedienelemente

TForm:

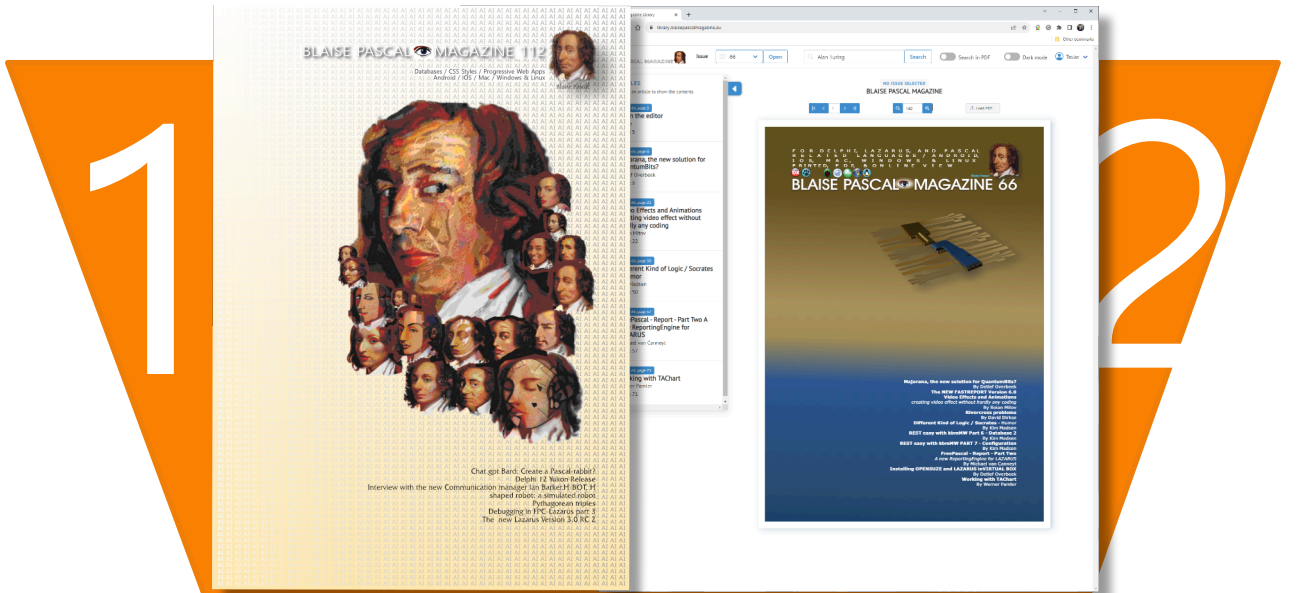
- Constraints festlegen
- `SessionProperties` verwenden
- `ShowHint` auf `true` setzen und Hint-Texte verwenden
- `ShowModal` vermeiden

Komponenten, Bedienelemente:

- Anchors für optimale Größen von Text- und Eingabefelder einstellen
- `TabStop`, `TabOrder` festlegen
- Eher `Enabled` statt `Visible` benutzen
- Hints erstellen
- Nur notwendige modale Dialoge verwenden
- Font Größe veränderbar



ADVERTISEMENT



LAZARUS HANDBOOK
POCKET Edition + shipment

3

LAZARUS HANDBOOK PDF

4

LEARN TO PROGRAM USING LAZARUS
 HOWARD PAGE-CLARK

5

DAVID DIRKSE
 including 50 example projects

6

BLAISE PASCAL MAGAZINE
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

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**





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.

