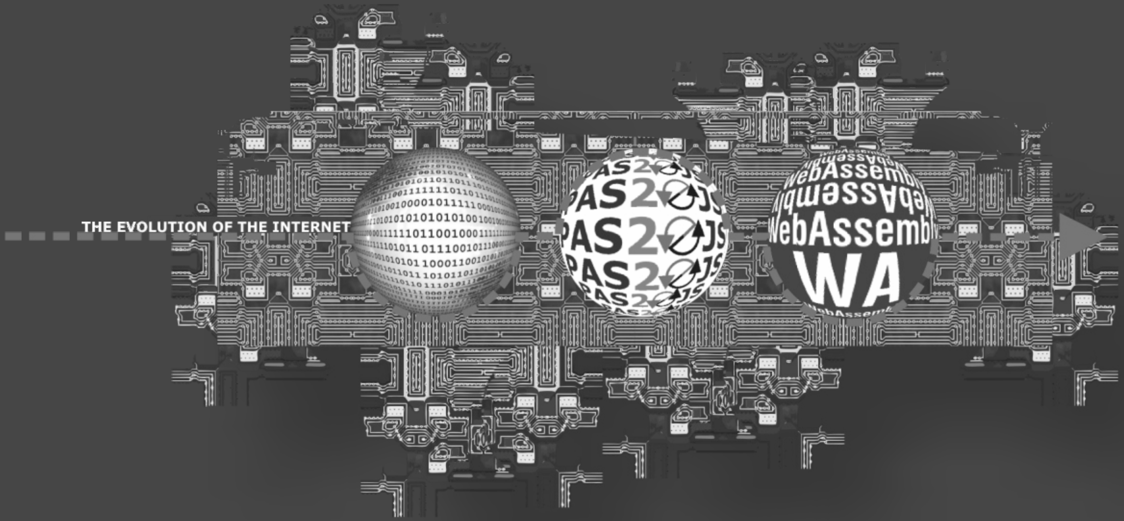
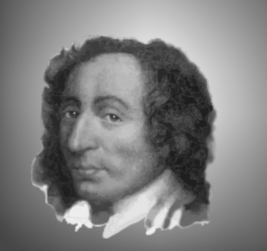
BLAISE PASCAL MAGAZINE 83

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



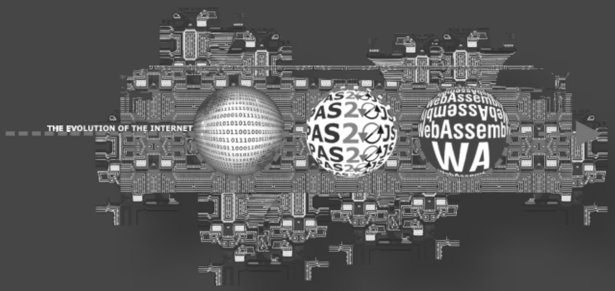
BLAISE PASCAL MAGAZINE 83

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



CONTENT

ARTICLES



ADVERTISERS

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

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

Niklaus Wirth

Stephen Ball http://delphiaball.co.uk @DelphiABall		**Peter Bijlsma -Editor** peter @ blaisepascal.eu
Dmitry Boyarintsev dmitry.living @ gmail.com	**Michaël Van Canneyt,** michael @ freepascal.org	**Marco Cantù** www.marcocantu.com marco.cantu @ gmail.com
David Dirkse www.davdata.nl E-mail: David @ davdata.nl	**Benno Evers** b.evers @ everscustomtechnology.nl	**Bruno Fierens** www.tmssoftware.com bruno.fierens @ tmssoftware.com
Holger Flick holger @ flixments.com		
Primož Gabrijelčič www.primoz @ gabrijelcic.org	**Mattias Gärtner** nc-gaertnma@netcologne.de	**Peter Johnson** http://delphidabbler.com delphidabbler @ gmail.com
Max Kleiner www.softwareschule.ch max @ kleiner.com	**John Kuiper** john_kuiper @ kpnmail.nl	**Wagner R. Landgraf** wagner @ tmssoftware.com
Vsevolod Leonov vsevolod.leonov@mail.ru		**Andrea Magni** www.andreamagni.eu andrea.magni @ gmail.com www.andreamagni.eu/wp
	Paul Nauta PLM Solution Architect CyberNautics paul.nauta @ cybernautics.nl	**Kim Madsen** www.component4developers
Boian Mitov mitov @ mitov.com		**Jeremy North** jeremy.north @ gmail.com
Detlef Overbeek - Editor in Chief www.blaisepascal.eu editor @ blaisepascal.eu	**Howard Page Clark** hdpc @ talktalk.net	**Heiko Rompel** info @ rompelsoft.de
Wim Van Ingen Schenau -Editor wisone @ xs4all.nl	**Peter van der Sman** sman @ prisman.nl	**Rik Smit** rik @ blaisepascal.eu
Bob Swart www.eBob42.com Bob @ eBob42.com	**B.J. Rao** contact @ intricad.com	**Daniele Teti** www.danieleteti.it d.teti @ bittime.it
Anton Vogelaar ajv @ vogelaar-electronics.com	**Robert Welland** support @ objectpascal.org	**Siegfried Zuhr** siegfried @ zuhr.nl

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

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

Correctors
Howard Page-Clark, Peter Bijlsma

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

	Internat. excl. VAT	**Internat. incl. 9% VAT**	**Shipment**
Printed Issue ±60 pages	€ 250	€ 261,60	€ 85,00
Electronic Download Issue 60 pages	€ 60	€ 65,40	——
Printed Issue inside Holland (Netherlands) ±60 pages	——	€ 200	€ 40,00

Member and donator of WIKIPEDIA

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

THE DELPHI COMPANY

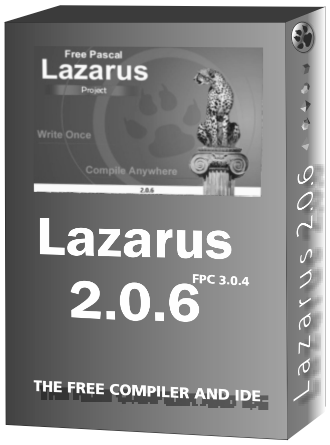
-est 1998-

OS X Android iOs Windows

Vier platforms
Eén ontwikkelomgeving
Eén expertise

DELPHI

www.delphicompany.nl
info@delphicompany.nl



WHAT MOST PEOPLE REALLY DO NOT KNOW ABOUT LAZARUS:

YOU DO NOT NEED TO REINSTALL YOUR SUITES AND COMPONENT LIBRARIES UNDER LAZARUS if you want to upgrade to a new stable version eg: 2.0.4 (or older) and install 2.0.6

If you follow the description you will have no problems keeping your total environment including the extra load components (suites). **THIS IS A GREAT ADVANTAGE,** because installing a new version will cost no more then max 10 minutes including recovering all of your settings.

Start simply by installing and when asked to keep your settings say yes. After that you simply go to the program and recompile. All your components will be there again.

THAT'S IT.

At the moment we are preparing to implement **WebAssembly** into **FreePascal,** (thus Lazarus) so it can be used in conjunction with **Pas2Js**

INTRODUCTION
WebAssembly (*often shortened to Wasm*) is an open standard that defines a portable binary code format for executable programs, and a corresponding textual assembly language, as well as interfaces for facilitating interactions between such programs and their host environment.

THE MAIN GOAL OF WEBASSEMBLY
is to enable high performance applications on web pages, but the format is designed to be executed and integrated in other environments as well.

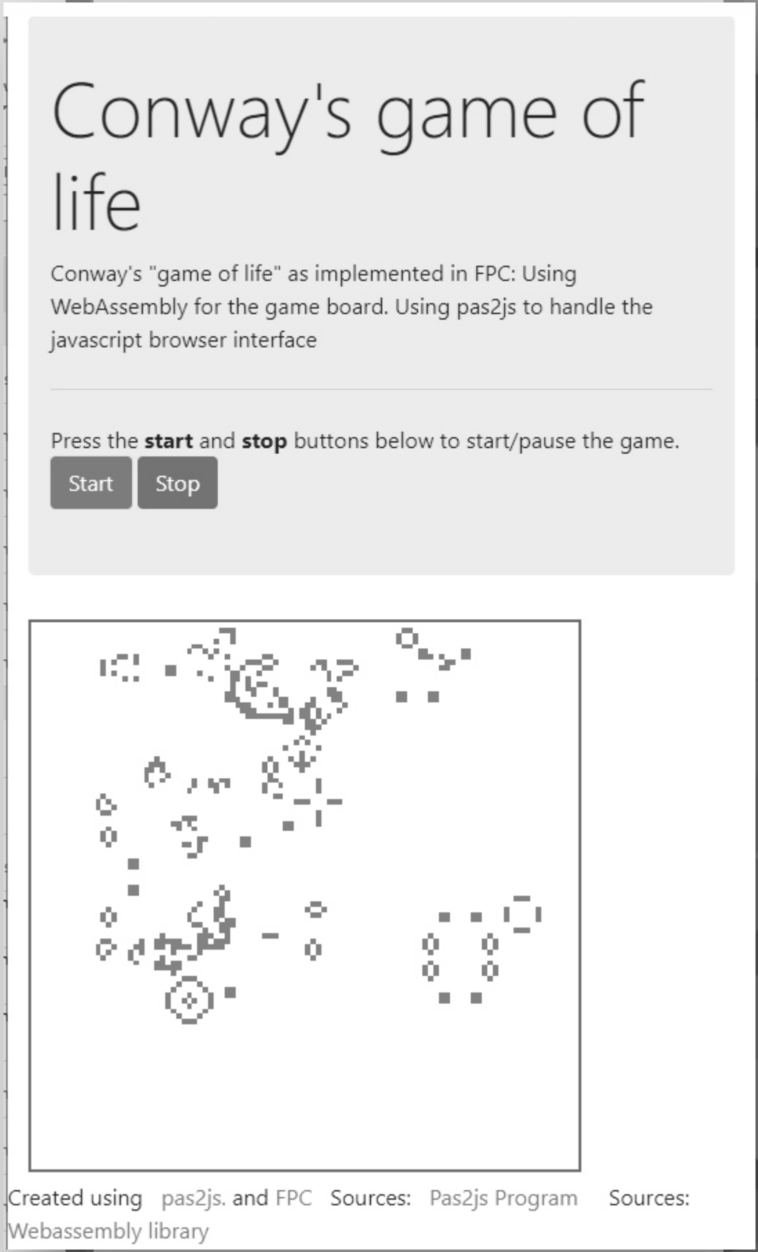
Wasm does not replace JavaScript;
in order to use Wasm in browsers, users may use Emscripten SDK to compile C++ (or any other LLVM-supported language such as D or Rust and **soon PASCAL (FPC))** source code into a binary file which runs in the same sandbox as regular JavaScript code; Emscripten provides bindings for several commonly used environment interfaces like **WebGL;** it has access only to an expandable memory and a small number of scalar values. There is no direct Document Object Model (DOM) access; however, it is possible to create proxy functions for this, for example through stdweb, web_sys, and js_sys. The World Wide Web Consortium (W3C) maintains the standard with contributions from Mozilla, Microsoft, Google, and Apple.

IN A NUTSHELL
WebAssembly has huge implications for the web platform — it provides a way to run code written in multiple languages on the web at near native speed, with client apps running on the web that previously couldn't have done so.

WebAssembly
is designed to complement and run alongside **JavaScript** - using the **WebAssembly JavaScript APIs.**

You can load **WebAssembly** modules into a **JavaScript** app and share functionality between the two. This allows you to take advantage of **WebAssembly's** performance and power and **JavaScript's** expressiveness and flexibility in the same apps, **even if you don't know how to write WebAssembly code.**

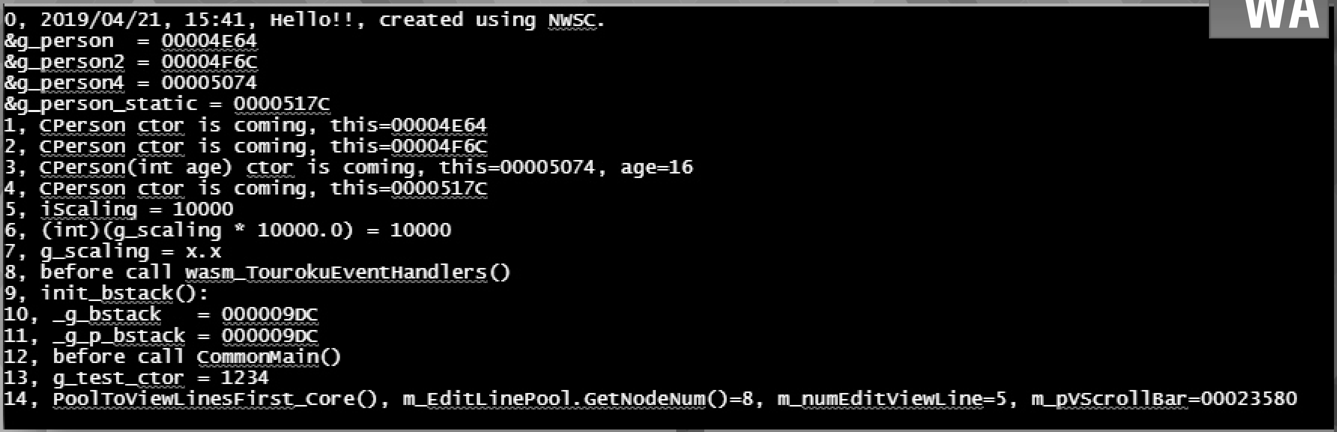
And what's even better, is that it is being developed as a web standard via the W3C WebAssembly Working Group and Community Group with active participation from all major browser vendors.


Conway's game of life

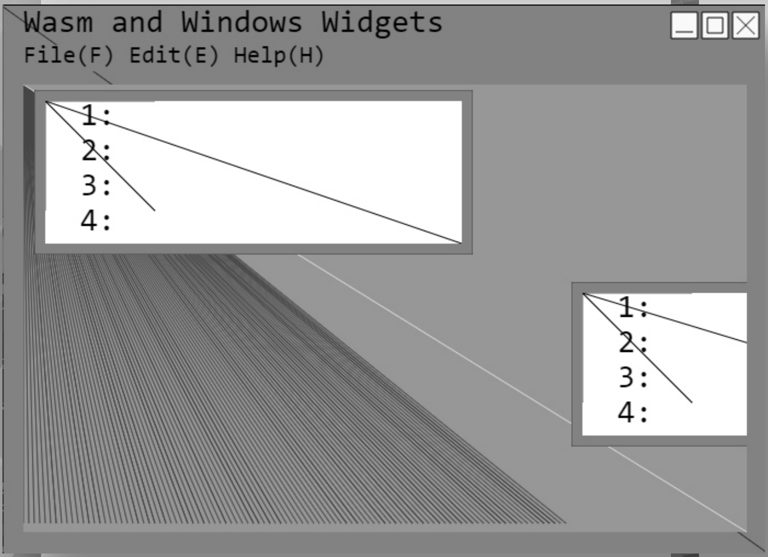
Conway's "game of life" as implemented in FPC: Using WebAssembly for the game board. Using pas2js to handle the javascript browser interface

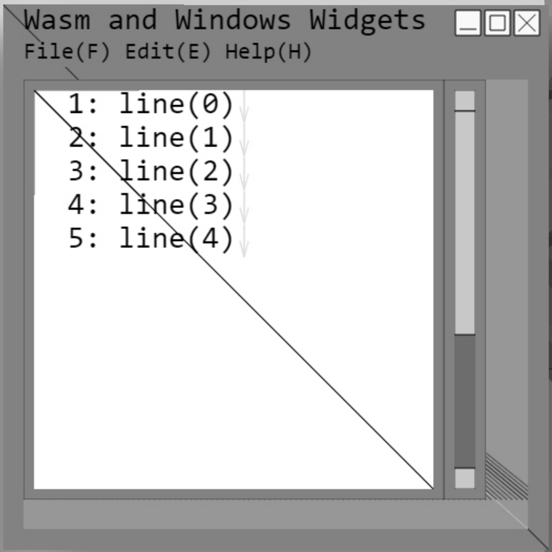
Press the **start** and **stop** buttons below to start/pause the game.

Created using pas2js. and FPC Sources: Pas2js Program Sources: Webassembly library



```
0, 2019/04/21, 15:41, Hello!!, created using NWSC.
&g_person  = 00004E64
&g_person2 = 00004F6C
&g_person4 = 00005074
&g_person_static = 0000517C
1, CPerson ctor is coming, this=00004E64
2, CPerson ctor is coming, this=00004F6C
3, CPerson(int age) ctor is coming, this=00005074, age=16
4, CPerson ctor is coming, this=0000517C
5, iScaling = 10000
6, (int)(g_scaling * 10000.0) = 10000
7, g_scaling = x.x
8, before call wasm_TourokuEventHandlers()
9, init_bstack():
10, _g_bstack   = 000009DC
11, _g_p_bstack = 000009DC
12, before call CommonMain()
13, g_test_ctor = 1234
14, PoolToViewLinesFirst_Core(), m_EditLinePool.GetNodeNum()=8, m_numEditViewLine=5, m_pVScrollBar=00023580
```





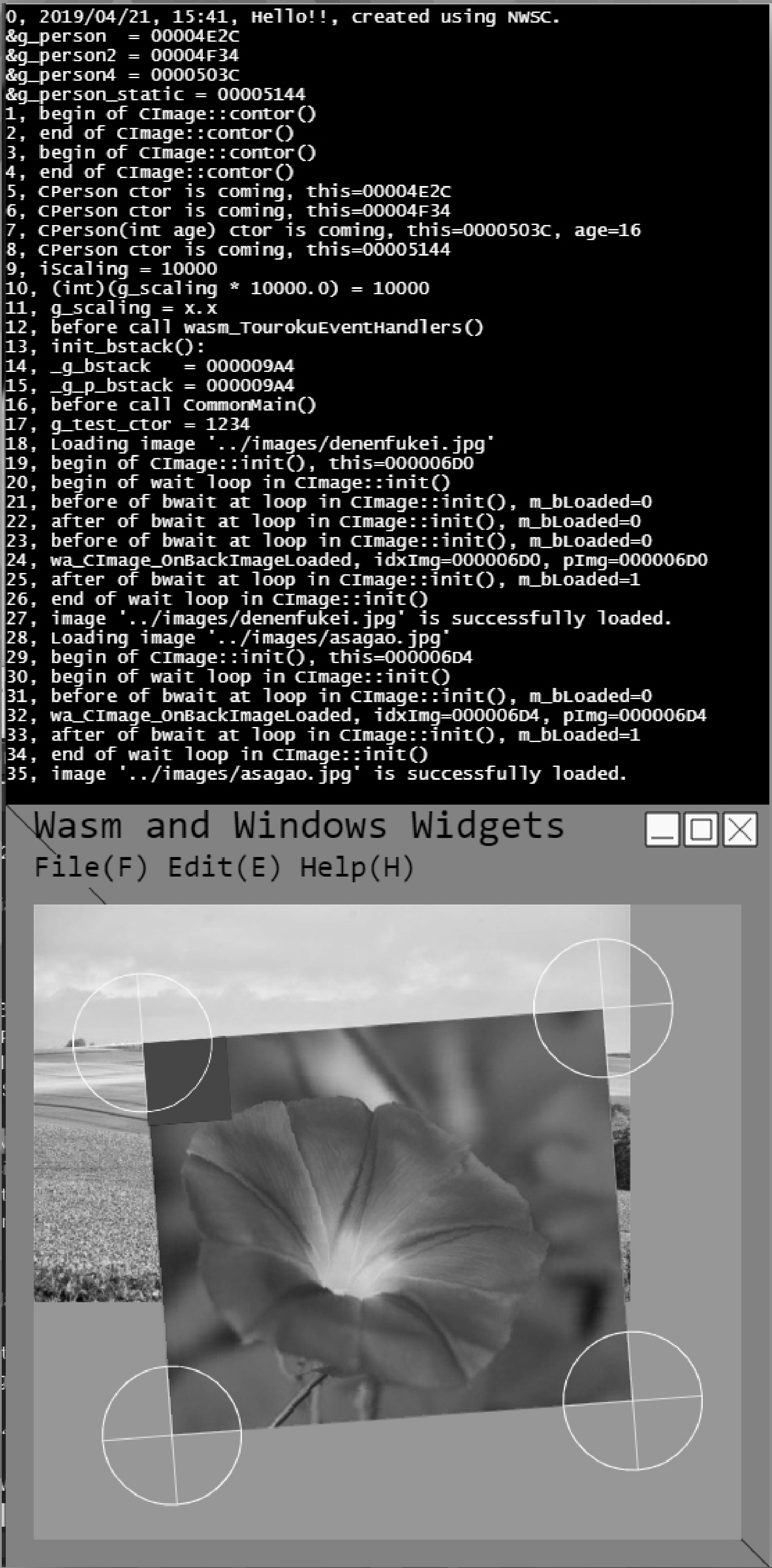
HISTORY
WebAssembly was first announced in 2015, and the first demonstration was executing **Unity's Angry Bots** in Firefox, Google Chrome, and Microsoft Edge. The precursor technologies were **asm.js** from Mozilla and Google Native Client and the initial implementation was based on the feature set of **asm.js**. In March 2017, the design of the Minimum Viable product (MVP) (*never thought Embarcadero got the abbreviation from WASM*) was declared to be finished and the preview phase ended. In late September 2017, Safari 11 was released with support. In February 2018, the **WebAssembly Working Group** published three public working drafts for the **Core Specification, JavaScript Interface, and Web API.**

WHAT IS WEBASSEMBLY?
WebAssembly is a new type of code that can be run in modern web browsers and provides new features and major gains in performance.

It is not primarily intended to be written by hand, rather it is designed to be an effective compilation target for low-level source languages like C, C++, Rust, etc.
This has huge implications for the web platform. It provides a way to run code written in multiple languages on the web at near-native speed, with client apps running on the web that previously couldn't have done so.

What's more, you don't even have to know how to create WebAssembly code to take advantage of it. WebAssembly modules can be imported into a web (or **Node.js**) app, exposing WebAssembly functions for use via **JavaScript.**

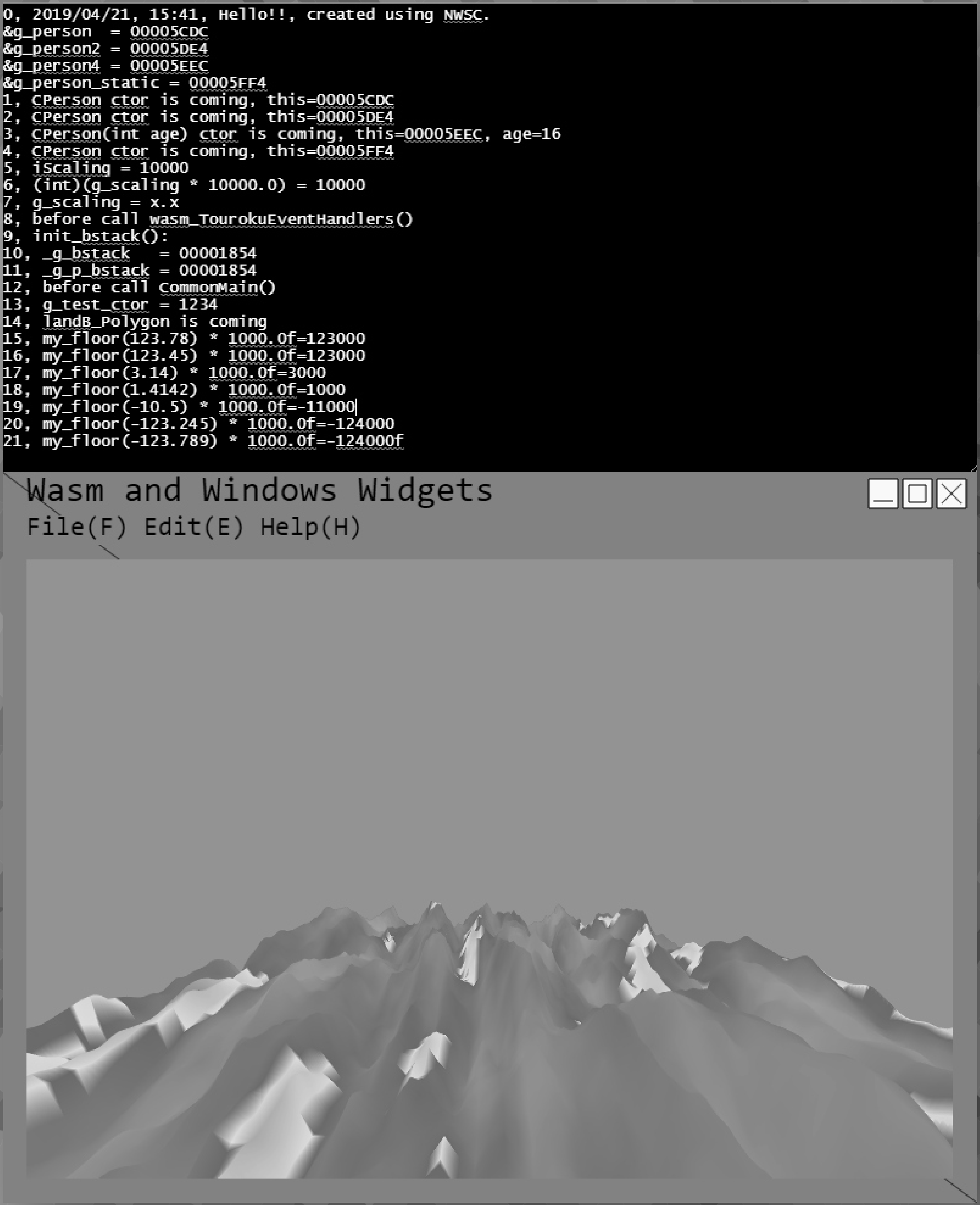
JavaScript frameworks can make use of **WebAssembly** to confer massive performance advantages and new features while still making functionality easily available to web developers.



```
0, 2019/04/21, 15:41, Hello!!, created using NWSC.
&g_person  = 00004E2C
&g_person2 = 00004F34
&g_person4 = 0000503C
&g_person_static = 00005144
1, begin of CImage::contor()
2, end of CImage::contor()
3, begin of CImage::contor()
4, end of CImage::contor()
5, CPerson ctor is coming, this=00004E2C
6, CPerson ctor is coming, this=00004F34
7, CPerson(int age) ctor is coming, this=0000503C, age=16
8, CPerson ctor is coming, this=00005144
9, iScaling = 10000
10, (int)(g_scaling * 10000.0) = 10000
11, g_scaling = x.x
12, before call wasm_TourokuEventHandlers()
13, init_bstack():
14, _g_bstack   = 000009A4
15, _g_p_bstack = 000009A4
16, before call CommonMain()
17, g_test_ctor = 1234
18, Loading image '../images/denenfukei.jpg'
19, begin of CImage::init(), this=000006D0
20, begin of wait loop in CImage::init()
21, before of bwait at loop in CImage::init(), m_bLoaded=0
22, after of bwait at loop in CImage::init(), m_bLoaded=0
23, before of bwait at loop in CImage::init(), m_bLoaded=0
24, wa_CImage_OnBackImageLoaded, idxImg=000006D0, pImg=000006D0
25, after of bwait at loop in CImage::init(), m_bLoaded=1
26, end of wait loop in CImage::init()
27, image '../images/denenfukei.jpg' is successfully loaded.
28, Loading image '../images/asagao.jpg'
29, begin of CImage::init(), this=000006D4
30, begin of wait loop in CImage::init()
31, before of bwait at loop in CImage::init(), m_bLoaded=0
32, wa_CImage_OnBackImageLoaded, idxImg=000006D4, pImg=000006D4
33, after of bwait at loop in CImage::init(), m_bLoaded=1
34, end of wait loop in CImage::init()
35, image '../images/asagao.jpg' is successfully loaded.
```

WA

WebAssembly is a new type of code that can be run in modern web browsers and provides new features and major gains in



```
0, 2019/04/21, 15:41, Hello!!, created using NWSC.
&g_person  = 00005CDC
&g_person2 = 00005DE4
&g_person4 = 00005EEC
&g_person_static = 00005FF4
1, CPerson ctor is coming, this=00005CDC
2, CPerson ctor is coming, this=00005DE4
3, CPerson(int age) ctor is coming, this=00005EEC, age=16
4, CPerson ctor is coming, this=00005FF4
5, iScaling = 10000
6, (int)(g_scaling * 10000.0) = 10000
7, g_scaling = x.x
8, before call wasm_TourokuEventHandlers()
9, init_bstack():
10, _g_bstack    = 00001854
11, _g_p_bstack = 00001854
12, before call CommonMain()
13, g_test_ctor = 1234
14, landB_Polygon is coming
15, my_floor(123.78) * 1000.0f=123000
16, my_floor(123.45) * 1000.0f=123000
17, my_floor(3.14) * 1000.0f=3000
18, my_floor(1.4142) * 1000.0f=1000
19, my_floor(-10.5) * 1000.0f=-11000
20, my_floor(-123.245) * 1000.0f=-124000
21, my_floor(-123.789) * 1000.0f=-124000f
```

SUPPORT

In **November 2017**, Mozilla declared support "in all major browsers" (by now all major on mobile and desktop), after WebAssembly was enabled by default in Edge 16.
The support includes mobile web browsers for iOS and Android.

As of **September 2019**, 87.42% of installed browsers (89.39% of desktop browsers and 87.4% of mobile browser) support WebAssembly.

But for older browsers, Wasm can be compiled into `asm.js` by a JavaScript polyfill.

Because **WebAssembly executables** are precompiled, it is possible to use a variety of programming languages to make them.
This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm.

There have been around 40 programming languages reported to support **Wasm** as a compilation target.

Emscripten* can compile C and C++ to Wasm using LLVM in the backend. Its initial aim is to support compilation from C and C++, though support for other source languages such as Rust and .NET languages is also emerging. After the MVP release, there are plans to support multithreading and garbage collection which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor) and F# (supported via Bolero with help of Blazor); Java, Julia, Ruby, as well as Go.

**Emscripten is a source-to-source compiler that runs as a back end to the LLVM compiler and produces a subset of JavaScript known as asm.js. It can also produce WebAssembly.*

This allows applications and libraries originally designed to run as standard executables to be integrated into client side web applications. asm.js *can be compiled by browsers ahead of time meaning that the compiled programs can run much faster than those traditionally written in JavaScript.*

The **LLVM compiler** *infrastructure project is a set of compiler and toolchain technologies,* **which can be used to develop a front end for any programming language and a back end for any instruction set architecture**.
LLVM is designed around a language-independent intermediate representation that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.

LLVM is written in **C++** and is designed for compile-time, link-time, run-time, and "idle-time" optimization.
Originally implemented for C and C++, the language-philosophy design of **LLVM** has since spawned a wide variety of front ends:
languages with compilers that use LLVM include ActionScript, Ada, C#, Common Lisp, Crystal, CUDA, D, **Delphi**, Dylan, Fortran, **FREE PASCAL**, Graphical G Programming Language, Halide, Haskell, Java bytecode, Julia, Kotlin, Lua, Objective-C, OpenGL Shading Language, Ruby, Rust, Scala, Swift, and Xojo.

Security considerations

In June 2018, a security researcher presented the possibility of using WebAssembly to circumvent browser mitigations
for Spectre and Meltdown security vulnerabilities once support for threads with shared memory is added. Due to this concern, WebAssembly developers put the feature on hold. Thread support was eventually added in October 2018.

Embedding

The general standards provide core specifications for JavaScript and Web embedding. While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts. WebAssembly System Interface (WASI) is an ABI designed by Mozilla intended to define a simpler ABI for WebAssembly that can be used in any platform. There are also a few other proposed ABI APIs.

WEBASSEMBLY KEY CONCEPTS
There are several key concepts needed to understand how WebAssembly runs in the browser. All of these concepts are reflected 1:1 in the WebAssembly JavaScript API.

- **Module:**
  Represents a WebAssembly binary that has been compiled by the browser into executable machine code. A Module is stateless and thus, like a Blob, can be explicitly shared between windows and workers (via postMessage()). A Module declares imports and exports just like an Es2015module.

- **Memory:**
  A resizable ArrayBuffer that contains the linear array of bytes read and written by WebAssembly's low-level memory access instructions.

- **Table:**
  A resizable typed array of references (*e.g. to functions*) that could not otherwise be stored as raw bytes in Memory (*for safety and portability reasons*).

- **Instance:**
  A Module paired with all the state it uses at runtime including a Memory, Table, and set of imported values.
  An Instance is like an **ES2015 module** that has been loaded into a particular global with a particular set of imports.

The JavaScript API provides developers with the ability to create modules, memories, tables, and instances.

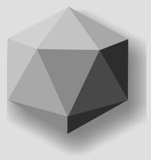
Given a **WebAssembly instance,** JavaScript code can synchronously call its exports, which are exposed as normal JavaScript functions.

Arbitrary JavaScript functions can also be synchronously called by WebAssembly code by passing in those JavaScript functions as the imports to a WebAssembly instance.

Since JavaScript has complete control over how WebAssembly code is downloaded, compiled and run, JavaScript developers could even think of WebAssembly as just a JavaScript feature for efficiently generating high-performance functions.

In the future, WebAssembly modules will be loadable just like ES2015 modules (using `<script type='module'>`), meaning that **JavaScript will be able to fetch, compile, and import a WebAssembly module as easily as an ES2015 module.**

`https://developers.google.com/web/shows/ttt/series-2/es2015`



WEBASSEMBLY GOALS

WebAssembly is being created as an open standard inside the W3C WebAssembly Community Group with the following goals:

- Be fast, efficient, and portable -
  WebAssembly code can be executed at near-native speed across different platforms by taking advantage of common hardware capabilities.

- Be readable and debuggable -
  WebAssembly is a low-level assembly language, but it does have a human-readable text format (*the specification for which is still being finalized*) that allows code to be written, viewed, and debugged by hand.

- Keep secure -
  WebAssembly is specified to be run in a safe, sandboxed execution environment.
  Like other web code, it will enforce the browser's same-origin and permissions policies.

- Don't break the web -
  WebAssembly is designed so that it plays nicely with other web technologies and maintains backwards compatibility.

Note: *WebAssembly will also have uses outside web and JavaScript environments (see Non-web embeddings).*

Non-Web Embeddings
While WebAssembly is designed to run on the Web, it is also desirable for it to be able to execute well in other environments, including everything from minimal shells for testing to full-blown application environments e.g. on servers in datacenters, on IoT devices, or mobile/desktop apps. It may even be desirable to execute WebAssembly embedded within larger programs.
Non-Web environments may provide different APIs than Web environments, which feature testing and dynamic linking will make discoverable and usable.
Non-Web environments may include JavaScript VMs (e.g. node.js), however WebAssembly is also being designed to be capable of being executed without a JavaScript VM present.

The **WebAssembly** spec itself will not try to define any large portable libc-like library. However, certain features that are core to **WebAssembly** semantics that are similar to functions found in native libc would be part of the core **WebAssembly** spec as primitive operators

(*e.g., the grow_memory operator, which is similar to the sbrk function on many systems, and in the future, operators similar to dlopen*).

Where there is overlap between the Web and popular non-Web environments, shared specs could be proposed, but these would be separate from the **WebAssembly** spec.

A symmetric example in JavaScript would be the in-progress Loader spec, which is proposed for both Web and node.js environments and is distinct from the **JavaScript spec**.

However, for most cases it is expected that, to achieve portability at the source code level, communities would build libraries that mapped from a source-level interface to the host environment's built-in capabilities (*either at build time or runtime*).

WebAssembly would provide the raw building blocks (**feature testing, builtin modules and dynamic loading**) to make these libraries possible.

Two early expected examples are **POSIX** and **SDL.**
In general, by keeping the non-Web path such that it doesn't require Web APIs, **WebAssembly** could be used as a **portable binary format on many platforms**, bringing great benefits in portability, tooling and language-agnosticity (*since it supports C/C++ level semantics*).

FEATURES

Stack machine

Wasm code is intended to be run on a portable abstract structured virtual stack machine (VM). The VM is designed to be **faster to parse than JavaScript, as well as faster to execute and to enable very compact code representation.**

Instruction set

The core standard defines a unique **Instruction Set Architecture** consisting of specific binary encoding and which is intended to be executed by VM. However it doesn't specify how exactly they must be invoked by it.

Representation

In March 2017, the WebAssembly Community Group reached consensus on the initial (MVP) binary format, JavaScript API, and reference interpreter.

It defines a **WebAssembly binary format**, which is not designed to be used by humans, as well as a human-readable linear assembly bytecode format that resembles traditional assembly languages.

The table below represents three different views of the same source code input from the left, as it is converted to a Wasm intermediate representation, then to Wasm binary instructions.

The WebAssembly text format can also be written in a folded format using s-expressions. This format is purely syntactic sugar and has no behavioral differences with the linear format. An example is shown below:

```
(module
 (import "math" "exp" (func $exp (param f64) (result f64)))
 (func (export "doubleExp") (param $0 f64) (result f64)
  (f64.mul
   (call $exp
    (get_local $0)
   )
   (f64.const 2)
  )
 )
)
```

WEBASSEMBLY
is a new type of code
that can be run in modern web browsers
- it is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++ and Rust with a compilation target so that they can run on the web.

It is also designed to run alongside JavaScript, allowing both to work together.

C input source	Linear assembly bytecode (intermediate representation)	Wasm binary encoding (hexadecimal bytes)
`int factorial(int n) {`	`get_local 0`	`20 00`
`  if (n == 0)`	`i64.eqz`	`50`
`    return 1;`	`if (result i64)`	`04 7E`
`  else`	`    i64.const 1`	`42 01`
`    return n * factorial(n-1);`	`else`	`05`
`}`	`    get_local 0`	`20 00`
	`    get_local 0`	`20 00`
	`    i64.const 1`	`42 01`
	`    i64.sub`	`7D`
	`    call 0`	`10 00`
	`    i64.mul`	`7E`
	`end`	`0B`

HOW DOES WEBASSEMBLY FIT INTO THE WEB PLATFORM?

The web platform can be thought of as having two parts:

- A **virtual machine (VM)** that runs the Web app's code, e.g. the JavaScript code that powers your apps.
- A set of **Web APIs** that the Web app can call to control web browser/device functionality and make things happen (*DOM, CSSOM, WebGL, IndexedDB, Web Audio API, etc.*).

Historically, the **VM** has been able to load only JavaScript. This has worked well for us as JavaScript is powerful enough to solve most problems people have on the Web today. We have run into performance problems, however, when trying to use **JavaScript** for more intensive **use cases like 3D games, Virtual and Augmented Reality, computer vision, image/video editing,** and a number of other domains that demand native performance (*see WebAssembly use cases for more ideas*).

Additionally, the cost of downloading, parsing, and compiling very large JavaScript applications can be prohibitive.

Mobile and other resource-constrained platforms can further amplify these performance bottlenecks.

WebAssembly is a different language from **JavaScript,** but it is **not intended as a replacement.**

Instead, It is designed to complement and work alongside **JavaScript,** allowing web developers to take advantage of both languages' strong points:

- **JavaScript** is a high-level language, flexible and expressive enough to write web applications.
  IT HAS MANY ADVANTAGES
  - it is dynamically typed, requires no compile step, and has a huge ecosystem that provides powerful frameworks, libraries, and other tools.

- **WebAssembly** is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages with low-level memory models such as C++ and Rust with a compilation target so that they can run on the web. (*Note that WebAssembly has the high-level goal of supporting languages with garbage-collected memory models in the future.*)

With the advent of **WebAssembly** appearing in browsers, the virtual machine that we talked about earlier will now load and run two types of code
- **JavaScript AND WebAssembly**.

The different code types can call each other as required - **the WebAssembly JavaScript API wraps exported WebAssembly code with JavaScript functions that can be called normally,**
and WebAssembly code can import and synchronously call normal JavaScript functions.
In fact, the basic unit of **WebAssembly** code is called a module and **WebAssembly** modules are symmetric in many ways to ES2015 modules.

ES2015 (formally ES6) is a fantastic step forward for the JavaScript language.
It brings new features and sugaring for patterns that required significant boilerplate in ES5.
This includes classes, arrow functions and modules.
In this episode, we cover tools we use to take full advantage of ES2015 when building JavaScript
ECMAScript 2015
The 6th edition, initially known as ECMAScript 6 (ES6) then and later renamed to ECMAScript 2015, was finalized in June 2015.

HOW DO I USE WEBASSEMBLY IN MY APP?

Above we talked about the raw primitives that WebAssembly adds to the Web platform:
a binary format for code and APIs for loading and running this binary code.

Now let's talk about how we can use these primitives in practice.

The WebAssembly ecosystem is at a nascent stage; more tools will undoubtedly emerge going forward.

Right now, there are four main entry points:

- **Porting**
  a **C/C++** application with **Emscripten.**

- **Writing or generating**
  WebAssembly directly at the assembly level.

- **Writing a Rust** application
  and targetting WebAssembly as its output.
  (Rust is a multi-paradigm system programming language focused on safety, especially safe concurrency. Rust is syntactically similar to C++, but is designed to provide better memory safety while maintaining high performance.

Rust was originally designed by Graydon Hoare at Mozilla Research, with contributions from Dave Herman, Brendan Eich, and others.
The designers refined the language while writing the Servo layout or browser engine, and the Rust compiler. The compiler is free and open-source software dual-licensed under the MIT License and Apache License 2.0.

Rust has been the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.

- **Using AssemblyScript**
  which looks similar to **TypeScript** and compiles to WebAssembly binary.

Two of the many options for creating WASM code are an online wasm assembler or Emscripten. There are a number of online WASM assembler choices, such as:

- WasmFiddle
- WasmFiddle++
- WasmExplorer

These are great resources for people who are trying to figure out where to start, but they lack some of the tooling and optimizations of Emscripten.

The **Emscripten** tool is able to take just about any C/C++ source code and compile it into a `.wasm` module, plus the necessary JavaScript "glue" code for loading and running the module, and an HTML document to display the results of the code.

The JavaScript glue code is not as simple as you might imagine.
For a start, Emscripten implements popular C/C++ libraries like SDL, OpenGL, OpenAL, and parts of POSIX.
These libraries are implemented in terms of Web APIs and thus each one requires some JavaScript glue code to connect WebAssembly to the underlying Web API.

So part of the glue code is implementing the functionality of each respective library used by the C/C++ code.
The glue code also contains the logic for calling the above-mentioned WebAssembly JavaScript APIs to fetch, load and run the `.wasm` file.

The generated HTML document loads the JavaScript glue file and writes stdout to a `<textarea>`.

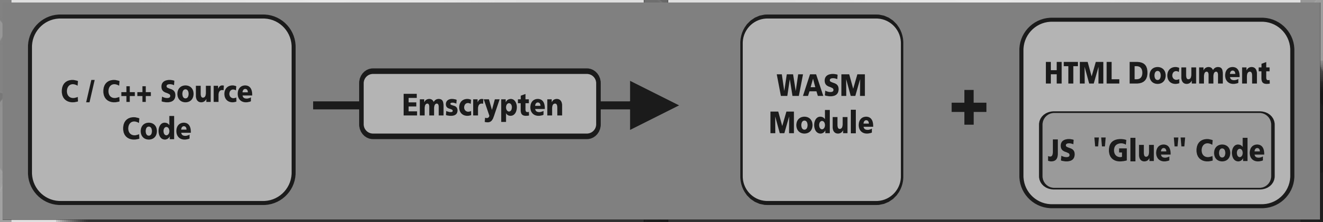
If the application uses OpenGL, the HTML also contains a `<canvas>` element that is used as the rendering target. It's very easy to modify the Emscripten output and turn it into whatever web app you require.
You can find full documentation on Emscripten at `emscripten.org`, and a guide to implementing the toolchain and compiling your own C/C++ app across to wasm at Compiling from C/C++ to WebAssembly.

Writing WebAssembly directly
Do you want to build your own compiler, or your own tools, or make a JavaScript library that generates WebAssembly at runtime?

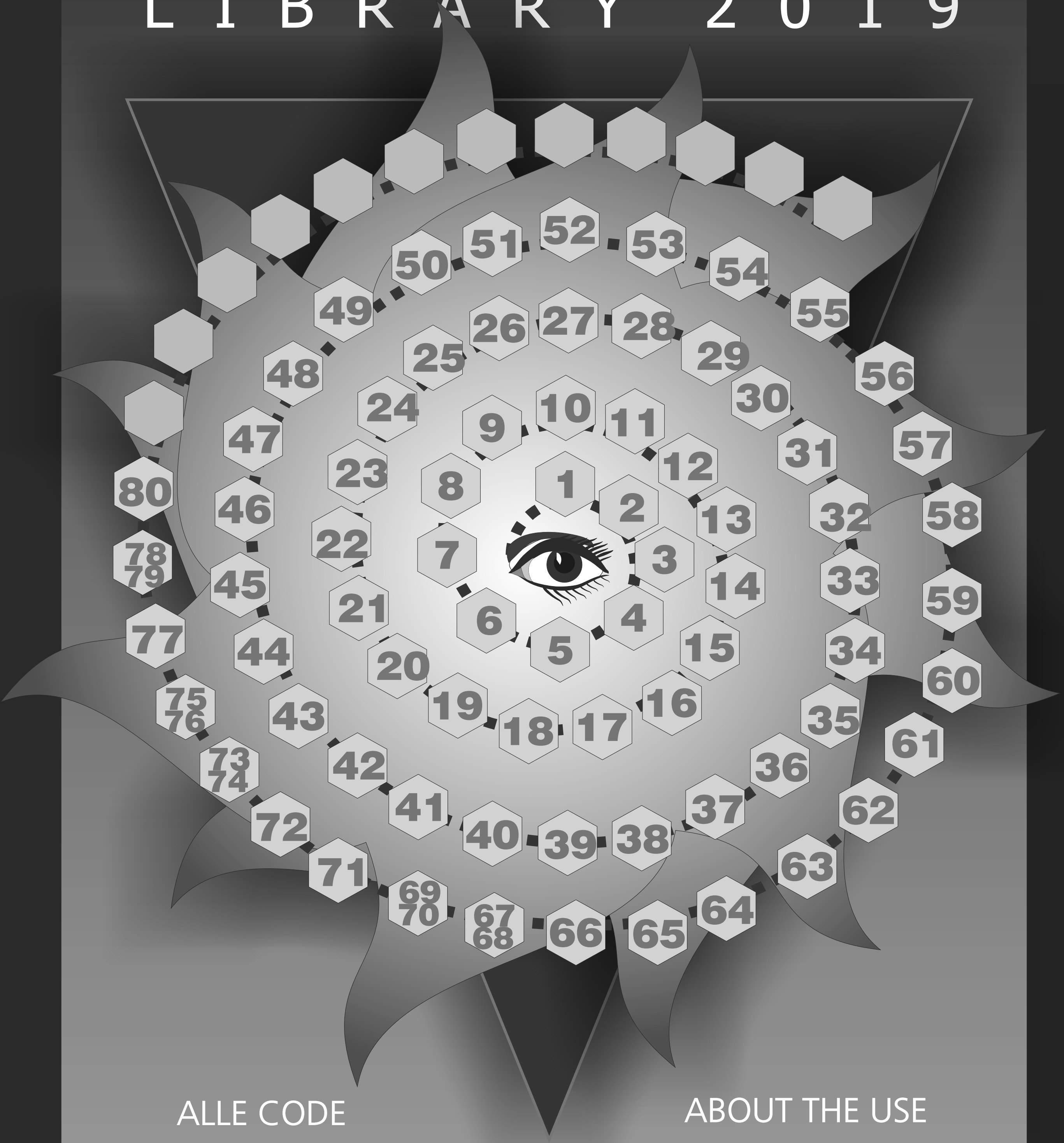
In the same fashion as physical assembly languages, the WebAssembly binary format has a text representation — the two have a 1:1 correspondence.
You can write or generate this format by hand and then convert it into the binary format with any of several WebAssemby text-to-binary tools.

For a simple guide on how to do this, see our Converting WebAssembly text format to wasm article.

USING ASSEMBLYSCRIPT
For web developers who want to try WebAssembly without needing to learn the details of C or Rust, AssemblyScript will be the best option.
It generates a small bundle and its performance is slightly slower compared to C or Rust. You can check its documentation on `https://docs.assemblyscript.org/`



LIBRARY 2019



ALLE CODE

ABOUT THE USE

BLAISE PASCAL MAGAZINE

ALL ISSUES IN ONE FILE

BY MICHAEL VAN CANNEYT

starter expert

ABSTRACT
In another article in this issue, WebAssembly has been introduced. In this article, we explain how to use WebAssembly from Free Pascal (FPC).

INTRODUCTION

WebAssembly is a form of byte code.This makes it possible to create a back-end to FPC which emits WebAssembly bytecode.This is a distinct approach from pas2js, which is implemented using a complete rewrite of the compilerwith a backend that emits javascript.The ability to create WebAssembly is currently being added to FPC. It is a work-in-progress (mainly, Dmitry Boyarintsev),and is far from finished, but some things can already be done.To be able to test it, for the moment, you must build your own version of FPC.The source code is in a GitHub mirror of FPC, awaiting merging entry in the main FPC repository.

So, the first step is to grab the sources. In a terminal window, this is done as follows:

```
git clone https://github.com/skalogryz/freepascal fpc-wasm
```

You can of course choose another name than **fpc-wasm** for the directory.
Then, the work is not in the main branch of the sources, but in a separate branch called **webasm**, so the next step is to switch to this branch:

```
cd fpc-wasm
git checkout webasm
```

`Then, the next step is to actually compile the compiler.

```
cd compilerfpc -dnoopt -dwasm -Fiwasm -Fuwasm -Fusystems -S2 pp.pas
```

Alternatively, you can open the **ppcwasm.lpi** project file in the Lazarus IDE, and hit the compile button. This will leave you with a pp executable, which you should rename:

```
mv pp ppwasm
```

or, on windows, in the command window:

```
mv pp.exe ppwasm.exe
```

The webassembly compiler currently still uses an external wasm linker and assembler and some extra tools. So you must get these from github. The linker and assembler are provided by the webassembly community:

```
https://github.com/WebAssembly/wabt/releases
```

The binaries must be put somewhere where the compiler can find them: in your **PATH** or next to the compiler binary. The extra tools are developed by the FPC team and must be downloaded and compiled separately:

```
git clone
https://github.com/skalogryz/
wasmbin.git
```

the tool that must be compiled is called wasmtool:

```
cd wasmbitfpc -S2h wasmtool.lpr
```

Alternatively you can compile the tool with Lazarus. The wasmtool executable must be placed somewhere where the compiler can find it.

COMPILING A WEBASSEMBLY PROGRAM

Now that we have a compiler, how to use it ?At the moment, the webassembly compiler is not yet able to compile separate units and glue them together: this ability is currently under development. You can have separate units, but they cannot contain code,only definitions. That means that only a single file with a complete program can be compiled and used. As an example (a proof of concept) for the webassembly output, Conway's "game of life" was ported to FPC. You can download it, from **github:**

```
git clone https://github.com/skalogryz/wasm-demo.git
```

The exact implementation of the game will not be discussed here, as it is irrelevant.
What is important is that the webassembly program exports 3 functions, which will be described in the next section.When downloaded, one can compile it just as any other program:

```
cd wasm-demo/lyff
ppcwasm -Fu.. lyff.pas
```

If all went well, the above command will produce a **lyff.wasm** file.

USING A WEBASSEMBLY PROGRAM.

Now that we have a `lyff.wasm` file, how can this be used in the browser?

Wasm files must be loaded expicitly in a web page, using **Javascript.**

It's much like loading a library, getting the addresses of the exported procedures in the library, and using these addresses, as you would in a native application.

Obviously, we will write a Pascal program for this and compile it with `pas2js`.
(we'll assume you have installed pas2js).
First, the HTML page. This is a quite standard HTML page for a `pas2js` project:

```
  <!doctype html>
<html lang="en">
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <title>FPC WebAssembly - Conway's game of life</title>
  <script src="project1.js"></script>
</head>
<body>
  <canvas id="canvas">
  <button id="btn-start" class="btn btn-primary">Start</button>
  <button id="btn-stop" class="btn btn-secondary">Stop</button>
  <script>
  rtl.ru
```

But it has a canvas (*in which the Game of life board will be drawn*) and two buttons to pause/start the game.
The actual HTML shown further in this article has more **markup** in it, but it is only markup, and irrelevant for the functionality.
As mentioned before, **WebAssembly** has some functions that it exports, and usually also some functions that it imports.

These functions must be set up when the assembly is loaded, and they are returned in an object by the browser's `WebAssembly API`.
The following definition mimics the object that will be returned when loading the `lyff.wasm` assembly:

```
Type
  TLyfAPI = Class external name 'anon' (TJSObject)
    Function BoardRef : NativeInt; external name '_board_ref';
    Procedure InitBoard; external name '_board_init';
    Procedure StepBoard; external name '_board_step';
  end;
```

We will see later how this object is obtained, and used.The game of life is a square board, with a function to `step` the board: execute the **'life'** rules to go to the next stage of life in the board.

There are 3 functions in the implementation:

BoardRef	This function returns a reference to the board (a **pointer**, so to speak). This is an index in a **memory area** given to the wasm engine: this is a byte array, and the **BoardRef function** returns the index in the array.
InitBoard	This is a callable procedure to initialise the board.
StepBoard	This is a procedure which performs one step in the game of life.

This API can be used used in the Game of Life project, in a class:.

```
Type
 TGameOfLife = Class
 private
  Exps : TLyfAPI;
  BoardCanvas : TJSHTMLCanvasElement;
  BoardContext : TJSCanvasRenderingContext2D;
  aMemory : TJSWebAssemblyMemory; // Memory of webassembly
  FInterval: NativeInt;
  Procedure DrawBoard;
  function Initlife(res: jsValue): JSValue;
  function CreateWebAssembly(Path: string; ImportObject: TJSObject): TJSPromise;
  function DoStart(aEvent: TJSMouseEvent): boolean;
  function DoStop(aEvent: TJSMouseEvent): boolean;
 Public
  procedure DoStep;
  procedure Execute;
  procedure InitCanvas;
  procedure InitWebAssembly;
 end;
```

The class is quite simple. Everything starts with the **Execute** method:

```
procedure TGameOfLife.Execute;

begin
 TJSHTMLElement(Document.GetElementByID('btn-start')).OnClick:=@DoStart;
 TJSHTMLElement(Document.GetElementByID('btn-stop')).OnClick:=@DoStop;
 InitCanvas;
 InitWebAssembly;
end;
```

First the **OnClick** handlers for the start/stop buttons are assigned, and the HTML canvas is initialized with an appropriate height and width:

```
procedure TGameOfLife.InitCanvas;

Var
 S : String;

begin
 BoardCanvas:=TJSHTMLCanvasElement(Document.getelementByID('canvas'));
 BoardContext:=BoardCanvas.getContextAs2DContext('2d');
 BoardCanvas.width:=dim+2;
 BoardCanvas.height:=dim+2;
 S:=IntToStr(dim*4)+'px';
 BoardCanvas.Style.setProperty('width',S);
 BoardCanvas.Style.setProperty('height',S);
end;
```

The **Dim** constant is the number of squares in the game board (100).Then the interesting part starts: loading the webassembly, and initializing it for execution:

```
procedure TGameOfLife.InitWebAssembly;

Var
 mDesc : TJSWebAssemblyMemoryDescriptor;
 tDesc: TJSWebAssemblyTableDescriptor;
 aTable : TJSWebAssemblyTable;
 ImportObj : TJSObject;

begin
 // Setup memory
 mDesc.initial:=256;
 mDesc.maximum:=256;
 aMemory:=TJSWebAssemblyMemory.New(mDesc);
 // Setup table
 tDesc.initial:=0;
 tDesc.maximum:=0;
 tDesc.element:='anyfunc';
 aTable:=TJSWebAssemblyTable.New(tDesc);
 // Setup ImportObject
 ImportObj:=new([
  'env',New([
    'abortStackOverflow', procedure begin raise Exception.Create('overflow'); end,
    'table', aTable,
    'tableBase', 0,
    'memory', aMemory,
    'memoryBase', 1024,
    'STACKTOP', 0,
    'STACK_MAX', aMemory.buffer.byteLength
  ])
 ]);
 CreateWebAssembly('lyff.wasm',ImportObj)._then(@initLife);
end;
```

A web assembly is executed in a specially prepared environment (*almost like a virtual machine*).

This environment must be prepared: one of the things that must be done is set up a memory block, which must be passed on to the execution environment. The browser has the **WebAssembly.Memory** class for this, and this is what we use.

The execution environment contains some functions that can be called: these must be imported (*or exported*) and the import is described in the import/export table **WebAssembly.Table** class: this is the meaning of the assignment:

```
 tDesc.initial:=0;
tDesc.maximum:=0;
tDesc.element:='anyfunc'; //Only possible value.
aTable:=TJSWebAssemblyTable.New(tDesc);
```

The above sets up the class for use by the **WebAssembly** engine:
When the assembly has been set up, the exports will contain the description of all functions exported by the assembly as a single object.

Finally, an import object is constructed which will be passed to the `CreateWebassembly` function.It contains a procedure that will be called in case of a stack overflow, the table to be filled with export defintions, the memory to use, and the location and size of the stack memory for the engine.

The `CreateWebAssembly function`
returns a Javascript Promise: when it is resolved, `InitLife` will be called:

The `CreateWebAssembly helper function`
does the actual loading and creation of the webassembly.
It hides the details of loading a file (*an asynchronous procedure*).
The location of the webassembly is passed on in the `Path` argument:

```
function TGameOfLife.CreateWebAssembly(Path: string; ImportObject: TJSObject): TJSPromise;

begin
 Result:=window.fetch(Path)._then(Function (res : jsValue) : JSValue
  begin
   Result:=TJSResponse(Res).arrayBuffer._then(Function (res2 : jsValue) : JSValue
    begin
     Result:=TJSWebAssembly.instantiate(TJSArrayBuffer(res2),ImportObject);
    end,Nil)
  end,Nil
 );
end;
```

The `Fetch` api of the browser is used to fetch the wasm file.
The response is then loaded into a `TJSWebAssembly` class, using the `instantiate` function.
The `instantiate` returns again a promise, and the promise is resolved when the assembly is ready to go: in our case, the `initLife` method is called:

```
function TGameOfLife.Initlife(res: jsValue): JSValue;

Var
 Module : TJSInstantiateResult absolute res;

begin
 Exps := TLyfAPI(TJSObject(Module.Instance.exports_));
 Exps.InitBoard;
 DrawBoard;
end;
```

The `InitLife` method gets an `InstantiateResult` instance as a parameter.
This object describes the loaded webassembly module.Module :
`TJSInstantiateResult absolute res`;

This now makes clear what the `TLyfAPI` class was all about:
it gives a correct **Pascal** description of the `exports_element`
of the `InstantiateResult` class: much like a class or interface description.

This class can immediatly be used to initialize the game board.

Once initialized, the board can be drawn:

```
procedure TGameOfLife.DrawBoard;

Var
 Board : TJSUint8Array;
 I,off,P,X,Y, doff: Integer;
 alive : boolean;
 Data : TJSImageData;

begin
 Board:=TJSUint8Array.New(aMemory.buffer,Exps.BoardRef);
 Data:=TJSImageData.new(BoardCanvas.Width,BoardCanvas.Height);
 for x:=1 to dim do
  For y:=1 to dim do
   begin
   P:=(Y*(dim+2))+X;
   I:=p div 8;
   off:=1 shl (p mod 8);
   alive:=(Board[i] and off)<>0;
   if alive then
    begin
    Doff:=(y*boardcanvas.width+X)*4;
    Data.data[doff+0]:=255;
    Data.data[doff+3]:=255;
    end;
   end;
 BoardContext.putImageData(Data,0,0);
end;
```

As can be seen, the board is accessed through a **"pointer"** in the memory block that was created for the execution environment:The **BoardRef function** returns the index in the memory byte array.

The first step is to create a copy of the memory block for use in the routine.

After that, a grid image is created and filled with red color marks (*this is the meaning of 255*) where the board contains a set bit: life is present at that point.

Finally, the image is displayed in the canvas.
The 2 buttons in the HTML are hooked up to the following **OnClick** events:

```
function TGameOfLife.DoStart(aEvent: TJSMouseEvent): boolean;
begin
 Result:=False;
 FInterval:=Window.setInterval(@DoStep,500);
end;

function TGameOfLife.DoStop(aEvent: TJSMouseEvent): boolean;
begin
 Result:=False;
 Window.clearInterval(FInterval);
end;
```

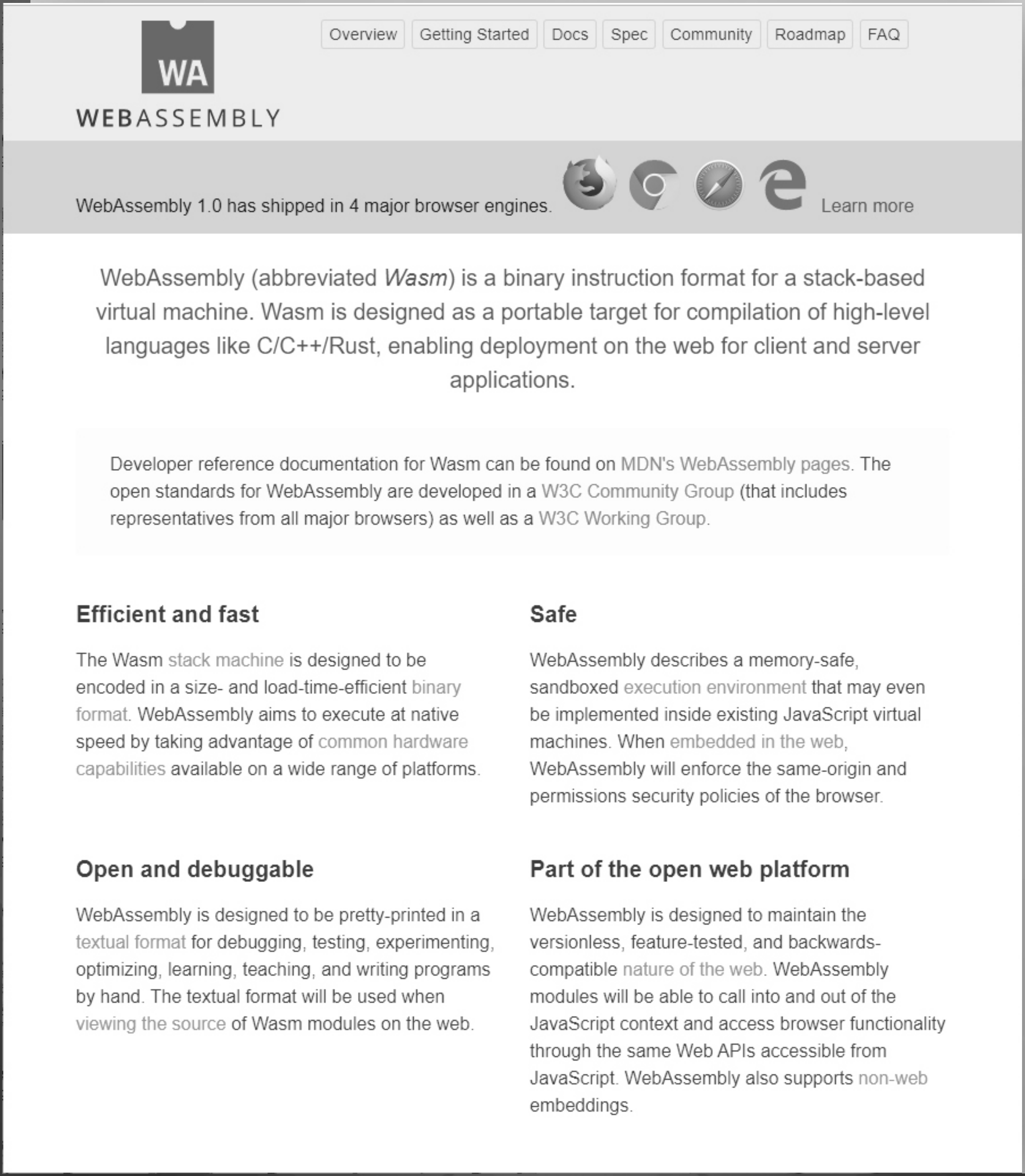
These methods start or stop an interval (*comparable to a* **TTimer** *in Delphi or Lazarus*) in the browser.
The interval function steps the board using the webassembly's **StepBoard** and draws the board:

```
procedure TGameOfLife.DoStep;

begin
 Exps.StepBoard;
 DrawBoard;
end;
```

That's it. The result can be tried on
https://www.freepascal.org/~michael/lyff/

CONCLUSION

WebAssembly is an active area of development in Browsers. Using FPC, it is possible to create WebAssemblies;the support is still embryonic, and many things still need to be implemented before it is usable in production code, but the basics are there, and the brave can already have a go at trying to compile and use wasm code.



WebAssembly 1.0 has shipped in 4 major browser engines.

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

Developer reference documentation for Wasm can be found on MDN's WebAssembly pages. The open standards for WebAssembly are developed in a W3C Community Group (that includes representatives from all major browsers) as well as a W3C Working Group.

Efficient and fast

The Wasm stack machine is designed to be encoded in a size- and load-time-efficient binary format. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms.

Safe

WebAssembly describes a memory-safe, sandboxed execution environment that may even be implemented inside existing JavaScript virtual machines. When embedded in the web, WebAssembly will enforce the same-origin and permissions security policies of the browser.

Open and debuggable

WebAssembly is designed to be pretty-printed in a textual format for debugging, testing, experimenting, optimizing, learning, teaching, and writing programs by hand. The textual format will be used when viewing the source of Wasm modules on the web.

Part of the open web platform

WebAssembly is designed to maintain the versionless, feature-tested, and backwards-compatible nature of the web. WebAssembly modules will be able to call into and out of the JavaScript context and access browser functionality through the same Web APIs accessible from JavaScript. WebAssembly also supports non-web embeddings.

The Lazarus Factory

Friday 29 and Saturday 30 November 2019

INTERNATIONAL LAZARUS/FPC EVENT

The event address:
"DE KLUIS" - Dr. Holtroplaan 1 Eindhoven
CONTACT: Mobile: +31 6 21.23.62.68
Email: admin @ blaisepascalmagazine.eu

All English spoken. (German and Dutch translation available)
The price includes: all drinks during the day (10.00-17.00), Lunch and coffee and thee, fresh and cake.
Hotel suggestion: Inntel Hotels Art Eindhoven, Lichttoren 22, 5611 BJ Eindhoven

Register: https://www.blaisepascalmagazine.eu/

All participants will receive a free CreditCard USB Stick wit the latest version of Lazarus:
Including mac Catalina, KbmwMemtable for Lazarus and TMSWebcore Trial version for Lazarus.

Program day 1 : November 2019 - 09.30 – 17.30
Holger Klemt:
IBExpert.com will show how to use Lazarus in combination with a Firebird databases and how to create a PAS2JS application with access to a Firebird database.
TMSWebcore 1.3 will be available as trial version and we will build a simple project with it.

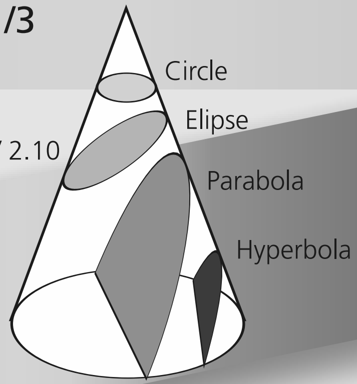
Mattias Gärtner:
WebAssembly: what is it, how to use. We will create a small program with it. Using PAS2JS with or without TMSWebcore together with Bootstrap etc. The past, present and future of PAS2JS.
The new version of Lazarus 2.0.6, the past, present and future of the Lazarus IDE.Tips and Tricks for using the Lazarus IDE with a focus on experienced Delphi developers: How to write code that can be compiled in both worlds with basic examples.How to convert existing Delphi code. The new Mac version for Lazarus 2.0.6: Catalina In the evening we will have an informal dinner for all participants that want to come

Program day 2 : Lazarus Advanced topics and components for Delphi and Lazarus developer 30 November 2019 - 09.30 – 17.30
Holger Klemt: Basic database programming in Lazarus with SQL DB, IBDAC and other components
Anton Vogelaar: Bluetooth connections based on Google API for Android (Chrome) and PAS2JS components
Detlef Overbeek: KBMmwMemtable Change tool program: the only ClientDataset with SQL search.
Bruno Fierens:
TMS Software: Crossplattform Components for Lazarus, Delphi and
Webcore news about Webcore version 1.3 and other TMS products for Lazarus
Detlef Overbeek: The Lazarus factory:
What we can do for your company and your personal Lazarus know-how?
Modular public Lazarus training's and individual Workshops in your company
Helping hands and Hotline Service
The new Lazarus book
Installation and working with FastReport 6.0 in Lazarus

THE DISTANCE OF TWO PARABOLAS PAGE 1/3
BY DAVID DIRKSE

starter expert

DX RIO Laz 2.0.6 / 2.10



INTRODUCTION

In mathematics, a parabola is a plane curve which is mirror-symmetrical and is approximately U shaped. It fits several other superficially different mathematical descriptions, which can all be proved to define exactly the same curves.

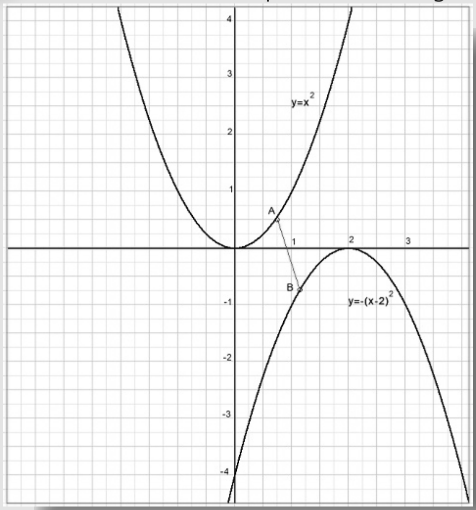
One description of a **parabola** involves a **point** (the focus) and a line (the **directrix).** The focus does not lie on the directrix. *The parabola is the locus of points in that plane that are equidistant from both the directrix and the focus.*
Another description of a parabola is as a conic section, *created from the intersection of a right circular conical surface and a plane which is parallel to another plane that is tangential to the conical surface.* (see right top)

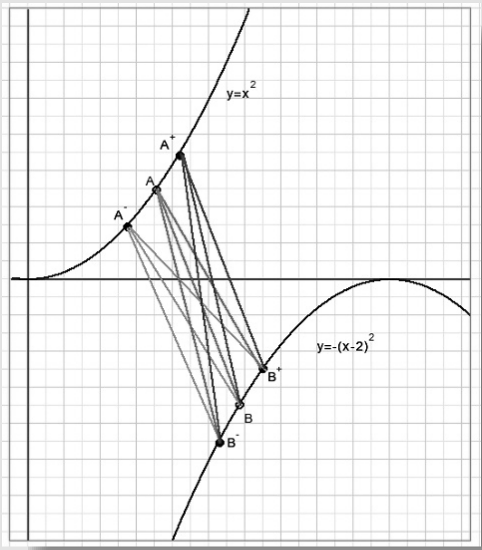
The line perpendicular to the directrix and passing through the focus (*that is, the line that splits the parabola through the middle*) is called the **"axis of symmetry"**. The point where the parabola intersects its axis of symmetry is called the **"vertex"**, and is the point where the parabola is most sharply curved. The distance between the vertex and the focus, measured along the axis of symmetry, is the **"focal length"**.
The **"latus rectum"** is the chord of the parabola which is parallel to the directrix and passes through the focus. Parabolas can open up, down, left, right, or in some other arbitrary direction. Any parabola can be repositioned and rescaled to fit exactly on any other parabola - that is, all parabolas are geometrically similar.

Parabolas have the property that, if they are made of material that reflects light, then light which travels parallel to the axis of symmetry of a parabola and strikes its concave side is reflected to its focus, regardless of where on the parabola the reflection occurs. Conversely, light that originates from a point source at the focus is reflected into a parallel *("collimated")* beam, leaving the parabola parallel to the axis of symmetry. The same effects occur with sound and other forms of energy. This reflective property is the basis of many practical uses of parabolas.

The parabola has many important applications, from a parabolic antenna or parabolic microphone to automobile headlight reflectors to the design of ballistic missiles.

Please look at the next picture showing two parabolas:





Asked is the minimum distance AB.
In general, there are two ways to solve this problem

1. **Numerical, by approximation**
2. **Analytical, which is precise**

THE NUMERICAL APPROACH
This was more difficult as expected because of the two variable points A and B. After some false starts, the following algorithm emerges.

❶ Choose A on the first and B on the second parabola in a somewhat arbitrary way, say A(1,1) and B(1,-1).
❷ Set a step value to 0.1 (not too large, not too small)
❸ Calculate distance AB.
❹ Repeat the next part
❺ Vary the x coordinates of A and B by **+step** and **-step**
❻ Now calculate all distances for the 8 cases **AB-, AB+, A-B, A- B-, A-B+, A+B, A+B-, A+B+**
❼ Save the shortest distance and also the corresponding x coordinates of A and B

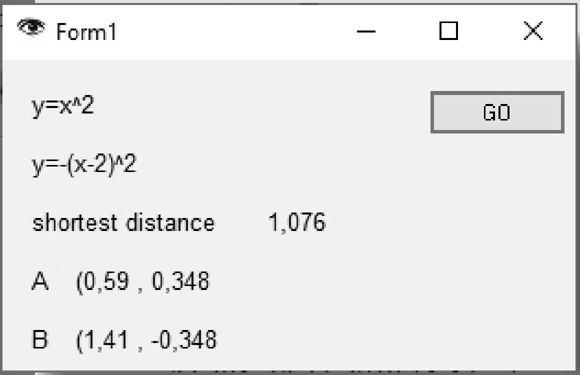
If shorter distance was found:
replace x coordinates of A and B by x coordinates of shortest distance
else divide step by 2.
Until step is very small.

The program

```
type TX = record
        a : single;//A x coord
        b : single;//B x coord
       end;

function Yvalue(fnr : byte; x : single) : single;
//central place for function values;
//fnr 1: y=x^2 //fnr 2: y=-(x-2)^2
begin
 case fnr of
  1 : result := sqr(x);
  2 : result := -sqr(x-2);
 end;
end;
```

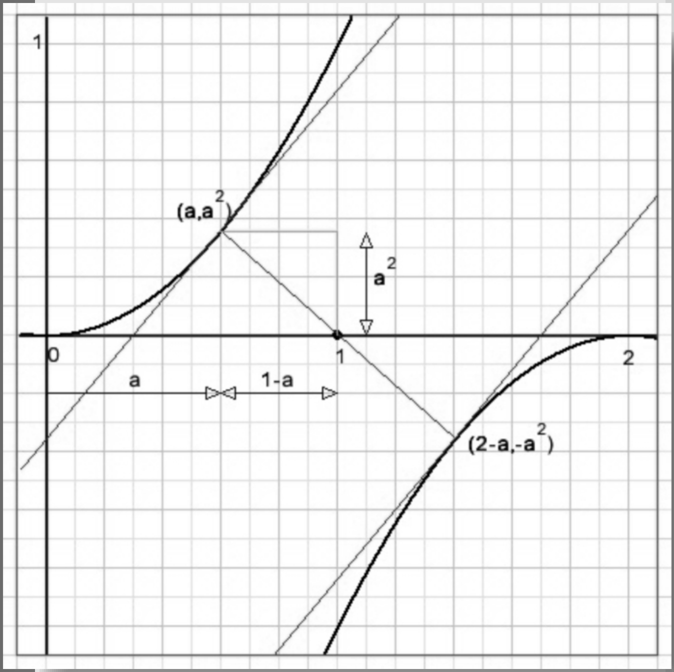
```
procedure TForm1.Button1Click(Sender: TObject);
const fs ='##0.0##';
var i : byte; s  : string;
   D,newD,step : single;
   CX,dX,saveX : TX; //Center X , offset X , saved X of A and B
   AY,BY : single; //Y coords of A,B
   modified : boolean;
begin
 decimalseparator := '.';
 CX.a := 1;
 CX.b := 1;
 step := 0.1;
//
 repeat
  modified := false;
  AY := Yvalue(1,CX.a);
  BY := Yvalue(2,CX.b);
  D := sqr(CX.a-CX.b) + sqr(AY-BY);
  for i := 1 to 8 do
   begin
    case i mod 3 of
     0 : dX.a := CX.a;
     1 : dX.a := CX.a-step;
     2 : dX.a := CX.a+step;
    end;
//
    case i div 3 of
     0 : dX.b := CX.b;
     1 : dX.b := CX.b-step;
     2 : dX.b := CX.b+step;
    end;
//
   AY := Yvalue(1,dX.a);
   BY := Yvalue(2,dX.b);
   newD := sqr(dX.a - dX.b) + sqr(AY - BY);
   if newD < D then begin
                D := newD;
                saveX := dX;
                modified := true;
               end;
   end;//for i
  if modified then CX := saveX
   else step := step/2;
 until step < 1e-6;
//
 s := '('+formatfloat(fs,CX.a);
 s := s + ' , '+formatfloat(fs,Yvalue(1,CX.a));
 Acoords.Caption := s;
 s := '('+formatfloat(fs,CX.b);
 s := s + ' , '+formatfloat(fs,Yvalue(2,CX.b));
 Bcoords.Caption := s;
 ABlabel.Caption := formatfloat(fs,sqrt(D));
end;
```

This is the result



```
y=x^2

y=-(x-2)^2

shortest distance      1,076

A   (0,59 , 0,348

B   (1,41 , -0,348
```

THE ANALYTICAL SOLUTION.



Start with coordinates (a,a2) on the left parabola.
The tangent in A has a slope of 2a .... { y'=2x }
Draw line through A and (1,0) to point B.
The slope of the tangent in B is the same for reasons of symmetry.
AB is the minimum distance if tangent and AB are perpendicular.
This is the case when the product of their slopes equals -1

$$2a.\frac{-a^2}{1-a} = -1$$

$$2a^3 + a - 1 = 0 \quad \rightarrow a^3 + \frac{1}{2}a - \frac{1}{2} = 0$$

The general solution of this type of cubic equation is

$$y^3 + py + q = 0$$

$$A = \frac{-q}{2} - \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}$$

$$B = \frac{-q}{2} + \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}$$

$$y = \sqrt[3]{A} + \sqrt[3]{B}$$

Note: A and B in above picture are local values only.
Note: for y read a.
For p=0.5 and q=-0.5 we find:

$$A = -0.00909$$
$$B = 0.5091$$

$$a = 0.583$$

$$\text{distance} = 2\sqrt{(1-a)^2 + a^4} = 1.076$$

Apple shop in Amsterdam

APPLE IS TRYING TO KILL WEB TECHNOLOGY

The company has made it extremely difficult to use web-based technology on its platforms, and it hopes developers won't bother

Article by Owen Williams
https://onezero.medium.com/apple-is-trying-to-kill-web-technology-a274237c174d

The programming languages used to build the web often find their way into apps, too.
That's largely due to software that allows developers to "reuse" the code they write for the web in products they build to run on operating systems like Linux, Android, Windows, and macOS.

But Apple has a reason not to like this recycling of web technology. **It wants its Mac App Store to be filled with apps that you can't find anywhere else, not apps that are available on every platform.**
With a recent policy change, the company has made it a little more difficult for developers to submit apps containing web code.

The Mac App Store has quietly started rejecting apps made with a popular tool called **Electron** that allows developers to base all of their apps on the web-based code.
Some of the most popular apps in the App Store, like **Slack, Spotify, Discord,** and **WhatsApp,** fall into this category.

In a discussion on the programming community Github, several developers say rejections for apps that they built using Electron - which would were approved in the past - came with an explanation that these apps "attempt to hide the use of private APIs," which are APIs built for Apple's internal usage, rather than for third-party developers.

Using private APIs to build public-facing apps is commonly frowned upon because they may change or break over time, and Apple bans apps that use them.

Electron has used these **private APIs for years** without issue. *These private APIs allow developers to, for instance, drastically improve power usage whereas Apple's sanctioned tools make the user experience worse.*

In the majority of these cases, Apple doesn't provide real alternatives for developers who want to access these private API features.

Now it's unlikely that the thousands of developers who have built their apps using **Electron** can release updates to them unless the Electron
framework releases a major change to its implementation.

Developers could distribute their apps from their own websites, **asking users to download them directly**.
But that means abandoning features like Apple's auto-update mechanism from the Mac App Store and iCloud sync. And this direct-to-consumer method could soon be locked down, too, with Apple's controversial notarization requirements potentially requiring their review.

Apple has a history of stunting the web's progress on its platforms. **On iOS, Apple doesn't allow fully independent third-party browsers, requiring all apps to leverage its Safari browser when rendering web-based content.**

WHILE BROWSERS LIKE CHROME AND OPERA ARE AVAILABLE IN THE APP STORE, THEY MUST USE APPLE'S SAFARI BROWSER BEHIND THE SCENES TO RENDER WEB PAGES, RATHER THAN THEIR OWN.

That means Apple has a monopoly on how iPhone and iPad users access the web.
To push developers toward building native apps on iOS rather than using web technologies, Apple ignores popular parts of the open web specification that other browsers implement, to its own benefit.

Apple's subtle, anti-competitive practices don't look terrible in isolation, but together they form a clear strategy.

A technology called WebRTC, for example, allows video calling in a web browser without additional software. It powers tools like Google Meet.

But Apple was incredibly slow to implement the specification, leaving out key pieces of functionality, and the technology didn't work when embedded inside apps.

Apple also handicapped an emerging standard called Progressive Web Apps (PWAs)
- which, like Electron, allows developers to build native-like apps for both desktop and mobile -
by partially implementing it in a way that makes it too inconsistent to rely on.

PWA DOESN'T HAVE THE SAME PROBLEM IF USERS OPEN APPS IN CHROME OR FIREFOX, BUT IPHONE AND IPAD USERS CAN'T INSTALL THIRD-PARTY BROWSERS, WHICH MAKES PWA-BASED TECHNOLOGY A NON-STARTER.

Developers use technologies like **Electron** and **PWA** because they allow for faster updates across platforms without an array of different codebases.

Some argue that this results in lower quality apps, but I'd argue the alternative is no app at all or apps that are rarely updated because maintaining unique Windows, Mac, and web-based products is complex and expensive.

Apple recently launched a competing framework called **Catalyst,** which allows developers with iPad apps to bring them to macOS quickly - a great tool for developers exclusively targeting Apple users, but not those building cross-platform apps.

Apple's subtle, anti-competitive practices don't look terrible in isolation, but together they form a clear strategy:
MAKE IT SO PAINFUL TO BUILD WITH WEB-BASED TECHNOLOGY ON APPLE PLATFORMS THAT DEVELOPERS WON'T BOTHER.

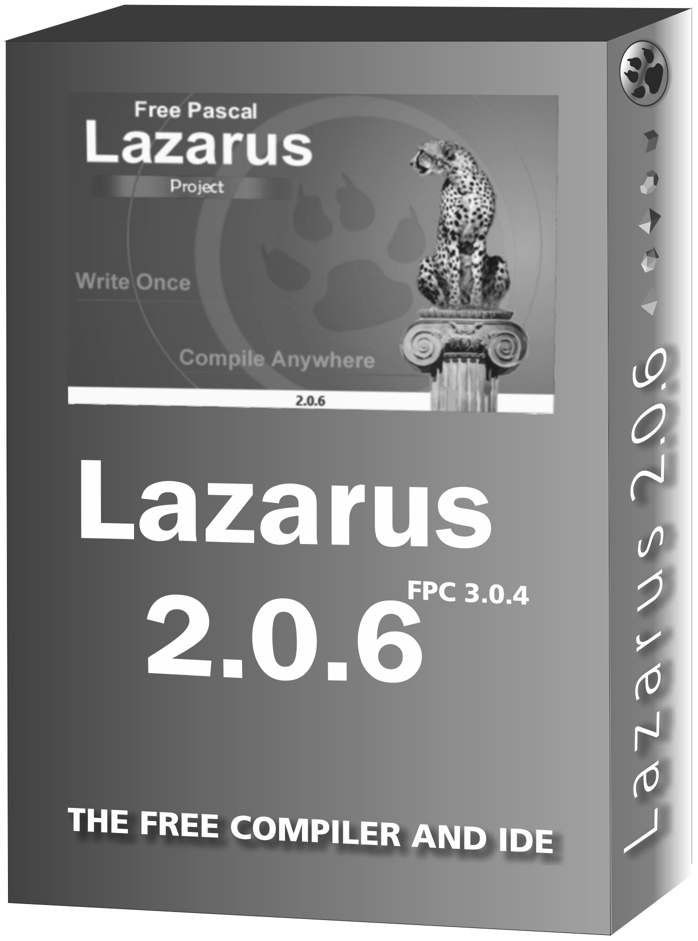
Now that the App Store is not accepting apps built using Electron, developers will likely find creative ways to work around it, but Apple is setting up for a continual cat-and-mouse game as it plans to exert more control over which apps can run on the platform in the future.

These types of changes may be made in the name of privacy or security, but the reality is that the argument looks weak when both users and developers simply don't have a choice because Apple controls the platform, browser engine, and the distribution method.

Regardless of your opinion of **Electron** app quality, choice is important.

Apple's control over its app ecosystem is a new type of monopoly that's hard to understand for lawmakers, and difficult for us to fight back against - because there simply isn't a way out of these restrictions when the company controls both the distribution method and the platform itself.

The Lazarus Factory



LAZARUS IS A DELPHI COMPATIBLE CROSS-PLATFORM IDE FOR FREE PASCAL.
It includes LCL which is more or less compatible with Delphi's VCL. Free Pascal is a GPL'ed compiler that runs on Linux, Win32, OS/2, 68K RasberryPie and more. Free Pascal is designed to be able to understand and compile Delphi syntax, which is OOP. Lazarus is the part of the missing puzzle that will allow you to develop Delphi like programs in all of the above platforms.

WHAT WIDGET SET?
You decide. Lazarus is being developed to be totally and completely API independent. Once you write your code you just link it against the API widget set of your choice. If you want to use GTK+, great! If you want it to be Gnome compliant, great! As long as the interface code for the widget set you want to use is available you can link to it. If it isn't available, well you can write it.

CAN YOU USE YOUR EXISTING DELPHI CODE?
IN GENERAL: YES. If you are using some very specific databases, OCX, or DCU then the answer would be no. THESE ITEMS ARE SPECIFIC TO WINDOWS AND WOULD ONLY WORK ON AND WITHIN WINDOWS.

CAN I CREATE COMMERCIAL PRODUCTS WITH THIS?
YES. The code for the Free Pascal compiler is licensed under the GPL.

The Lazarus Factory

WHAT DOES THE LAZARUS FACTORY?

1. SUPPORTING COMPANY'S AND USERS ON A COMMERCIAL BASIS
2. DEVELOPING AND PROVIDING COURSES AND INSTRUCTION METHODS
3. CONTRIBUTING FINANCIALLY AND TECHNICALLY TO FREEPASCAL AND LAZARUS

- LAZARUS COMPILES FOR THE WEB USING PAS2JS
- LAZARUS HAS EMBEDDED WEBASSEMBLY
- LAZARUS NOW HAS GENERICS: FOR FPC, LAZARUS AND PAS2JS
- LAZARUS CAN COMPILE TO MULTIPLE PLATFORMS: WINDOWS / LINUX / MAC RASBERRY AND MORE
- LAZARUS USES NATIVE COMPONENTS
- LAZARUS CAN BE USED TO COMPILE DELPHI CODE. IF NOT LAZARUS FACTORY CAN HELP
- LAZARUS HAS A HANDBOOK OF ABOUT 1000 PAGES.

The Lazarus Factory

THE FREE COMPILER AND IDE

The Lazarus Factory

starter expert

A SHORT INTRODUCTION:

BufDataSet in two versions with simple Buttons and color Buttons and one as CDS file for Delphi) I think it's a good idea to have another version under Lazarus based on the Free kbmMemTable Standard Edition.

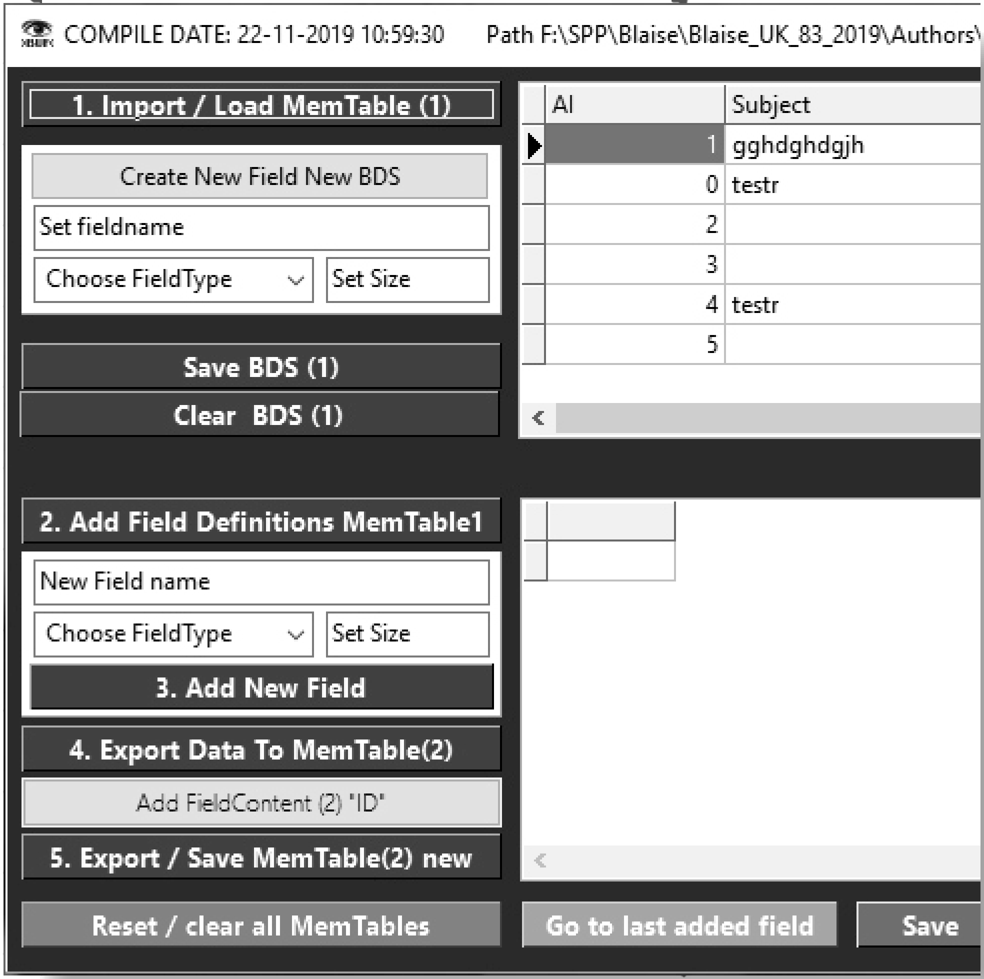
This version of kbmMemTable Standard Edition has been build as a free version for Lazarus and is of course also separately available. You will have the full code of this project and can download that in two versions; We created a Lazarus version which has this Memtable build-in together with other extra software: Tms Webcore 1.3 and Colorbutton. They are pre-installed. You also can download the kbmMemTable Standard Edition also to install it in to your existing version of Lazarus.

FUNCTIONALITY

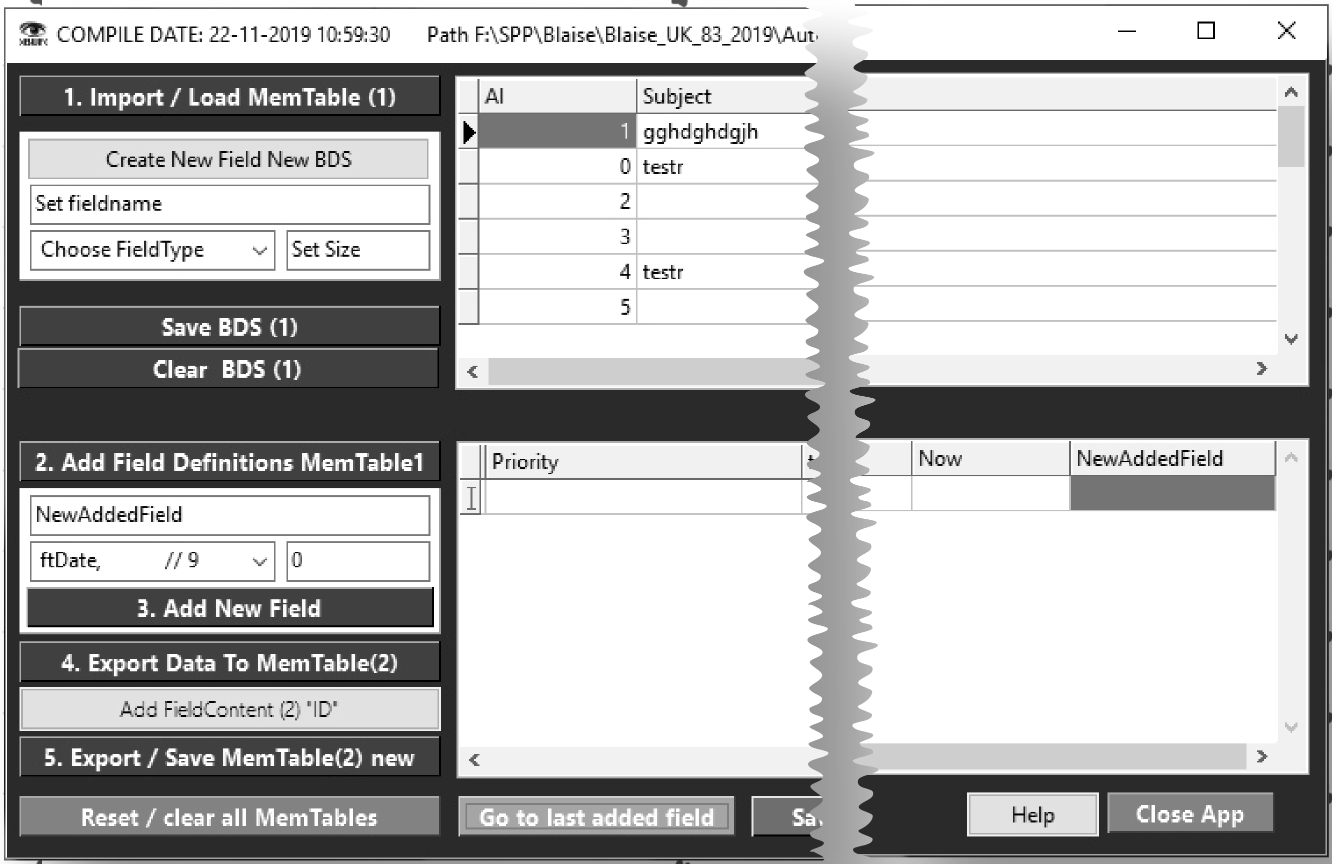
The main functions are in no way different from the other functions in the Change Tools. So we do not add any explanation to that, you can find the user descriptions in issue 81/82. The code for it self is somewhat different: Memtable has an other way of handling code. You have the complete files. The big advantage of Memtable is that you can use SQL instead of Filtering. The standard version is free: the professional version cost only a very small amount and is absolutely worth having it: you then will have the code as well.

In this short article I pasted some images to make sure you do understand what I talk about and behind that you find the code in details. The fun of this all is that you can alter a existing table, which is normally not possible. Hope you will enjoy it.

Have a look at the special advertisment at page 58: Black Friday.


COMPILE DATE: 22-11-2019 10:59:30
Path F:\SPP\Blaise\Blaise_UK_83_2019\Authors\
1. Import / Load MemTable (1)
Create New Field New BDS
Set fieldname
Choose FieldType
Set Size
Save BDS (1)
Clear BDS (1)
2. Add Field Definitions MemTable1
New Field name
Choose FieldType
Set Size
3. Add New Field
4. Export Data To MemTable(2)
Add FieldContent (2) "ID"
5. Export / Save MemTable(2) new
Reset / clear all MemTables
Go to last added field
Save

AI	Subject
1	gghdghdgjh
0	testr
2	
3	
4	testr
5	

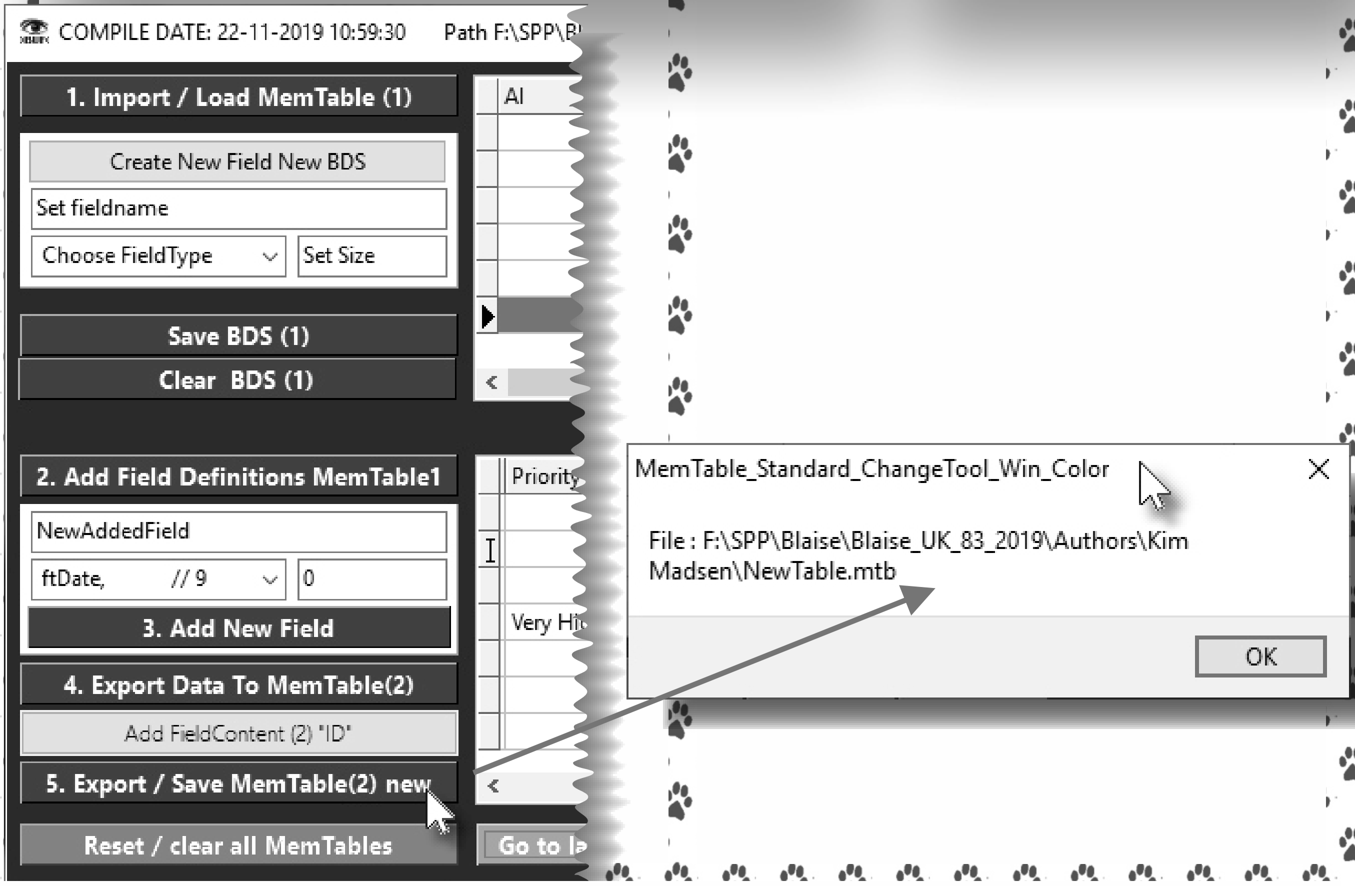

COMPILE DATE: 22-11-2019 10:59:30 Path F:\SPP\Blaise\Blaise_UK_83_2019\Aut

AI	Subject
1	gghdghdgjh
0	testr
2	
3	
4	testr
5	

Priority		Now	NewAddedField

Set fieldname
Choose FieldType
Set Size
NewAddedField
ftDate, // 9
0

1. Import / Load MemTable (1)
Create New Field New BDS
Save BDS (1)
Clear BDS (1)
2. Add Field Definitions MemTable1
3. Add New Field
4. Export Data To MemTable(2)
Add FieldContent (2) "ID"
5. Export / Save MemTable(2) new
Reset / clear all MemTables
Go to last added field
Help
Close App


COMPILE DATE: 22-11-2019 10:59:30 Path F:\SPP\B

Set fieldname
Choose FieldType
Set Size
NewAddedField
ftDate, // 9
0

MemTable_Standard_ChangeTool_Win_Color

File : F:\SPP\Blaise\Blaise_UK_83_2019\Authors\Kim Madsen\NewTable.mtb

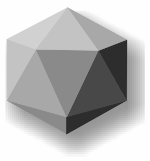
AI
Priority
Very Hi
1. Import / Load MemTable (1)
Create New Field New BDS
Save BDS (1)
Clear BDS (1)
2. Add Field Definitions MemTable1
3. Add New Field
4. Export Data To MemTable(2)
Add FieldContent (2) "ID"
5. Export / Save MemTable(2) new
Reset / clear all MemTables
Go to la
OK

```
 unit uGUI;
{$mode objfpc}{$H+}
interface
uses
 Classes, SysUtils, db, Forms, Controls, Graphics, Dialogs,
 ExtCtrls, Buttons, StdCtrls, DBGrids, DBCtrls, HSButton, kbmMemTable,
 kbmMemBinaryStreamFormat, kbmMemCSVStreamFormat, kbmMemSQL;
type

 { TGUI }

 TGUI = class(TForm)
...
   procedure BtnCreateNewFieldClick(Sender: TObject);
...
 private
 public

 end;

var
  GUI: TGUI;

  AddFieldName, NwFieldName         : string;
  AddFieldKind, NwFieldKind         : TFieldType;
  AddFieldSize, NwFieldSize         : Integer;
  ColumnLast                        : Integer;
  ColumTitleString, MemTable_Dataset_Filename : String;

implementation
{$R *.lfm}
{ TGUI }

procedure TGUI.BtnCreateNewFieldClick(Sender: TObject);
begin
 kbmMemTable1.Close;
 CLrBtnImportLoad.Enabled:= False; // Dependant on choice 1 new datset
 GUI.ShowHint := False;

 if Edit1.Text <= '' then    // Check if Edit1 contains text
  begin
   Showmessage('You must enter a name for the field');
   BtnCreateNewField.Enabled:=False;
  end
 else
  begin
   BtnCreateNewField.Enabled := True;
   if ComboBox1.Text <= '' // Check if Combobox contaisn a field type
   then Showmessage('Please select a field kind')
   else  // Check if EdtAddField contains teh field size
   If Edit1.Text <= '' then Showmessage('Please select a field size');
   // The boolean set at the end is already set to true
   // It is required by the syntax
  end;

 NwFieldName := Edit1.Text;
 NwFieldKind := TFieldType(ComboBox1.ItemIndex);
 NwFieldSize := StrToInt(EdtAddField.Text);

 kbmMemTable1.FieldDefs.Add (NwFieldName, NwFieldKind, NwFieldSize, True);
 // kbmMemTable2.FieldDefs.Add ('NAAM',   ftString,    20,          True);



 kbmMemTable1.CreateTable;
end;
```

```
procedure TGUI.BtnHelpOnOffClick(Sender: TObject);
begin
 Panel4.Visible := Not Panel4.Visible ;
end;

procedure TGUI.BtnIdxClick(Sender: TObject);
Var I : Integer;
Begin
  I := 1;
  kbmMemTable2.First;
  While Not kbmMemTable2.Eof Do
  Begin
    kbmMemTable2.Edit;
    kbmMemTable2.FieldByName ('ID').AsInteger := I;
    kbmMemTable2.Post;
    kbmMemTable2.Next;
    Inc (I);
  End;
End;

procedure TGUI.ButtonLoadClick(Sender: TObject);
begin
  kbmMemTable1.EmptyTable; // kbmMemTable1.Clear;
end;

procedure TGUI.ClrBtnAddNewFieldClick(Sender: TObject);
begin
 kbmMemTable2.Close;
 kbmMemTable2.Fields.Clear;
 AddFieldName :=  EdtAddField.Text;                // reeds the fieldname
 AddFieldKind :=  TFieldType(ComboBox2.ItemIndex);  // reeds the fieldkind by the index number of the combobox
 AddFieldSize :=  StrToInt(EdtAddSize.Text);        // reeds the fieldsize
 // The simple way: KbmMemtable.FieldDefs.Add ('ID',   ftInteger, 0, False);
 kbmMemTable2.FieldDefs.Add (AddFieldName, AddFieldKind, AddFieldSize, False);
 kbmMemTable2.active:= true;
end;

procedure TGUI.ClrBtnClearClick(Sender: TObject);
begin
 kbmMemTable1.ClearFields;
end;

procedure TGUI.ClrBtnAddFieldDefinitionClick(Sender: TObject);
begin
 kbmMemTable2.FieldDefs.Assign (kbmMemTable1.FieldDefs);
  kbmMemTable2.Active:= true;
end;

procedure TGUI.ClrBtnExportBDS2Click(Sender: TObject);
var I : Integer ;
begin
 DBGrid1.Visible := False;
 DBGrid2.Visible := False;
 kbmMemTable1.First;

 While Not kbmMemTable1.Eof Do
 Begin
  kbmMemTable2.Insert;
  For I := 0 To kbmMemTable1.FieldCount - 1 Do kbmMemTable2.Fields [I].Assign (kbmMemTable1.Fields [I]);
  kbmMemTable2.Post;
  kbmMemTable1.Next
 End;

 DBGrid1.Visible := True;
 DBGrid2.Visible := True;
end;
```

```
procedure TGUI.ClrBtnExport_SaveClick(Sender: TObject);
begin
 if SaveDialog2.Execute then  ShowMessage('File : ' + SaveDialog2.FileName);
 kbmMemTable2.SaveToFile(SaveDialog2.FileName);
end;

procedure TGUI.ClrBtnFirstClick(Sender: TObject);
begin
 With kbmMemTable2 do
 if active then
  begin
 //   showmessage('active');
    First
  end
 else
  begin
  //  showmessage(' not active');
    exit;
  end
end;

procedure TGUI.ClrBtnNextClick(Sender: TObject);
begin
  With kbmMemTable2 do
 if active then
  begin
  //  showmessage('active');
    Next
  end
  else
   begin
  //  showmessage(' not active');
    exit;
  end
end;

procedure TGUI.ClrBtnPriorClick(Sender: TObject);
begin
 With kbmMemTable2 do
 if active then
  begin
  //  showmessage('active');
    Prior
  end
  else
   begin
  //  showmessage(' not active');
    exit;
  end
end;

procedure TGUI.ClrBtnLastClick(Sender: TObject);
begin
 With kbmMemTable2 do
 if active then
  begin
  //  showmessage('active');
    Last
  end
  else
   begin
 //   showmessage(' not active');
    exit;
  end
end;
```

```
procedure TGUI.ClrBtnSaveClick(Sender: TObject);
begin
 if SaveDialog1.Execute then ShowMessage('File : ' + SaveDialog1.FileName);
 kbmMemTable1.SaveToFile(SaveDialog1.FileName);
end;

procedure TGUI.ClrBtnHelpCloseClick(Sender: TObject);
begin
 Panel4.Visible := False;
end;

procedure TGUI.ClrBtnSaveDatset2Click(Sender: TObject);
begin
 With kbmMemTable2 do
  begin
   Post;

   SaveToFile(MemTable_Dataset_Filename);  // global Var = extracted from OpenDialog1
  end;
end;

procedure TGUI.ClrBtnTestClick(Sender: TObject);
begin
 DBGrid2.SelectedIndex := (0);
 ColumnLast := dbgrid2.Columns.Count;
 DBGrid2.SelectedIndex := (ColumnLast -1);
 kbmMemTable2.Edit;
end;


procedure TGUI.FormCreate(Sender: TObject);
Var Compiledate : String; aDate : TDateTime;
begin
 // Compiledate:=DateTimeToStr(FileDateToDateTime(FileAge(ExtractFileName(Application.ExeName))));
 // Deprecated  so this is the best way to do it: use function Exedate

 if FileAge(Application.ExeName,adate) then
  CompileDate:=DateTimeToStr(aDate)
 else CompileDate:='?';

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

end;

procedure TGUI.ClrBtnImportLoadClick(Sender: TObject);
begin
 DBGrid1.Visible := True;
 DBGrid2.Visible := True;
 if OpenDialog1.Execute then // ShowMessage('File : ' + openDialog1.FileName);
 kbmMemTable1.LoadFromFile(openDialog1.FileName);//
 MemTable_Dataset_Filename := OpenDialog1.FileName;
end;

procedure TGUI.ClrBtnResetClick(Sender: TObject);
begin
 kbmMemTable1.Close;
 kbmMemTable2.Close;
 kbmMemTable1.Fields.clear;
 kbmMemTable2.Fields.clear;
end;

end.
```



PAINTING DASH-DOT LINES PAGE 1/4
BY DAVID DIRKSE

starter expert

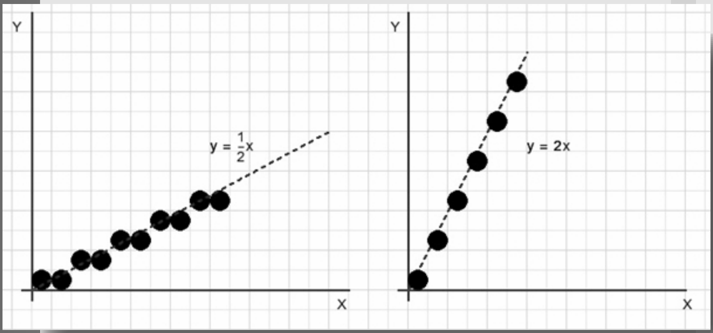
DX RIO Laz 2.0.6 / 2.10

INTRODUCTION

The Delphi programming language has the option to draw dotted lines. However, only dash-dot lines having a pen width of 1 are possible. This article describes an efficient method to overcome this limitation.

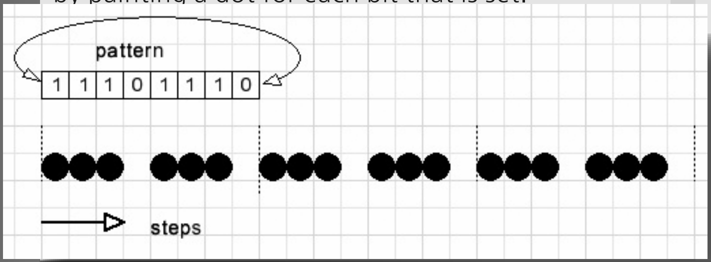
PAINTING LINES IN GENERAL

Beyond in this article we need to paint lines pixel by pixel, dot by dot.
The general function of a line crossing the origin (0,0) of a coordinate system is y = px, where p is the tangent.
In the case of p < 1 the procedure is to step x (0,1,2,3,..) while increasing y by p and painting a dot at each (x,y) coordinates.
However if p > 1 a problem arises: the line shows holes.



So, if p > 1 the function is rewritten as x = y/p and y is stepped (0,1,2,3,..) while x is incremented by steps of 1/p.
Considering steps (1,2,3,..) and increment values (p, or 1/p) without wondering which one applies to x and y, we can focus on simple horizontal lines for the next explanations.

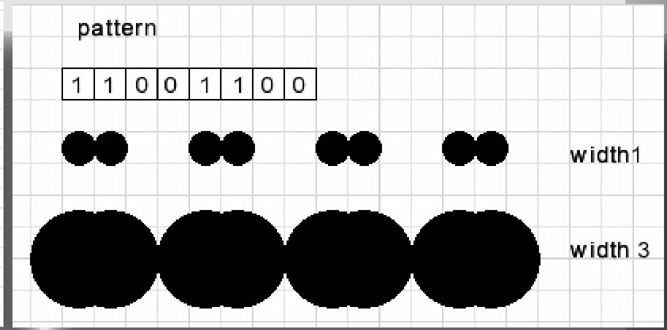
Painting dash-dot lines with a pen width of 1. The dash-dot pattern is coded as bits in a byte and we call this the linepattern. Painting a dash-dot line amounts to duplicating this pattern by painting a dot for each bit that is set.



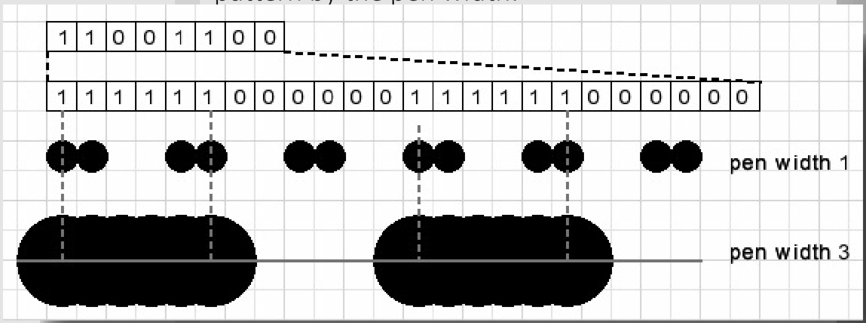
Below left is shown a very much enlarged result. To find out if a dot must be painted for step N, N mod 8 supplies the pattern bit to be examined for "1".
The value of 8 is choosen on purpose: N mod 8 is calculated simply by N and $7, extracting the least three bits of N.

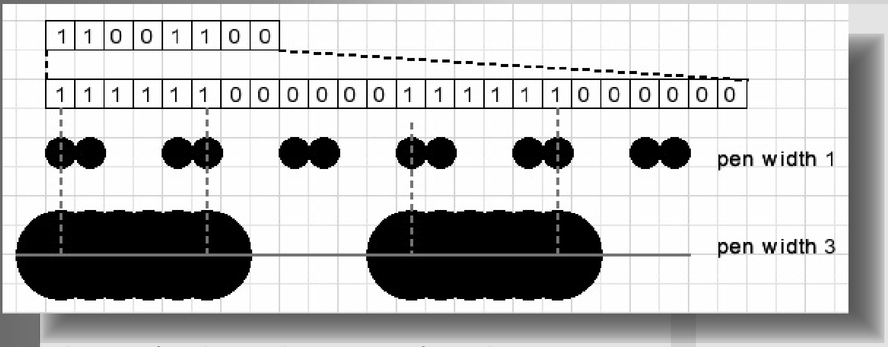
PAINTING LINES WITH HIGHER PENWIDTHS

We apply the same principle as above to paint a line using a pen width of 3.

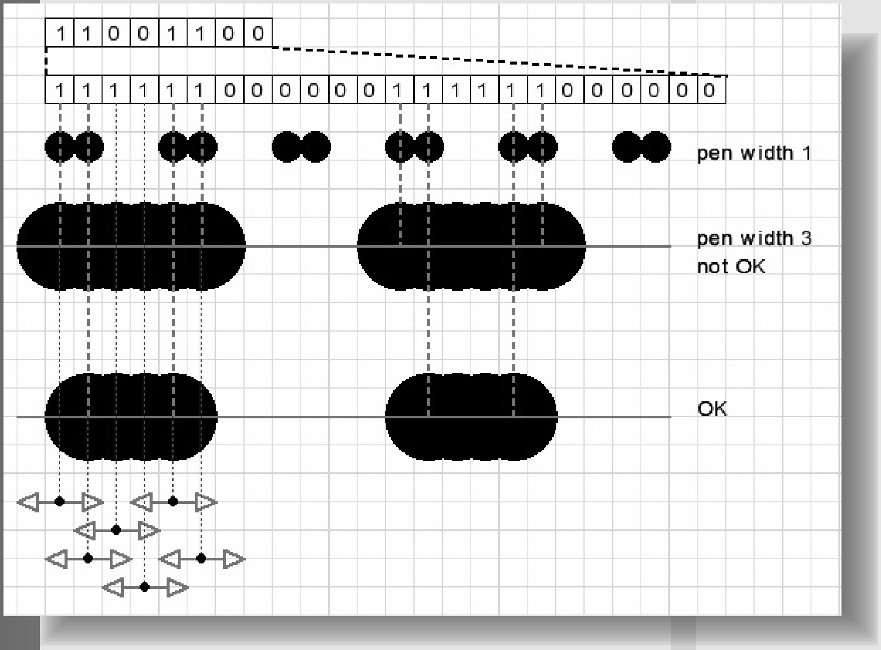


The line - space ratio is completely disturbed.
A way to overcome this problem is to stretch the pattern by the pen width:



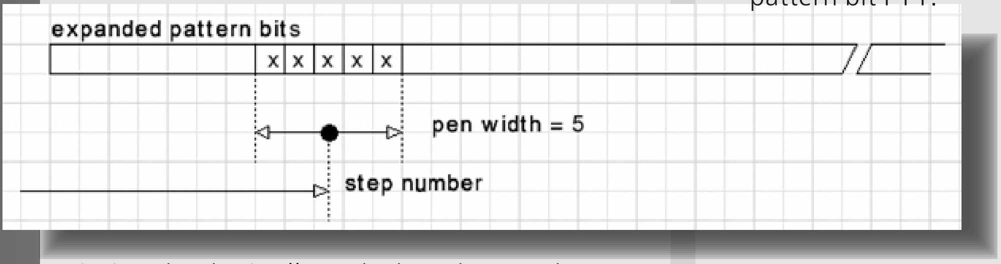


This result is better but not perfect: the empty space to line ratio is less than 1:1. The pen may not overlap empty spaces.
To solve this problem the total width of the pen is considered and dot painting is suppressed if overlap with an empty space ('0' pattern bit) occurs:

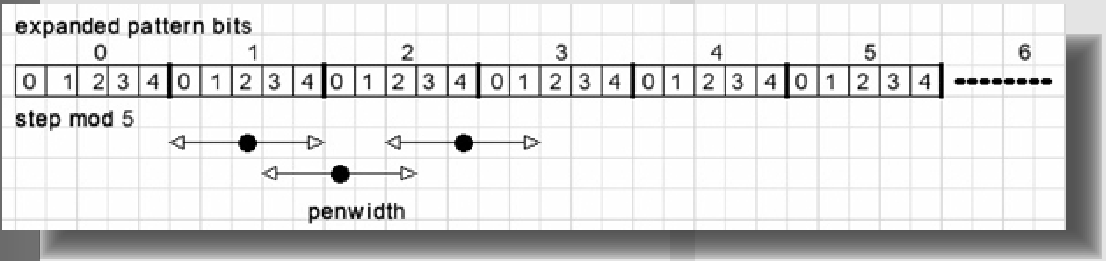


Now the result is fine which raises the question how to program this efficiently, which means without much time consuming arithmetic operations.

IMPLEMENTATION
Let's consider a penwidth of 5.
The pattern is stretched 5 times, its size is 5 x 8 = 40 bits.
The pen slides over this expanded pattern with a width of 5.



Painting the dot is allowed when the pen does not overlap any '0' pattern bit.



- In case of N (mod 5) = 2
  then just 1 pattern bit must be checked.
- If (N mod 5) = 0,1
  the previous pattern bit must be
  checked as well.
- If (N mod 5) = 3,4 the next pattern bit must be checked also.
- Numbering the groups of 5 (0,1,2,3,..) we see : group G = N div 5.
- The pattern bit P is this group number mod 8, so P = ((N div 5) mod 8)

- With 8 pattern bits and a `penwidth` of 5, after 5 x 8 = 40 steps the situation of step 0 is back, step 41 needs the same decision as step 1 etcetera.

Knowing the `penwidth` and the selected dash-dot pattern, an array of boolean may be generated that instantly shows if a dot has to be painted.

In general, having 8 pattern bits and a `penwidth` of w we have to calculate the paint decisions for steps 0..5w-1.

- In case w is odd:
  Counting the pen pixels x from 0..w-1 the middle pixel of the pen is (w-1)/2.

At step N, test pattern bit
P = ((N div w) mod 8).

- if (N mod w) = (p-1)/2 then this test is enough.
- If (N mod w) < (p-1)/2 then also check pattern bit P-1.
- If (N mod w) > (p-1)/2 then also check pattern bit P+1.

For even pen widths, a small change is necessary:

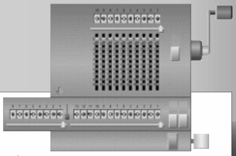
- Pen pixel position x = w/2 - 1 needs checking of 1 pattern bit.

Again, a smaller x needs checking the previous, a larger x needs checking of the next pattern bit.

Below is a table which puts all patterns with previous, present, next bits together.

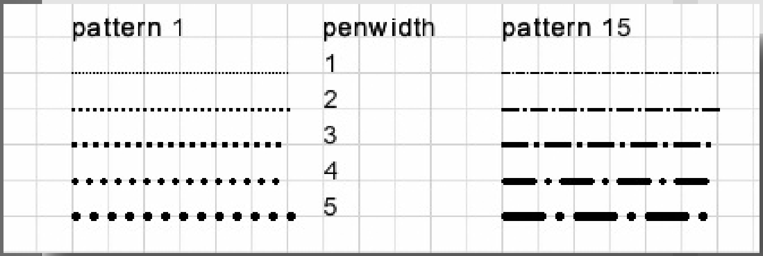
```
type Tmask = (mp,mc,mn);//previous, center,next

const maskTable : array[0..7,mp..mn] of byte =
   (($01,$80,$40),
    ($80,$40,$20),
    ($40,$20,$10),
    ($20,$10,$08),
    ($10,$08,$04),
    ($08,$04,$02).
    ($04,$02,$01),
    ($02,$01,$80);
```

Following procedure generates the 8*w boolean array DDcheck:



```
 procedure makeDashDotTable(pat,width : byte);
 //pat : pattern
 //width : pen width
 //DDcheck[] is boolean array for dash-dot paint checking
var   x : byte; //steps 0,1,2,3..
    wx : byte;// 0,1..width-1,0,1,..
  mask : byte;
 center : byte;//for check next,previous p bit
 begin
  wx := 0;
  center := (w shr 1) - ((w and 1) xor 1);
  for x := 0 to (width shl 3)-1 do
   begin
    m8 := (x div width) and $7;//step mod 8
    with masktable[m8] do
     begin
      mask := mc;
      if wx > center then mask := mask or mn;
      if wx < center then mask := mask or mp;
     end;
    DDcheck[x] := (mask and pat) = mask;
    inc(wx);
    if wx = width then wx := 0;
   end;//for x
  end;
```

Some results:

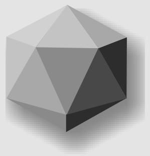


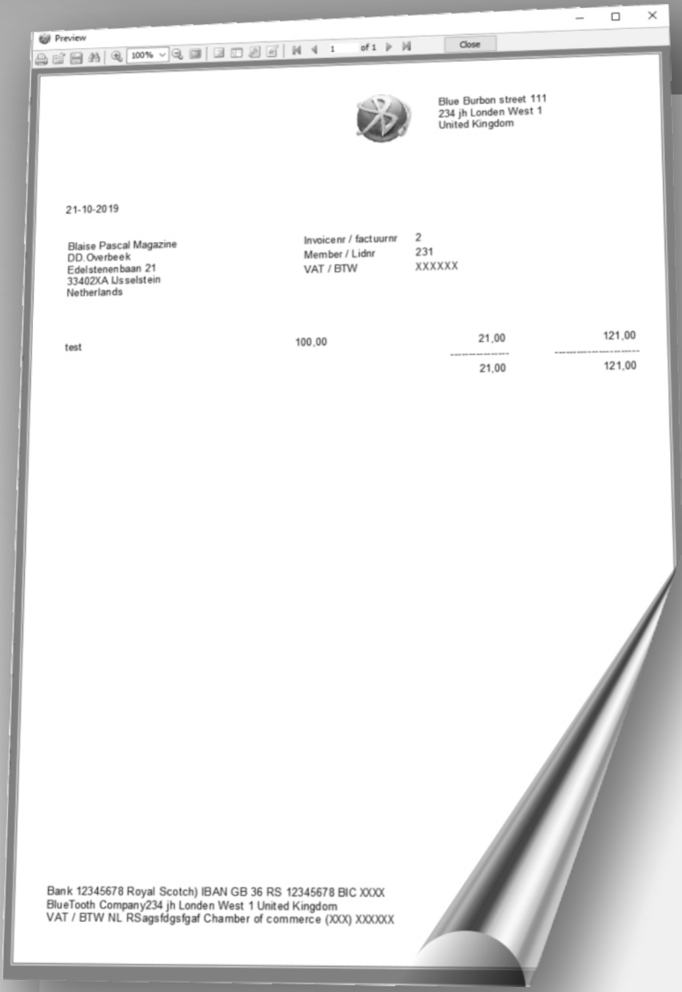
THE TXBITMAP CLASS
Above procedures are implemeted in my TXBitmap class, which is a TBitmap with many extra options.
The drawing (of all penwidths) is done by copying array FXpen[0..15] of word to the canvas.
FXpen holds a bit coded circle with the diameter of the penwidth.
Changing the penwidth builds a new FXpenwidth array and also builts a new FXddCheck array of boolean.
FddEnd holds the length of the FXddCheck array.
Selecting a new dash-dot pattern also bults a new FXddCheck array.
Now, for line drawing a step counter (ddc) is needed as index to the FXddCheck array.
The following code is the core of line drawing:

```
//....
//starting x1,y1 values calculated
//increment values dx,dy pre calculated
 ddc := 0;
 for i := 0 to steps do
  begin
   if FXddCheck[ddc] then Dot(x1,y1);
   inc(ddc);
   if ddc = FddEnd then ddc := 0;

   x := x + dx; x1 := trunc(x + 0.5);
   y := y + dy; y1 := trunc(y + 0.5);
  end;
```

For details I refer to the TXBitmap source code.
An article about all Xbitmap features is found here

This concludes the dash-dot line painting explanation.





Blue Burbon street 111
234 jh Londen West 1
United Kingdom

21-10-2019

Blaise Pascal Magazine
DD.Overbeek
Edelstenenbaan 21
33402XA IJsselstein
Netherlands

Invoicenr / factuurnr	2
Member / Lidnr	231
VAT / BTW	XXXXXX

test	100,00	21,00	121,00
		21,00	121,00

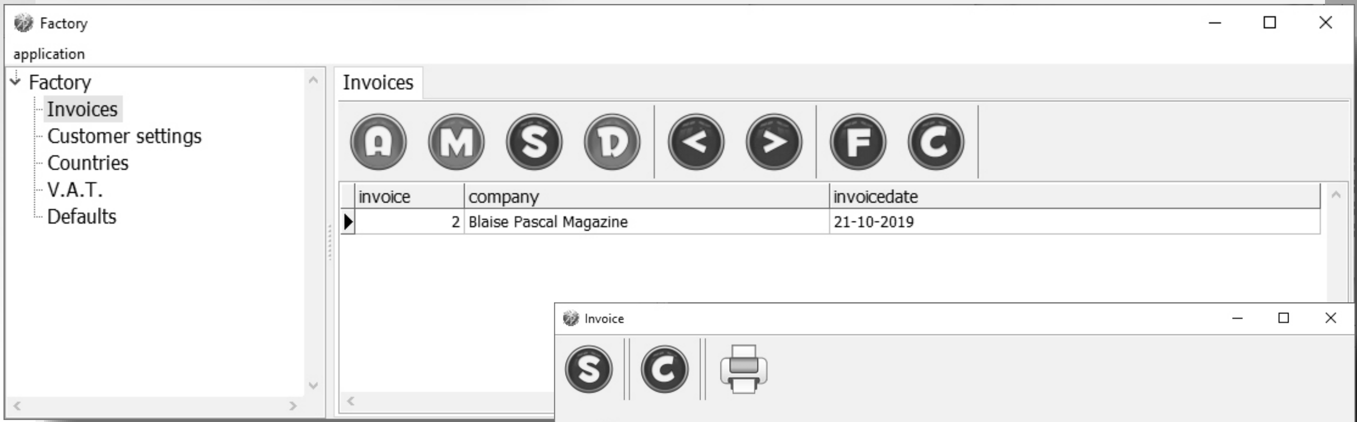
Bank 12345678 Royal Scotch) IBAN GB 36 RS 12345678 BIC XXXX
BlueTooth Company234 jh Londen West 1 United Kingdom
VAT / BTW NL RSagsfdgsfgaf Chamber of commerce (XXX) XXXXXX

starter expert

INTRODUCTION

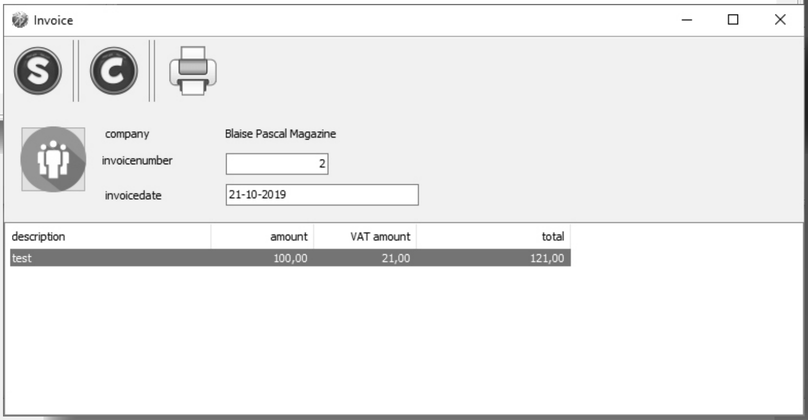
In this program you can create , administer and use several company's as sender - you can create several forms for your companies, they all can be completely different - and of course per address several invoices to your clients, with the help of the free report version of FastReport. Because I always want have maximum Simplicity of operation and Clarity is very important I want to try here to provide an example of how a GUI should function. Of course there is always room for improvement. So if you have better solutions let me know.
We start with the main form and show a diagram that shows the workflow.

Figure 1: The Invoice finalised, result from the program and the Lazarus Report version of FastReport.



invoice	company	invoicedate
2	Blaise Pascal Magazine	21-10-2019

Figure 2: The main window of the application



company	Blaise Pascal Magazine
invoicenumber	2
invoicedate	21-10-2019

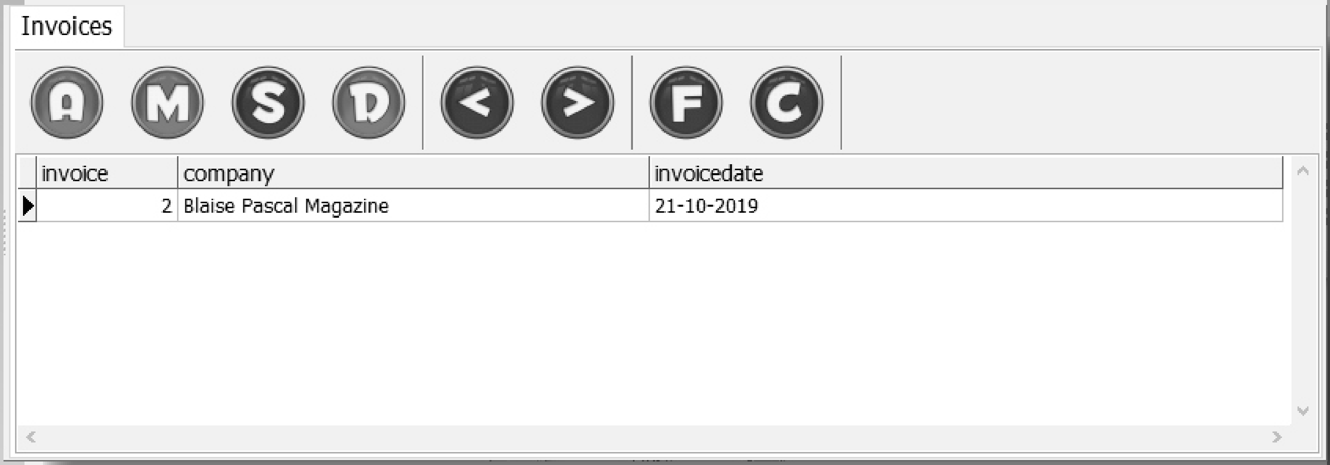
description	amount	VAT amount	total
test	100,00	21,00	121,00

THE WORKINGS

CREATING INVOICES

❶ Double click on the name **Factory.**
The tree view will open. We will follow the menu step by step

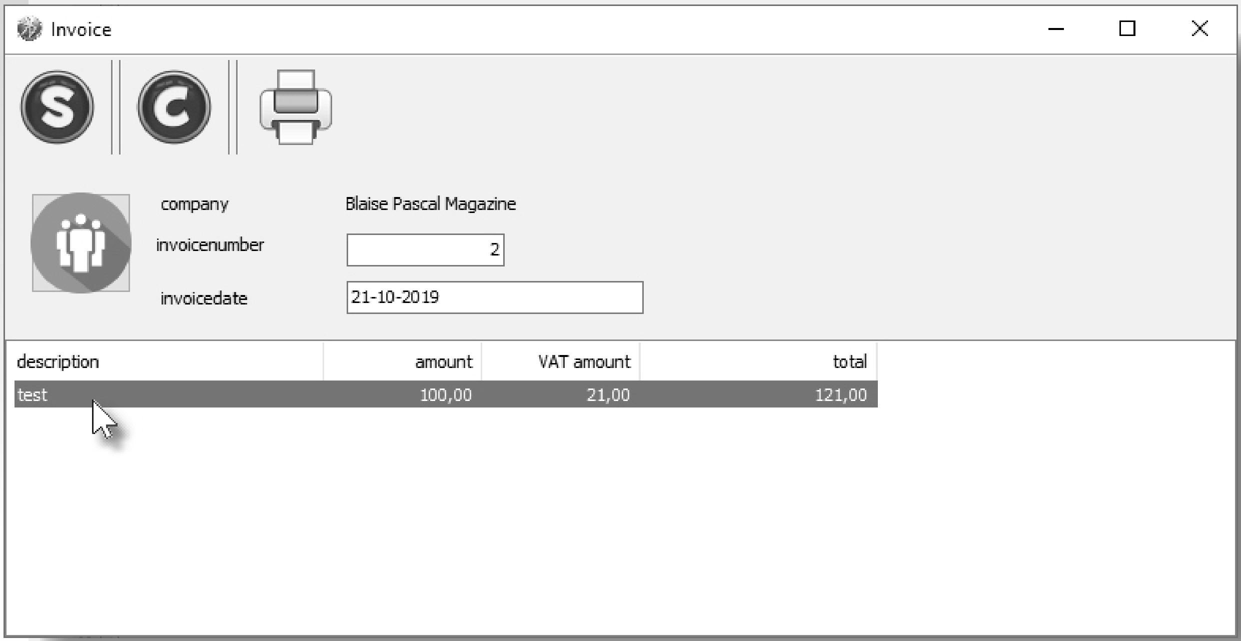
❷ In the first branch you see Invoices:
to illustrate it I have already added an invoice.



invoice	company	invoicedate
2	Blaise Pascal Magazine	21-10-2019

Figure 4: The main window.

Double click on the line Blaise Pascal Magazine. This is the address where the invoice is send to.
The next form will show: **Invoice.**
You can add a new Invoice: A,modify: M. etc. The buttons hints will guide you.



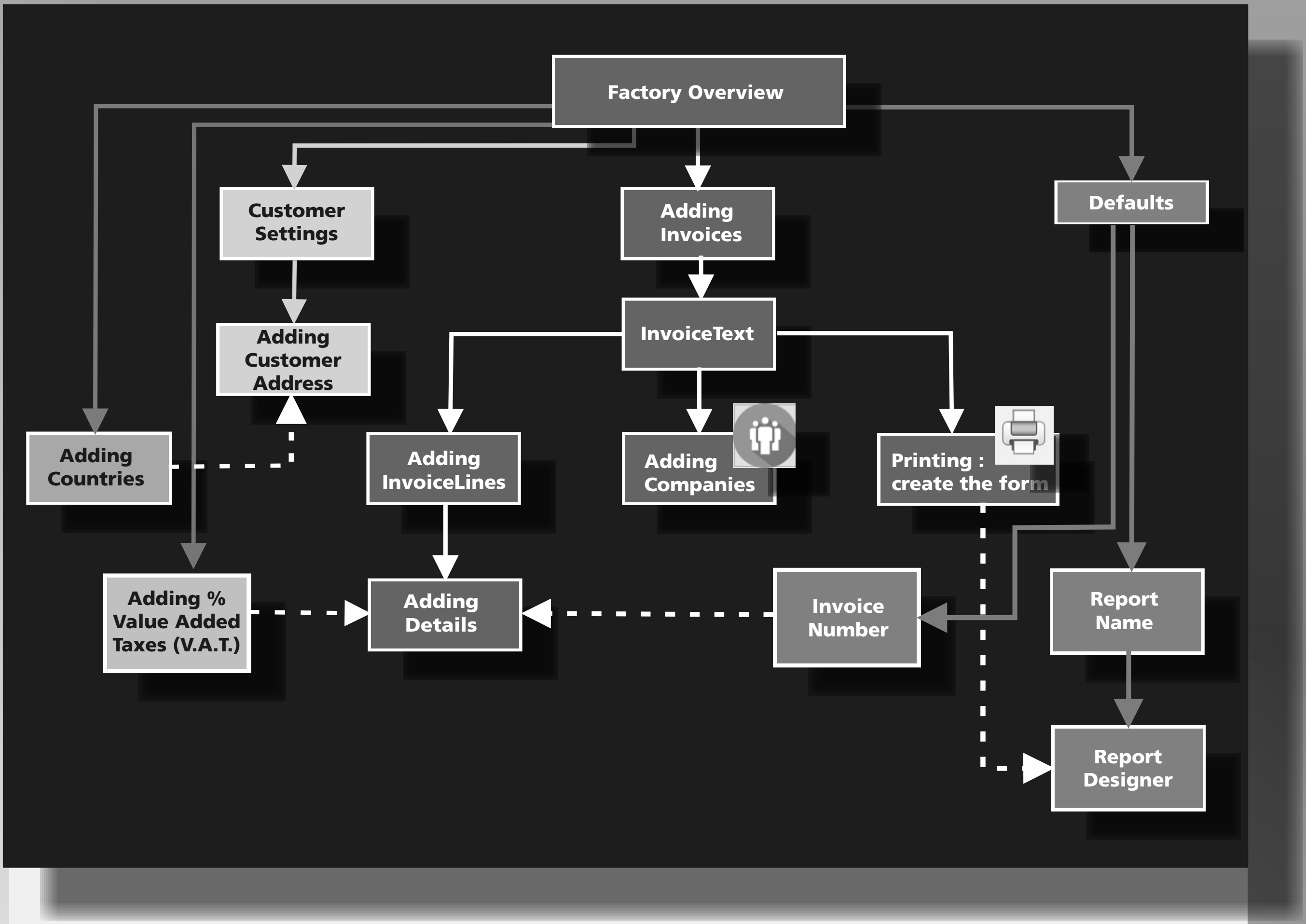
company: Blaise Pascal Magazine
invoicenumber: 2
invoicedate: 21-10-2019

description	amount	VAT amount	total
test	100,00	21,00	121,00

Figure 5: Here you can add new lines, open the company or Print

Here is what happens: A new form is opened. You can see the line description of the order.

Separated in to several other things: amount, vat and total. These details are shown later.
You need to double click that line and a form **invoice-lines** will open.



THE DIAGRAM OF THE WORKFLOW

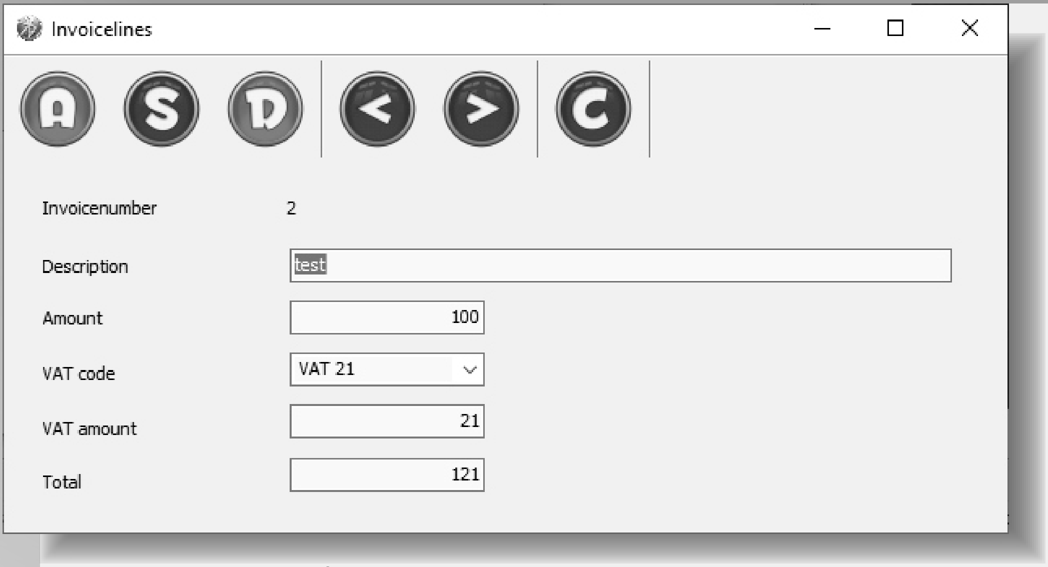
Figure 3: The diagram gives you the overview of the consistency of the program.

It shows:

1. The flow of "Adding Invoices": white.
2. Detail "Customer Settings": yellow
3. Adding "Countries": Light Blue
4. Adding "Vat": Light Green
5. Handling "Defaults": Green

First we show how the workflow is running.
The workings are explained in detail.

This is program is meant to allow you to make your own changes...
But first try to understand the workings of it. You will have the complete code, so you can alter anything.



Invoicenumber	2
Description	test
Amount	100
VAT code	VAT 21
VAT amount	21
Total	121

Figure 6: The result is this form **invoice lines** and will be explained later.

Back to the main-form Invoice:
On the form Invoice there are two options : S for Save C for Cancel. The printer symbol is of course for printing. Printing allows you to create or alter the form. Again a new form will show. The appearance is dependent on what choice you make under the form Defaults. It is explained later.
You can either choose to have your form for

1. **printing the invoice opened (see figure: 7)**
2. **or get the designer available to design your Invoice.**

You can click on the printer and the next form will come forward.

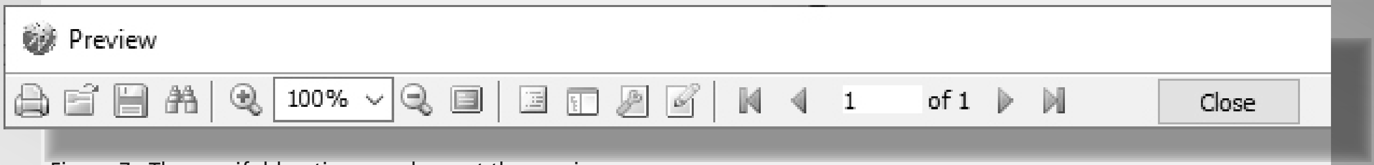
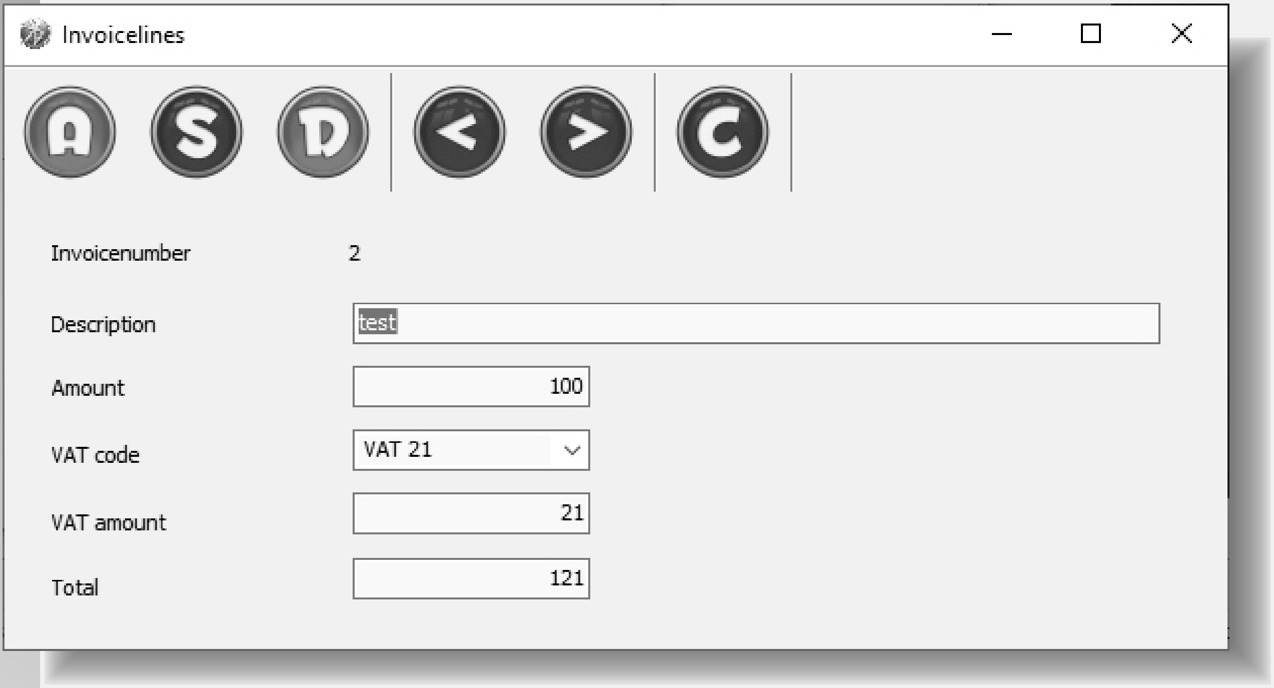


Figure 7: The manifold options you have at the preview menu...

On this form you can do a number of things that you need to find out.
From left to right:

- **print**
  if you press this button you will be presented
  a sub menu where you can fine tune your
  printer, printer settings etc.
  The format is PDF.
  You can alter the PDF or export it.
- **open**
  the saved invoice
- **save**
- **find**
- **zoom**
- **zoom percentage**
- **zoom out**
- **full screen**
- **report outline**
- **thumbnails**
- **page settings**
- **edit page**
- **first page**
- **next page**
- **page number to go**
- **next page**
- **last**
- **close**

Now back to the details of figure 5, which is repeated here.

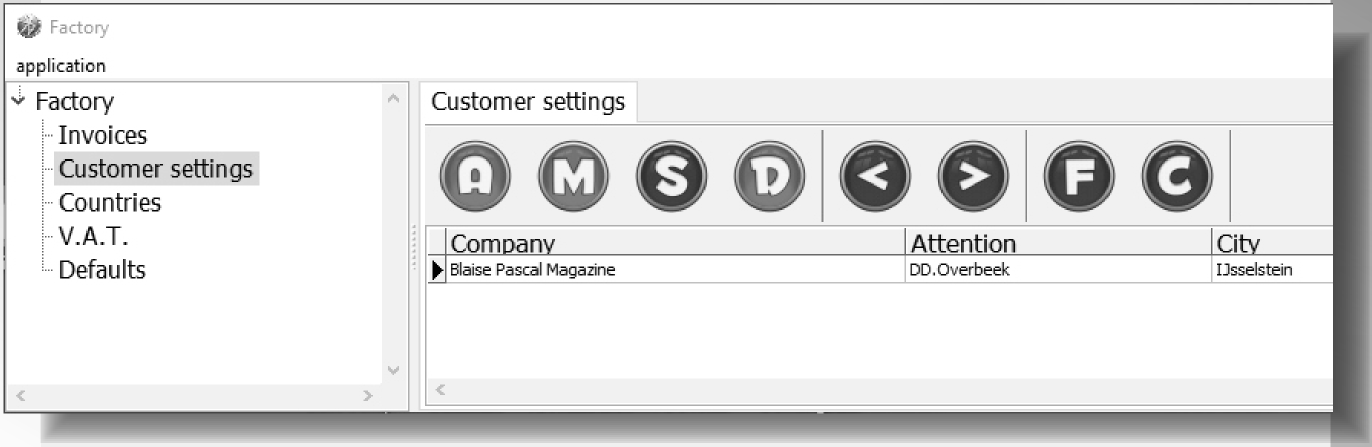


Invoicenumber	2
Description	test
Amount	100
VAT code	VAT 21
VAT amount	21
Total	121

Figure 6 Repeated.

As you can see in the title: here you can add save and delete new lines for your invoice.
Here comes an other element: Adding the Vat. (it’s a simple form which is explained in Figure 13 page 48.) It's self explanatory. The invoice number is handled in the Default options.

THE CUSTOMER SETTINGS

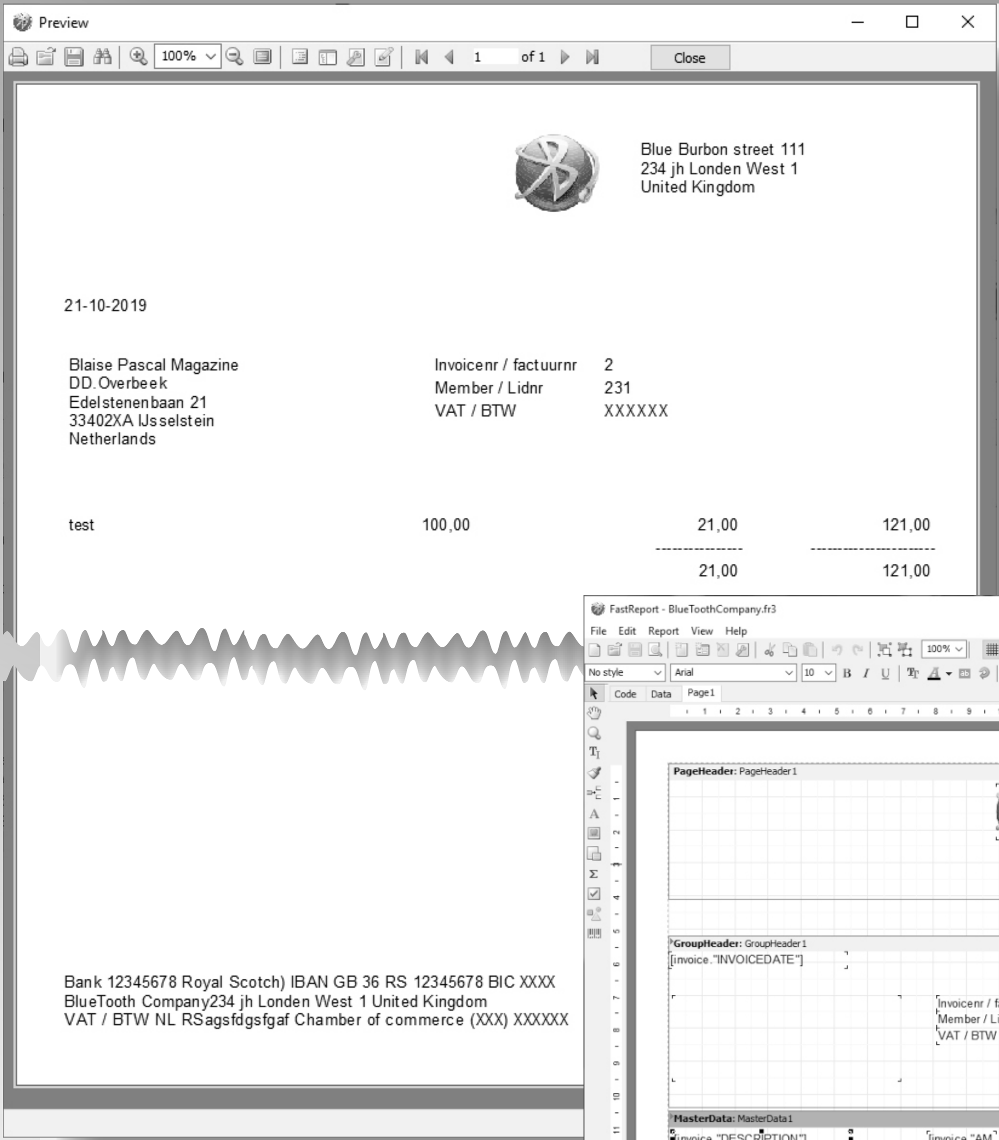


Company	Attention	City
Blaise Pascal Magazine	DD.Overbeek	IJsselstein

Figure 10: The Customer settings- not to be confused with the Company settings

In this form you actually add the details of the company or persons you want to send an invoice to. So you can have here a large overview of all the Customers.

Now when you double click on the first line you will be presented the detail form of the customers where you can add or alter customers.



Blue Burbon street 111
234 jh Londen West 1
United Kingdom

21-10-2019

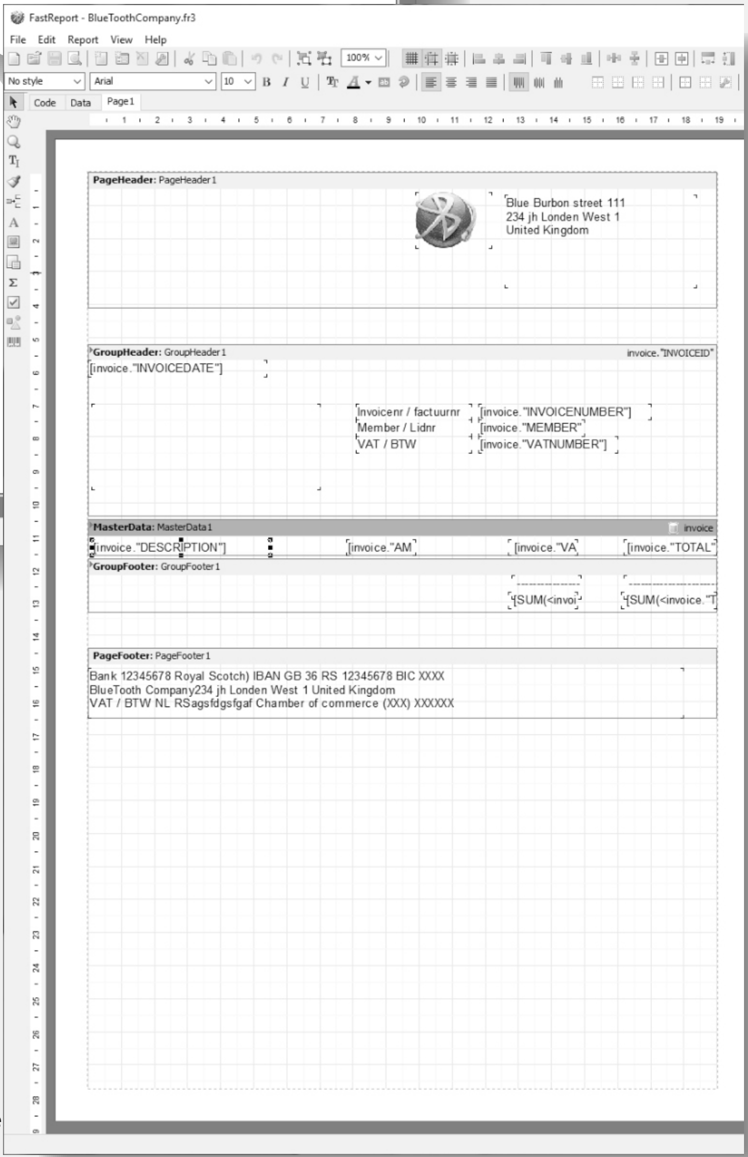
Blaise Pascal Magazine
DD.Overbeek
Edelstenenbaan 21
33402XA IJsselstein
Netherlands

Invoicenr / factuurnr	2
Member / Lidnr	231
VAT / BTW	XXXXXX

```
test                 100,00          21,00          121,00
                              ---------------  ---------------------
                                     21,00          121,00
```

Bank 12345678 Royal Scotch) IBAN GB 36 RS 12345678 BIC XXXX
BlueTooth Company234 jh Londen West 1 United Kingdom
VAT / BTW NL RSagsfdgsfgaf Chamber of commerce (XXX) XXXXXX

Figure 8: the result of opening a saved invoice: the format is PDF



PageHeader: PageHeader1

Blue Burbon street 111
234 jh Londen West 1
United Kingdom

GroupHeader: GroupHeader1 invoice."INVOICEID"

[invoice."INVOICEDATE"]

Invoicenr / factuurnr [invoice."INVOICENUMBER"]
Member / Lidnr [invoice."MEMBER"]
VAT / BTW [invoice."VATNUMBER"]

MasterData: MasterData1 invoice

[invoice."DESCRIPTION"] [invoice."AM [invoice."VA [invoice."TOTAL"

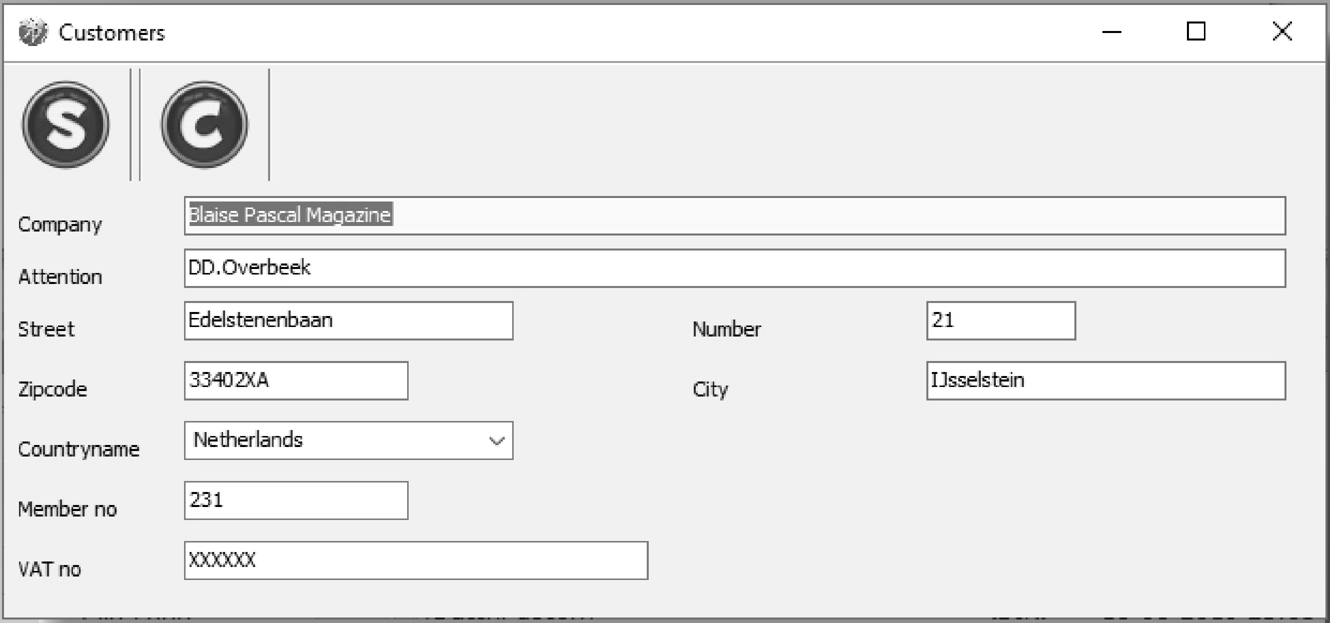
GroupFooter: GroupFooter1

[SUM(<invoi [SUM(<invoice."T

PageFooter: PageFooter1

Bank 12345678 Royal Scotch) IBAN GB 36 RS 12345678 BIC XXXX
BlueTooth Company234 jh Londen West 1 United Kingdom
VAT / BTW NL RSagsfdgsfgaf Chamber of commerce (XXX) XXXXXX

Figure 9: the form in design mode so you can change anything in the design. Don't forget to save it.

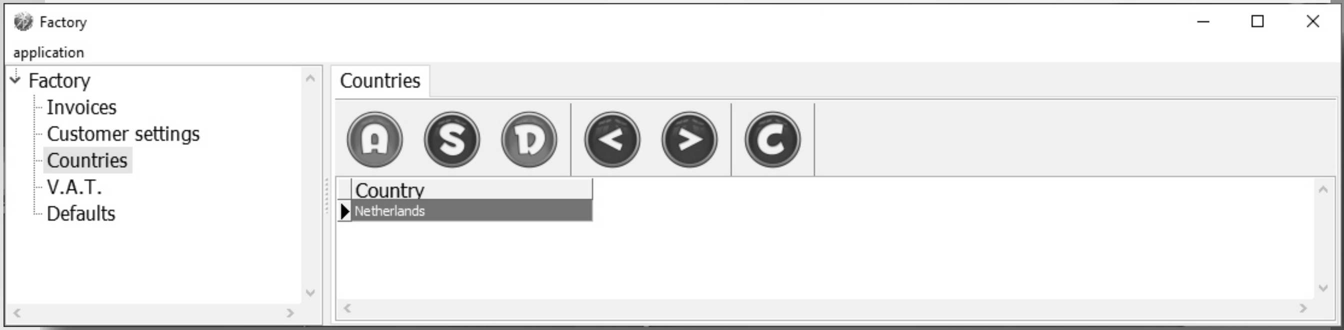


Company	Blaise Pascal Magazine
Attention	DD.Overbeek
Street	Edelstenenbaan
Number	21
Zipcode	33402XA
City	IJsselstein
Countryname	Netherlands
Member no	231
VAT no	XXXXXX

Figure 11: Details for the customer to be created

Here you can add the new customers address and choose the country. The country name is created in an other form: Countries.

THE FORM COUNTRIES - SETTINGS

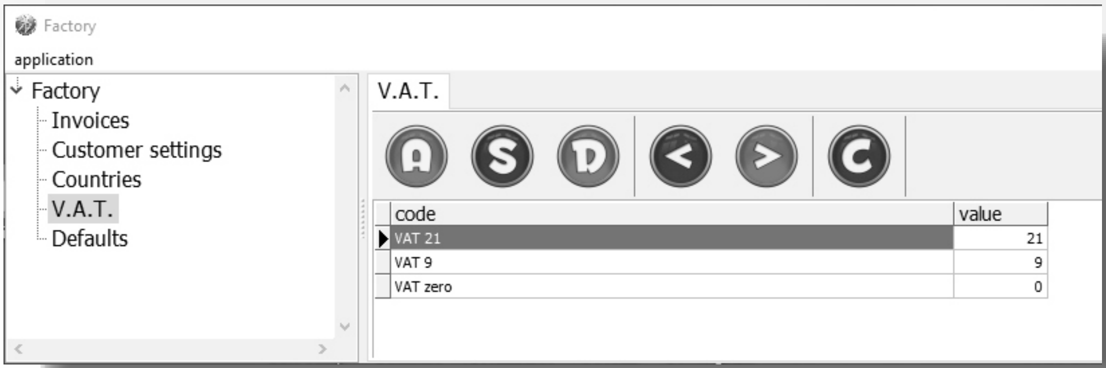


Country
Netherlands

Figure 12: The country settings

This form speaks for it self. Maybe you could add an import list of all countries.

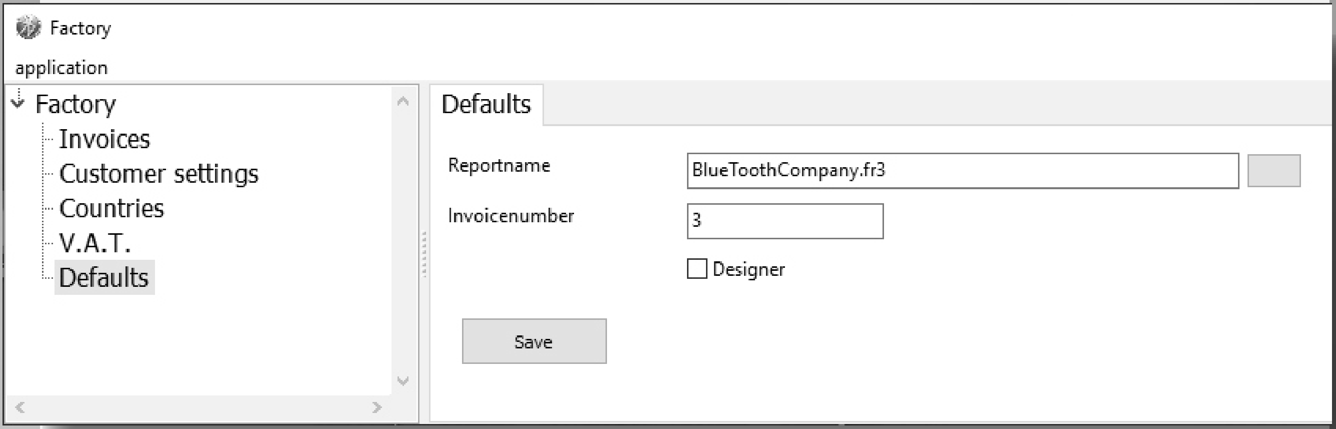
THE V.A.T. SETTINGS



code	value
VAT 21	21
VAT 9	9
VAT zero	0

Figure 13: The VAT codes (in percentage) can be added or altered here.

THE DEFAULT SETTINGS


Factory

application

- Factory
  - Invoices
  - Customer settings
  - Countries
  - V.A.T.
  - Defaults

Defaults

Reportname: BlueToothCompany.fr3

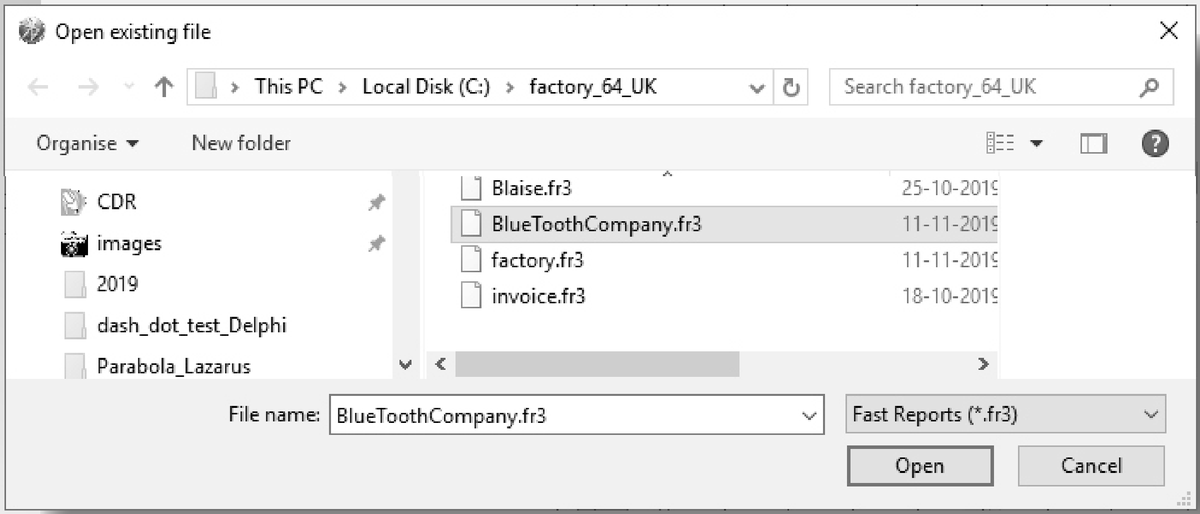
Invoicenumber: 3

☐ Designer

Save

Figure 14: This is a very important form with three main options:

1. Report name
the report name can be added here. You can of course choose between a number of reports or your design can be different for each of these Companies. Which means you can use this program for several companies (Those are not Customers: let’s say you help several foundations)


Open existing file

This PC › Local Disk (C:) › factory_64_UK

Blaise.fr3	25-10-2019
BlueToothCompany.fr3	11-11-2019
factory.fr3	11-11-2019
invoice.fr3	18-10-2019

File name: BlueToothCompany.fr3

Fast Reports (*.fr3)

Figure 15: Saving The report

2. Creating the Invoice number
You can add any kind of number, but not combined with text.

3. Designer Mode
This is a very nice and special mode: you can use the report designer of FastReport as a free tool. Which means that you can create your own report for the invoice and even change every report to your wishes, add images or pictures or logos.

There are two options: If you do not use the designer but simply want the invoice you still have the option to get into designer mode: click on the small icon with the pencil:

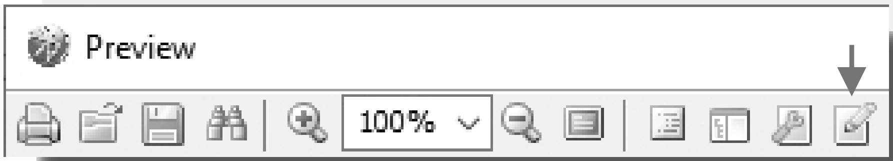
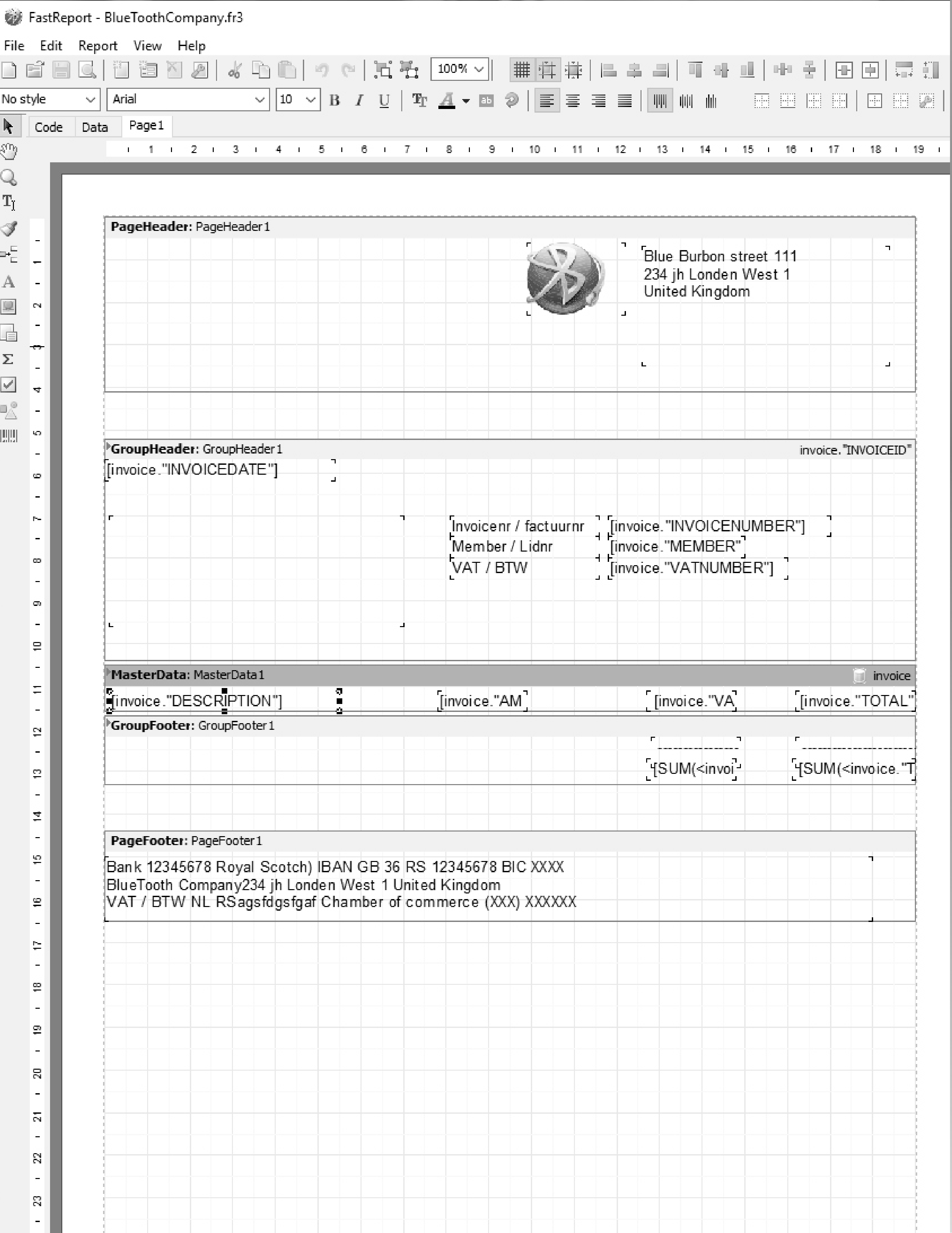


Figure 16: The pencil opens the designer mode.

So dependent on your choice: if you enable the designer mode you will always get the designer opened in front of you.



PageHeader: PageHeader1

Blue Burbon street 111
234 jh Londen West 1
United Kingdom

GroupHeader: GroupHeader1 invoice."INVOICEID"

[invoice."INVOICEDATE"]

Invoicenr / factuurnr [invoice."INVOICENUMBER"]
Member / Lidnr [invoice."MEMBER"]
VAT / BTW [invoice."VATNUMBER"]

MasterData: MasterData1 invoice

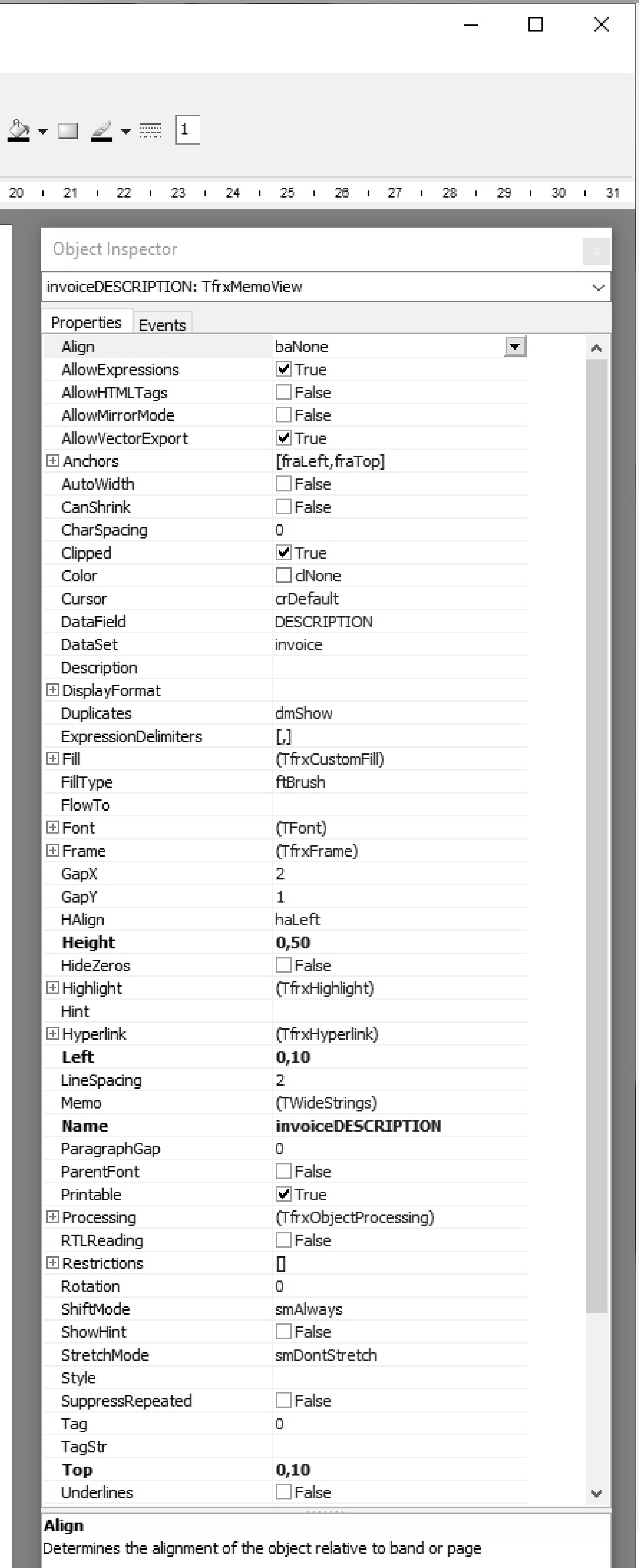
[invoice."DESCRIPTION"] [invoice."AM [invoice."VA [invoice."TOTAL"

GroupFooter: GroupFooter1

--------------- ---------------------
[SUM(<invoi [SUM(<invoice."T

PageFooter: PageFooter1

Bank 12345678 Royal Scotch) IBAN GB 36 RS 12345678 BIC XXXX
BlueTooth Company234 jh Londen West 1 United Kingdom
VAT / BTW NL RSagsfdgsfgaf Chamber of commerce (XXX) XXXXXX



Object Inspector

invoiceDESCRIPTION: TfrxMemoView

Properties | Events

Align	baNone
AllowExpressions	✓ True
AllowHTMLTags	False
AllowMirrorMode	False
AllowVectorExport	✓ True
⊞ Anchors	[fraLeft,fraTop]
AutoWidth	False
CanShrink	False
CharSpacing	0
Clipped	✓ True
Color	clNone
Cursor	crDefault
DataField	DESCRIPTION
DataSet	invoice
Description	
⊞ DisplayFormat	
Duplicates	dmShow
ExpressionDelimiters	[,]
⊞ Fill	(TfrxCustomFill)
FillType	ftBrush
FlowTo	
⊞ Font	(TFont)
⊞ Frame	(TfrxFrame)
GapX	2
GapY	1
HAlign	haLeft
Height	**0,50**
HideZeros	False
⊞ Highlight	(TfrxHighlight)
Hint	
⊞ Hyperlink	(TfrxHyperlink)
Left	**0,10**
LineSpacing	2
Memo	(TWideStrings)
Name	**invoiceDESCRIPTION**
ParagraphGap	0
ParentFont	False
Printable	✓ True
⊞ Processing	(TfrxObjectProcessing)
RTLReading	False
⊞ Restrictions	[]
Rotation	0
ShiftMode	smAlways
ShowHint	False
StretchMode	smDontStretch
Style	
SuppressRepeated	False
Tag	0
TagStr	
Top	**0,10**
Underlines	False

Align
Determines the alignment of the object relative to band or page

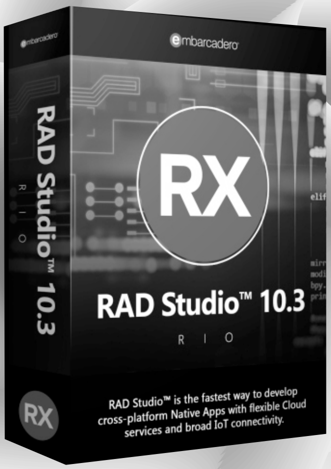
barnsten

• development tools • training
• consultancy • hands-on workshops
• components • support

YEAR END DEALS AT BARNSTEN

UNTIL DECEMBER 15, 2019

VALID FOR DELPHI C++BUILDER AND RAD STUDIO LICENSES


embarcadero
RX
RAD Studio™ 10.3
RIO
RAD Studio™ is the fastest way to develop cross-platform Native Apps with flexible Cloud services and broad IoT connectivity.

15% discount on all Professional versions
25% discount on all Enterprise versions
30% discount on all Architect versions

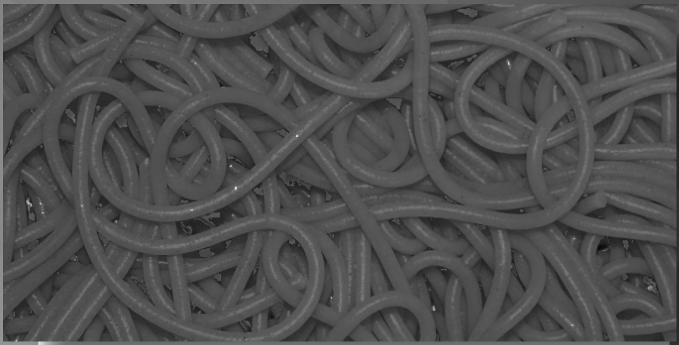
RAD Studio™ is the fastest way to design and develop modern, cross-platform native apps and services.

It's easier than ever to create stunning, high performing apps for Windows, macOS, iOS, Android and Linux Server (Linux Server is supported in Delphi Enterprise or higher), using the same native code base.

Share visually designed UIs across multiple platforms that make use of native controls and platform behaviors, and leverage powerful and modern languages with enhancements that help you code faster.

Developers pick RAD Studio™ because it delivers 5x the speed for development and deployment across multiple desktop and mobile platforms.

https://www.barnsten.com/default/promotions



PREFACE

I have presented **SmartBinding** in a couple of articles. The purpose of **SmartBinding** was to bind data containers together.
Often one container would be something that could display and / or interact with its data, while the other could be a container holding data elsewhere, like the result from a query, or a list of objects or a record or something else.
SmartBinding also introduced data proxies, which made it very easy to separate visual design and functionality from hardcore data manipulation/retrieval and storage.

In new release of **`kbmMW`**, **SmartEvent** has been added. The purpose of **SmartEvent** is primarily to separate user interaction from business logic.
In other words a super version of the old hardwired event handling.

Someone might say… action lists… they are the answer. Yes, but no.
`TActionList` is still a hardwired fairly simple way to operate events. **SmartEvent** is much more generic than that.

STARTING TO DIG INTO IT

What makes SmartEvent tick?
Well.. it is a publish / subscribe based notification framework.
I was not as such aware about it, but **Jim McKeeth** recently told me that I had essentially **"Delphinized"** the way Apples ObjectiveC operates.
I don't know if that is to be understood as a good or a bad thing ☺.
ObjectiveC has an absolutely horrible syntax IMO.

However… I'm sure the designers have had the same objective as I… finding ways to separate view from control and model.

So what means **publish / subscribe based notification**? It means that you define which parts of your code have interest in something and makes them subscribe for a "happening"… an event notification. In good old days (*BSE – Before introduction of SmartEvent's*) that would be your …OnSomething eventhandler.

In modern days (*ASE – After introduction of Smart Events*) it would basically be any method you have in your code.

So you could have a method like this:

```
procedure TForm1.ShowSomeMessage
(const AMessage:string);
```

which could subscribe for every intent to show messages. I'll show in a moment how.

But there is no fun in subscribing for anything if nothing or nobody is offering it… *(I know all about it… having subscribed for fun, long-drinks with umbrellas on the beach of my own tropical island, and an endless amount of money…. and still not found the sponsors for it*).

So we need to be able to offer information to the subscribers, and we need a mechanism that makes it possible to indicate what type of offering one is making, and thus only subscribers that have an interest in that particular offering, receives it.

The key thing here is the term **"subject".** A subject is a sentence of . (dot) separated words.

For example one could make a subject called:
`SHOW.ERROR or SHOW.INFO or`
`SHOW.ICECREAM or SHOW.FOOD.BANANA`
etc.

Basically you define what the subject should be.
It doesn't even have to have dots in it.
But usually its a good idea to make it hierarchical, in the sense that you have starting with logical generic context words, and ending with more specialised words that match the topic at hand.

When you then have a situation where you want to notify potential subscribers about that you have an error to show, you simply send a notification with the subject **"SHOW.ERROR"**.

So now we know what a subject is, we need to learn how we subscribe to that subject.

In the former case, we want the method **ShowSomeMessage** to be called everytime someone publishes a **SHOW.ERROR** notification

That is very easily defined:

```
TForm1 = class...
...
[kbmMW_Event('SHOW.ERROR')]
procedure TForm1.ShowSomeMessage(const
AMessage:string);
...
```

And it is as easy to send the notification:

```
...
  Event.Notify('SHOW.ERROR','Some error happened');
...
```

We just need to do one thing to make it all work… tell the SmartEvent framework about your form instance is interested in subscribing for stuff.

So somewhere (perhaps in your TForm1 constructor), you will have this statement:

```
...
  Event.Subscribe(self);
...
```

where self is the form instance itself.
You can also reference another instance of a class on which methods has been tagged with the **kbmMW_Event** attribute.

The moment the **Notify** command is sent, the method **ShowSomeMessage** is called.

A use case

There are many scenarios, in which **SmartEvent** will make live easier for you. One of them is if you have an application with a couple of main form / datamodules, and a host of frames or other forms, you want to display and remove during the use of the application.

Previously you would either have to make sure each of the forms directly referenced the main form or data module units in which you have your business code and / or data available, or you would have to write plenty of event handlers to make it possible to do IOC… basically to hook up external code in your main datamodule / form to your event handlers.

In both cases you would have to write alot of code, and for each small extra feature that would need interaction from other parts of your application, you would manually have to link the stuff together somehow. You would typically start to limit in which units you place your code, to minimize the complexity and the number of unit uses clauses ending up in a royal spaghetti mess (*with meatballs*).

SmartEvent makes this extremely easy for you, since any part of your code, can choose to subscribe for happenings, and any other parts of your code can choose to publish notifications where needed.

Since you simply write **Event.Subscribe(yourform/frame/datamodule/otherclassinstance)** when you need that instance to participate in handling notifications, it is very easy to plug in new functionality without the rest of your code needing to know about its implementation.

Obviously, you might want to close down a form / frame etc. when you no longer want to use it…
It could for example be a form displaying a list of grocery items that the user can select between. The form with the list will then publish a notification with a wish for getting a relevant list of grocery items to display, and when the user selects an item, publish a notification about which item has been selected, and then close down the form.

Before closing down the form, simply write:

```
...
Event.Unsubscribe(yourform/frame/datamodule/oth
erclassinstance);
...
```

Then the form will no longer be notified, which makes sense because you are about to destroy it.

But what about....
Showing other messages?

Remember the procedure was named **ShowSomeMessage,** so it is intended not only to show error messages… but presumably any messages.
But since it only subscribes for **SHOW.ERROR,** it will not react on for example **SHOW.INFO.**
This is easy to remedy:

```
TForm1 = class...
...
[kbmMW_Event('SHOW.>')]
procedure TForm1.ShowSomeMessage(const
AMessage:string);
...
```

Now we are using a wildcard. We have told the framework, that ShowSomeMessage should be called everytime anybody publishes a notification starting with **SHOW.**
The **>** indicates that all parts (*each word between dots are called parts*) starting at the place of the > are to be considered a match.

One can also use ***** as a wildcard for a specific part. In our **SHOW.ERROR** vs **SHOW.INFO** sample, a subscription for **SHOW.*** and **SHOW.>** would essentially result in the same. But **SHOW.>** would also match **SHOW.FOOD.BANANA,** which **SHOW.*** would not match with since there are extra parts.

More arguments?
Simply add the additional arguments to the **Notify** call. Methods subscribing will receive as many arguments as possible that match. If there is a mismatch between types of data, an exception will be raised. Like notifying a method that only accepts an integer, with a string.
Notify directly accepts up to 9 arguments.
If you have more you can use the version of **Notify** that takes an array **TValue.**

For example

```
Event.Notify('SOME.SUBJECT',10,20,30,'ABC');
Event.Notify('SOME.SUBJECT',[10,20,30,'ABC']);
```

If there are no arguments to provide, you can just publish without an argument:

```
Event.Notify('SOME.OTHER.SUBJECT');
```

ANONYMOUS FUNCTIONS?

You can directly let specific parts of your code subscribe like this:

```
Event.Subscribe('SOMETHING.>',
  procedure(const AContext:IkbmMWEventContext)
  begin
     Log.Debug('Global SOMETHING.> subject:'+AContext.Notification.Subject);
  end).NamedAs('Sub1').Activate;

Event.Subscribe('SOMETHING.MORE.>',
  procedure(const AContext:IkbmMWEventContext)
  begin
     Log.Debug('Global SOMETHING.MORE> subject:'+AContext.Notification.Subject);
  end).NamedAs('Sub2').Synchronized.Activate;
```

In the above code, we have even named each of the subscriptions (optional). You can use the name to unsubscribe later on if you need to, alternatively use the IkbmMWEventSubscriber instance returned from the Event.Subscribe method for later unsubscription.

Further we have in the subscription for SOMETHING.MORE.> requested the subscription to be executed synchronously, which means that it will be running in the main VCL/Firemonkey GUI thread. This is important if the code needs to update the GUI somehow, and since you, as the developer, knows exactly what your event handling code is doing, you can easily specify for the framework, that its GUI sensitive.

Then the framework will automatically make sure it is executed accordingly.

RETURNING DATA?

Sometimes (often) you will want to publish a notification with the expectancy of getting some data back. Like in our grocery list example further up. The frame/form publishes a notification about the with to receive a list of groceries.

You would make a method available somewhere in your application, that can produce a grocery list. In this example by returning a TkbmMemTable instance with the items:

```
...
  [kbmMW_Event('REQUEST.GROCERYLIST')]
  function RequestGroceryList:TkbmMemTable;
...
```

The frame/form containing the list would publish the wish for the grocery list like this:

```
Event.Notification('REQUEST.GROCERYLIST')
   .WantSubscribers
   .WantResults
   .WhenNotified(
    procedure(const AContext:IkbmMWEventContext)
    var
      mt:TkbmMemTable;
    begin
       mt:=Use.AsMyObject<TkbmMemTable>(AContext.Result);
       if mt<>nil then
         ShowList(mt);
    end)
   .Send;
```

We are now using a slightly different variant of the Notify method. One that allows chaining different bits and pieces together to augment the published notification with additional information.
In this situation, we ask the framework to ensure that we get informed about all the subscribers reacting to our notification, and also that we want the results from each of them.
The code in WhenNotified is run each time a subscriber event is done running, which then makes the memory table instance available for the list. The ShowList method (or the frame/form) is now the owner of the memory table instance (because we used Use.AsMyObject to get the instance from Result). If we used Use.AsObject instead, then the framework would automatically clean up the memory the moment Result goes out of scope.

But multiple subscribers (*or none)* could have heard our plea for a grocery list? Yes. that is very much possible. In the above case, the list will be shown for each of the subscribers that returned data to us, which may not be the result we wanted.

We probably only wanted the first if multiple subscribers responded.

In such case we use WhenDone instead of WhenNotified:

```
Event.Notification('REQUEST.GROCERYLIST')
   .WantSubscribers
   .WantResults
   .WhenDone(
    procedure(const AContext:IkbmMWEventContext)
    var
      mt:TkbmMemTable;
    begin
       mt:=Use.AsMyObject<TkbmMemTable>(AContext.Result);
       if mt<>nil then
         ShowList(mt);
    end)
   .Send;
```

Then it will only be called once… and only the first result being provided will be used.

However if you would like to see all the results, then do like this: (*see code below*)

```
Event.Notification('REQUEST.GROCERYLIST')
   .WantSubscribers
   .WantResults
   .WhenDone(
    procedure (const AContext:IkbmMWEventContext)
    var
      lst:TList<variant>;
      i:integer;
      mt:TkbmMemTable;
    begin
       Log.Debug('Done notifying '+inttostr(AContext.Notification.NotifiedCount)
                                   +' about '+AContext.Notification.Subject);
       Log.Debug('  Number of results: '+inttostr(AContext.Notification.ResultCount));
       lst:=AContext.Notification.Results.BeginRead;
       try
         for i:=0 to lst.Count-1 do
         begin
            mt:=Use.AsObject<TkbmMemTable>(lst.Items[i]);
            if mt<>nil then
              Log.Debug('  Result '+inttostr(i)+': Number of records: '+inttostr(mt.RecordCount));
         end;
       finally
         AContext.Notification.Results.EndRead;
       end;
    end)
   .Send;
```

MAIN VCL/FIREMONKEY GUI THREAD?
I touched it a bit before where one of the code based subscription's used the Synchronized method to indicate that the event handler should be run within the main GUI thread, probably because it needs to update the GUI.

The **kbmMW_Event** attribute can also state that requirement:

```
 TForm1 = class...
...
[kbmMW_Event('SHOW.ERROR',[mweoSync])]
procedure TForm1.ShowSomeMessage(const AMessage:string);
...
```

Then ShowSomeMessage will always be run in the scope of the main GUI thread which makes it safe to update the GUI.

What about the **WhenDone** or **WhenNotified** notification augmentations? What if the code in them should be run **synchronized?**
Simple… just add the **.Synchronized** augmentation to the notification:

```
Event.Notification('REQUEST.GROCERYLIST')
        .Synchronized
        .WantSubscribers
        .WantResults
        .WhenDone(...)
        .Send;
```

FINAL THOUGHTS
At first this may seem more difficult than just writing an eventhandler.
And for simple applications you may very well be fine without **kbmMW SmartEvent.**
But the more complex application is getting, the more cumbersome it gets to manage all the different dependencies, and interactions between forms, data and code.

By combining **kbmMW SmartBind** and **kbmMW SmartEvent** you have the complete solution to split your application into a number of selfcontained bits and pieces, who only needs to know about a list of subjects they can use, and perhaps some names of some data they can bind to.

Plugging the form / frame in and out will then be extremely simple.

COMPONENTS DEVELOPERS 4

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.10.00 RELEASED!

We are happy to announce v5.10.00 of the most complete development add on for Delphi and C++Builder for building native modular and fast multi-tier solutions.
We strive hard to ensure kbmMW continues to set the bar for what an n-tier product must be capable of in the real world!
Notice that kbmMemTable v. 7.83.00 or newer is a prerequisite to this update.
This is a major release containing major new features, updates to existing features and bugfixes.

The release includes:

- NEW! SmartBind now fully supports VCL, FMX, including image/graphics and TListView
- NEW! SmartBind data generators and data proxies for easy separation of data sharing concerns in modular applications
- NEW! SmartEvent for easy separation of event and execution workflow based concerns for the ultimate in modular application design
- NEW! Native highly scalable TCP server transport now also supports REST
- Significant improvements and fixes in many areas including
  - RTTI
  - Scheduler
  - LINQ
  - Object Notation
  - ORM

Professional and Enterprise Edition is available for all with a current active SAU. If your SAU has run out, please visit our shop to extend it with another 12 months.
CodeGear Edition may be available for free, but only supports a specific Delphi/Win32 SKU, contains a limited feature set and do not include source.
`https://portal.components4developers.com` to download

kbmMW is the premiere n-tier product for Delphi, C++Builder and FPC on Win32, Win64, Linux, Java, PHP, Android, IOS, .Net, embedded devices, websites, mainframes and more.
`http://www.components4developers.com` for more information about kbmMW.

Components4Developers is a company **established in 1999** with the purpose of providing high quality development tools for developers and enterprises. The primary focus is on SOA, EAI and systems integration via our flagship product kbmMW. kbmMW is currently used as the backbone in hundreds of central systems, in hospitals, courts, private, industries, offshore industry, finance, telecom, governments, schools, laboratories, rentals, culture institutions, FDA approved medical devices, military and more.

5.10.00 Nov 10 2019:

NEW ITEMS

- Huge amount of improvements to kbmMW SmartBind including
  See features explained in blog posts at components4developers.blog
  Data generators
  Listview VCL and FMX support
  Binding used string based settings.
  AutoBind which locates properties or methods which has the attribute kbmMW_AutoBind.
  The attribute can be either a string based binding or refer to a method which will be called during autobinding to setup all the required bindings for that particular instance.
  Binding to data proxies for clear model/view separation.
  Binding images/graphics
  Break values (grouping header) support (currently ListView)
  Binding kbmMWDateTime and kbmMWNullable
  Saving and loading binding definitions to/from object notation formats like XML, YAML, JSON etc.
  Horizontalscroll, verticalscroll and cell based grid binding.
- Added kbmMWSmartEvent.pas. kbmMW SmartEvent is an advanced pub/sub based event mechanism to replace regular Onxxxx event handlers, making it much easier to separate concerns in an application.
  See videos and blogposts.

NEW ITEMS (CONTINUATION)

- Added new TkbmMWConcatStream in kbmMWGlobal.pas.
  It allows multiple streams to seem as one when reading, without reallocating memory.
- Added TkbmMWBoyerMoore class, containing multiple ways to search for sub data in data very quickly to kbmMWGlobal.
- Added kbmMWSockClient.pas for a skeleton for an easy to use async client request/response setup. Currently in experimental phase.
- Added support for REST to native TCP Transport.

CHANGES/MINOR ADDITIONS

- - Updated look of wizards.
- - Added support in kbmMWScheduler for scheduled events indicating ownership to other scheduled events. Purpose is establishing termination ownership and more in nested scheduled events.
- - Improved TkbmMWScheduledEvent.WaitRuns to allow for external anonymous function to determine is wait is to be extended.
- - Updated kbmMW LINQ to support using TStrings instances for input and output with optional autoconversion of numeric data.
- - Updated streamformat controllers with new GetTotalSize class function.
- - Updated TkbmMWSortedList to support Unique constraint.
- - Updated TkbmMWThreadList, TkbmMWThreadDictionary<TKey,TValue> and TkbmMWThreadObjectDictionary<TKey,TValue> to support providing a comparer.
- - Added IkbmMWThreadAutoValue and TkbmMWThreadAutoValue for scope based reference oounted thread safe value variables.
- - Added CardinalToText, CardinalToHexText, HexTextToCardinal, AddChars to TkbmMWTextPChar.
- - Added CardinalToText, CardinalToHexText, HexTextToCardinal, AddBytes, ExpectOneOfBytes to TkbmMWTextPByte.
- - Improved TkbmMWPrettyBinary to include pretty versions of relevant 8 bit characters.
- - Moved TkbmMWSubscriptionHash to kbmMWGlobal.pas
- - Added global functions kbmMWVariantToByteArray and kbmMWByteArrayToVariant to kbmMWGlobal.pas.
- - Added PrettyOptions:TkbmMWPrettyHexOptions and UseStackMap:boolean to IkbmMWLogFormatter in kbmMWLog.pas. Controls how pretty data and stackdumps are produced when logging.
- - Made minor improvements to kbmMWNullable
- - Updated kbmMWWinsock.pas
- - Added ASkipCache:boolean option to TkbmMWCustomPooledDataSet.PopulateData
- - Added InheritsFromTControl, InheritsFromTComponent, GetInstanceUnit, GetMethods to TkbmMWRTTI
- - Added String2DateTimeStyle and DateTimeStyle2String to TkbmMWDateTime.
- - Updated kbmMWObjectNotation with various minor features.

FIXES

- - Fixes and minor improvements to kbmMW ORM
- - Fixed minor issues in kbmMWDebugStackTrace
- - Fixed format issues with TkbmMWScheduledEvent.Cron
- - Fixed bugs in TkbmMWSceduler
- - Fixed various bugs in kbmMWGlobal.pas
- - Fixed kbmMWFindReplace in kbmMWHTTLUtils.pas
- - Fixed bugs in TkbmMWPooledDatasetRefreshSchedulerThread.RefreshDataset
- - Fixed TkbmMWCustomPooledDataSet.MasterChanged issues.
- - Fixed TkbmMWRTTI.CompareValue
- - Minor fixes and improvements in kbmMWYAML, kbmMWSQLite, kbmMWBinaryParser, kbmMWSecurity
- - Fixed timezone parsing in TkbmMWDateTime.ParseRFC1123DateTime
- - Fixed bug in TkbmMWInterbaseSQLRewriter.RewriteDescribeTable
- - Fixed procedure TkbmMWLockFreeHashArray.GC

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.10.00 RELEASED!

- RAD Studio XE2 to 10.3 Rio supported
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support
- Native high performance 100% developer defined application server
- Full support for centralized and distributed load balancing and failover
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multithread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronouncable password generators.
- High performance LZ4 and Jpeg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, JSON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

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

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

♦ **NEW! SmartBind now fully supports VCL, FMX, including image/graphics and TListView**

♦ **NEW! SmartBind data generators and data proxies for easy separation of data sharing concerns in modular applications**

♦ **NEW! SmartEvent for easy separation of event and execution workflow based concerns for the ultimate in modular application design**

♦ **NEW! Native highly scalable TCP server transport now also supports REST**

♦ **Significant improvements and fixes in many areas including**
- ◊ **RTTI**
- ◊ **Scheduler**
- ◊ **LINQ**
- ◊ **Object Notation**
- ◊ **ORM**

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

COMPONENTS 4 DEVELOPERS

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