BLAISE PASCAL MAGAZINE 117

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

IT FINALLY HAPPENS:
WITH FRESNEL YOU CAN CREATE
A UNIVERSAL GRAPHICAL APPLICATION
RUNNING ON ALL NATIVE PLATFORMS AND IN THE BROWSER

Fresnel: the new alternative lcl for lazarus

Coming Technology: **Glass Cores** (CPU) from Intel

Controlling the browser using webassembly
Accessing the Browser APIs from Webassembly.

Database Workbench 6.5 added **support for SQL LITE**
The Swiss army knife for database development

The Lazarus Debugger – An introduction and tutorial
Part 7: Change happens – Waiting for it

PUTS: Pascal User Tips & Solutions
Delphi: create your own Components
A delay component and a Time lapse component

PUTS: Pascal User Tips & Solution
The component Treeview

BLAISE PASCAL MAGAZINE 117
Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js /
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

CONTENT

ARTICLES

From your Editor Page 4
From our Technical Advisor, Jerry King Page 5

PUTS: Pascal User Tips & Solutions Page 7 / *By Detlef Overbeek*
Delphi: create your own Components
A delay component and a Time lapse component

PUTS: Pascal User Tips & Solutions Page 17 / *By Detlef Overbeek*
The component Treeview

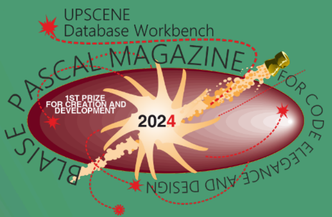
Database Workbench 6.5 added **support for SQL LITE** Page 19
The Swiss army knife for database development / By Detlef Overbeek

Coming Technology: **Glass Cores** (CPU) from Intel Page 53
By Detlef Overbeek

The Lazarus Debugger – An introduction and tutorial Page 71
Part 7: Change happens – Waiting for it
By Martin Friebe

Controlling the browser using webassembly Page 76
Accessing the Browser APIs from Webassembly.
By Michael van Canneyt

PROJECT FRESNEL: THE TIME IS NOW - AN UPDATE
IT FINALLY HAPPENS: Page 92
WITH FRESNEL YOU CAN CREATE
A UNIVERSAL GRAPHICAL APPLICATION
RUNNING ON ALL NATIVE PLATFORMS
AND IN THE BROWSER *By Michael van Canneyt*
Fresnel: the new alternative LCL for lazarus


UPSCENE
Database Workbench
BLAISE PASCAL MAGAZINE
FOR CODE ELEGANCE AND DESIGN
1ST PRIZE FOR CREATION AND DEVELOPMENT
2024

ADVERTISING

ARTICLES

LAZARUS HANDBOOK Page 6
David Dirkse Book Computer /Graphs / Games & Math Page 16
SUPERPACK Page 52/54
LIBRARY Stick including USB Card Page 58LAZARUS HANDBOOK + SUBSCRIPTION Page 29
DELPHI SUMMIT Announcement Page 62
Blaise Pascal Magazine We are at the summit Page 63
DELPHI SUMMIT Barnsten Page 64
DELPHI SUMMIT Sale Page 65/66
DELPHI SUMMIT Agenda Page 67/68/69
Database Workbench / Upscene Page 75
FASTREPORT Page 90
FRESNEL Page 91
LAZARUS HANDBOOK + SUBSCRIPTION Page 60/61/110
Ukraine Special Offer Page 111
Components4Developers Page 112

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

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

CONTRIBUTORS

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

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

Subscriptions can be taken out online at **www.blaisepascalmagazine.eu** or by written order, or by sending an email to **office @ blaisepascal.eu.** Subscriptions can start at any date. All issues published in the calendar year of the subscription will be sent as well. Subscriptions run 365 days. Subscriptions will not be prolonged without notice. Receipt of payment will be sent by email. Subscriptions can be paid by sending the payment to: ABN AMRO Bank Account no. 44 19 60 863 or by credit card or PayPal
Name: Pro Pascal Foundation (Stichting Ondersteuning Programeertaal Pascal) IBAN: NL82 ABNA 0441960863 BICABNANL2A
VAT/NL814254147B01
Subscription department Edelstenenbaan 21 / 3402 XA IJsselstein, Netherlands + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

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

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

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

From your editor

Hello dear readers,
I have a very important announcement to make:
FRESNEL is here.
Ill try to explain in short what it means.
Fresnel is the new LCL.
We have talked about that before and at the summit in Amsterdam we will show some results of it, and in October at the University of Cologne (Köln) Germany we will show the achievements we made.
Mattias Gärtner and Michael van Canneyt have been partners in crime.
They finally did it:
drag Lazarus into the Future...
Since ever we had discussions about the way Lazarus looked and handled the graphical environment
for all of the Os's. Far to difficult etc. I asked Michael and Mattias and also Martin what could be done about that and they came up with the new Framework FRESNEL.

So they started working on it and here is final proof. It works.
At page 92 you'll find an explanation of the many details there are.
The achievement is enormous:
We can now work with totally new designed components on the basis of the complete component set that Lazarus and FPC has. These components are custom-drawn components that we use for the LCL and now for FRESNEL.
What makes it very special is that we added CSS Style to these components and we are very busy creating the new components and we can use help if you dare to.
So for now we have a platform running on all native platforms and in the browser,
wit the help of WebAssembly we can create apps that can run on the desktop and the web.
Michael created even a filesystem for this.
For the future we will implement Android as well.
All this using a single codebase,
and running at native speed.
And obviously,
all this using your favourite Programming language: Object Pascal.

Have lots of fun reading the articles...

From our technical Advisor, Jerry King

"I don't need to know your age. I saw you using a flip phone, so I'll just assume you're old."

CREATING TWO SIMPLE COMPONENTS FOR DELPHI

Rewritten and converted by Detlef Overbeek, inspired by an old Dutch article of Paul J Gellings Blaise 34 page 169.

This article covers the creation of two very simple **Delphi** components.

It has **two purposes:**
Show how to **create a component from scratch** and create something useful. **For those for whom this is entirely new, the entire procedure is followed step by step.**

The **first component** makes it possible **to set a wait time** even in **Delphi** (`delay`) in **Delphi**, for example to be able to view screen output or intermediate results or something similar and is especially useful during the development of a program.
The ancient Borland Pascal possessed the **built-in Delay** procedure, but **Delphi** doesn't have it.

The **second component** is intended to **measure times**, for example the time needed for a certain calculation or for some other task performed by Delphi and with an accuracy of milliseconds. The components can then be placed directly from the component palette on a Form and thus included in an application and in doing so, the unit name associated with the component is automatically included in the uses clause. Much more could be said about components, of course but for the purposes of this article we must leave it at that.

The source code of the two components discussed is contained in the units **`UDelay.pas`** and **`UClock.pas`**.
For the icons to be placed on the component palette, we will handle that in a separate article.

Since Delphi constantly changes handling of menus, the user interface etc we simply have to repeat certain things since they are not always very logical ordered
A simple test program for these two components is the **`TestTime.dpr`** with the corresponding unit **`UTestTime.pas`**. The code of these files will be made available for our readers at **`https://www.blaisepascalmagazine.eu/en/your-downloads/`** *see image at the bottom.*

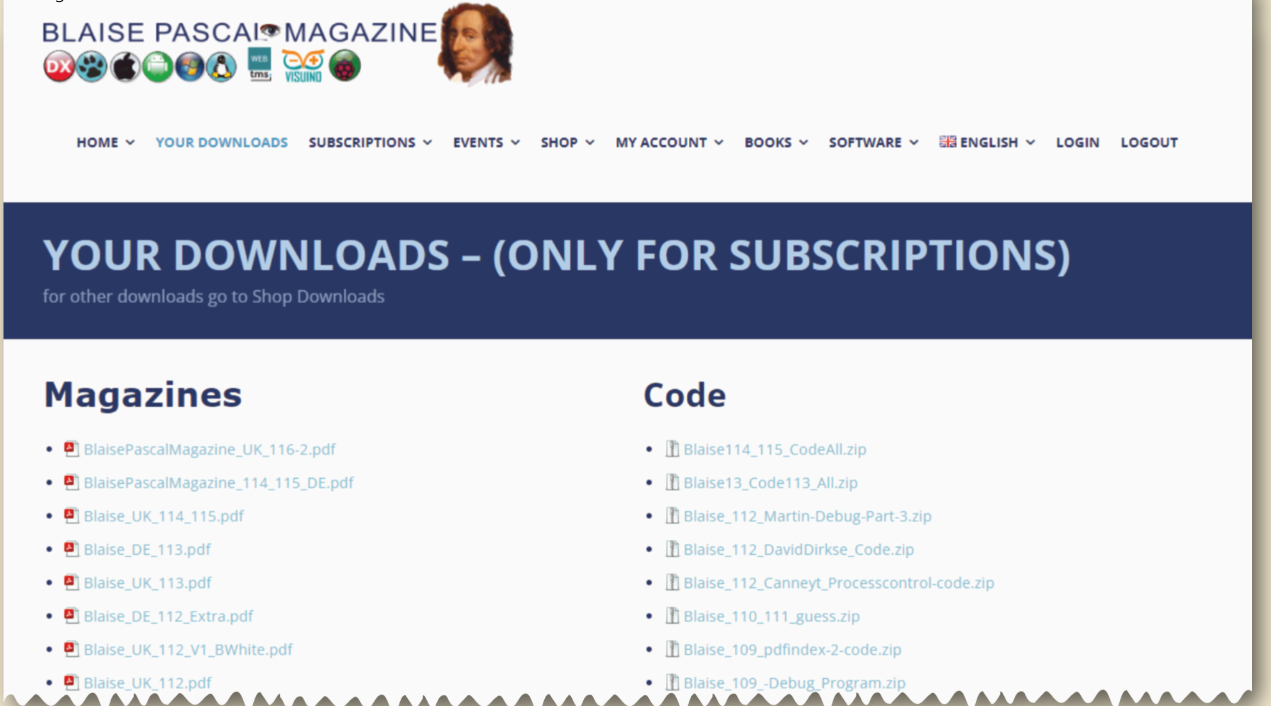
CREATING A NEW COMPONENT

We start with the **step-by-step description** of the **component** which we call **`TDelayer`** and which can be used to set a delay set. This component is itself derived from **`TComponent` the highest class** on the tree - the most general component, with which **`TDelayer`** also automatically inherits the methods associated with **`TComponent`**.

To create a new component, (*this action is to show how component code looks like*) we first choose from the top listing: (*See page 3 of this article*) → **Component → New Component → VCL for Delphi Win32**.

A new widow appears: **Ancestor Component**.
Since we have only the highest class to use: (*there is nothing in between*) we select **`TComponent`**. If you choose in the next window `TComponent` it will become the name of the unit: `Component1.` The number is for auto increment, if you want to do more. At this point again a wizard pops up that will create the component text. **Click finish**.
(*See page 4 of article for the code listing*).

Figure 1 Downloads



Welcome Detlef Overbeek

BLAISE PASCAL MAGAZINE

YOUR DOWNLOADS – (ONLY FOR SUBSCRIPTIONS)

for other downloads go to Shop Downloads

Magazines

- BlaisePascalMagazine_UK_116-2.pdf
- BlaisePascalMagazine_114_115_DE.pdf
- Blaise_UK_114_115.pdf
- Blaise_DE_113.pdf
- Blaise_UK_113.pdf
- Blaise_DE_112_Extra.pdf
- Blaise_UK_112_V1_BWhite.pdf
- Blaise_UK_112.pdf

Code

- Blaise114_115_CodeAll.zip
- Blaise13_Code113_All.zip
- Blaise_112_Martin-Debug-Part-3.zip
- Blaise_112_DavidDirkse_Code.zip
- Blaise_112_Canneyt_Processcontrol-code.zip
- Blaise_110_111_guess.zip
- Blaise_109_pdfindex-2-code.zip
- Blaise_109_-Debug_Program.zip

WHAT IS A COMPONENT AND WHAT IS ITS USEFULNESS

Delphi is an application development environment that is based is based on the use of components. This means that developing an application with **Delphi** is done at least partly by placing components from the component palette on a Form.

Such a component then contains data and functionality and the user (*of course as opposed to the creator of a component*) does not have to worry about its implementation.
Among the components we distinguish **visual** and **non-visual ones**.
The former include buttons, dialogs and so on.
The second kind includes **TTimer**, **TDataSet**, and so on.

There can be all kinds of **reasons for developing a component.**
Two of the most important of which are:

- something is needed that can be **easily reused**;
- there is **no existing component for that purpose**.

In addition, it could just be play, whereby someone, by making a couple of components gets a better understanding of the "**Visual Component Library (VCL)**" and its use in creating **Delphi** applications.

In this article, the creation of **two non-visua**l components is discussed.
In principle these could also be created and used as separate **units**, but it has several advantages to create them as components. They can then be placed directly from the component palette on a Form and thus incorporated into an application and in doing so, the **unit name** associated with the component is automatically included in the **uses clause**.

LISTING ❶.
The basic component unit as created by **Delphi**
Of interest is the division into **private, protected, public** and **published sections**.

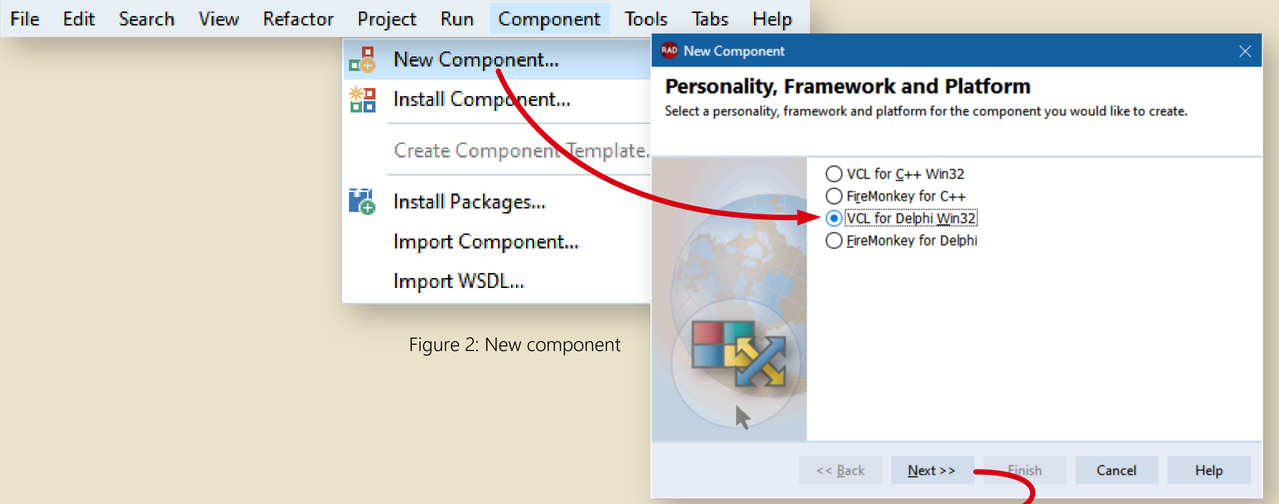
The following important rules apply to this:

- the **private** section is for **internal use** in a component only, can only be used in the class unit and **cannot be applied directly by the user of a component**;
- what is declared in the **protected** section can be seen within the component's class unit and in any new class derived from it;
- the **public** section is **used for the runtime interface** and **everything declared in this section can be seen and used by all components** of the application and in it are placed those properties and methods that the user of a component during the running of the application;
- the published section is analogous to the public section but however, the properties declared in this section are visible in the the **Object Inspector** and can therefore **be modified** there to the user's requirements.

In the present case, only one **method** is needed in the unit which can be seen in **Listing 2** of the **unit UDelay** and with it the delay is set.
It is included under the **Public** declarations because it must, of course, be able to be called from the program in which a delay is needed, **it should be callable**.

This call, when the component in the relevant program is included with the name **Delayer**, it takes the form: `Delayer.Delay(2000)` where the number is the desired delay time in milliseconds, i.e.

For the **required time measurement** use the function **GetTickCount** which gives the number of milliseconds since **Windows** last started. The quantity (`GetTickCount - FirstTick`) thus gives the number of milliseconds that elapsed after the time `FirstTick`.
The result of `GetTickCount` becomes zero again after **Windows** has been running for about **49.7** days of continuous running.
That is about **2^32** milliseconds, so the result of this function is apparently an "**unsigned integer**".



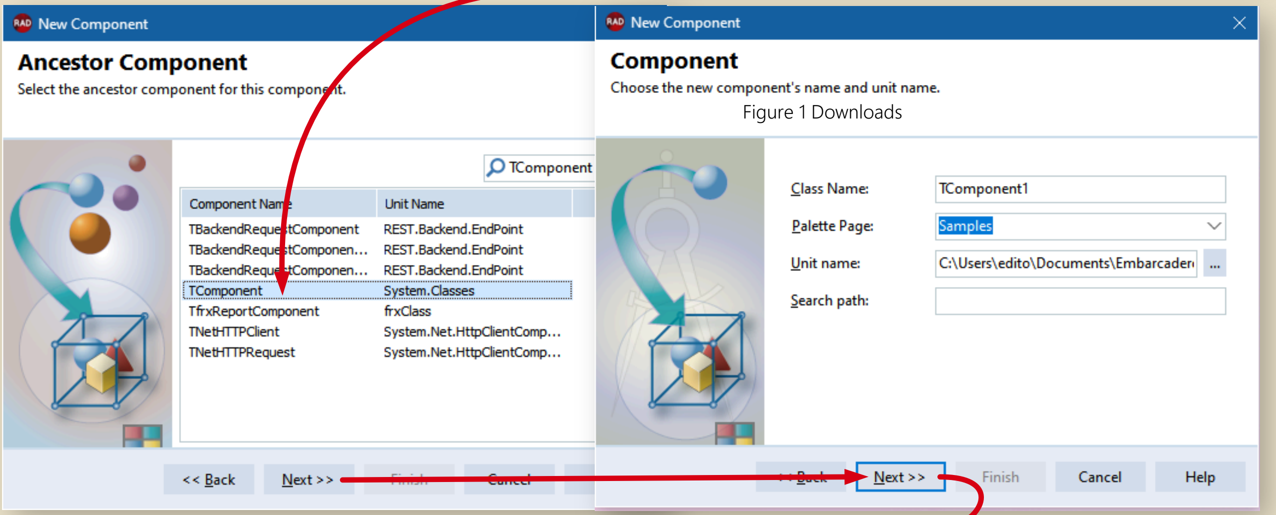
New Component

Personality, Framework and Platform

Select a personality, framework and platform for the component you would like to create.

○ VCL for C++ Win32
○ FireMonkey for C++
● VCL for Delphi Win32
○ FireMonkey for Delphi

Figure 2: New component



New Component

Ancestor Component

Select the ancestor component for this component.

Component Name	Unit Name
TBackendRequestComponent	REST.Backend.EndPoint
TBackendRequestComponen...	REST.Backend.EndPoint
TBackendRequestComponen...	REST.Backend.EndPoint
TComponent	System.Classes
TfrxReportComponent	frxClass
TNetHTTPClient	System.Net.HttpClientComp...
TNetHTTPRequest	System.Net.HttpClientComp...

New Component

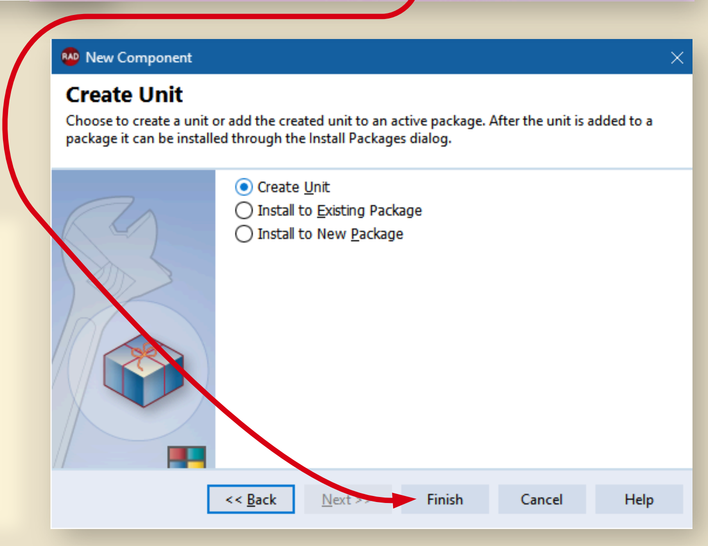
Component

Choose the new component's name and unit name.

Figure 1 Downloads

Class Name: TComponent1
Palette Page: Samples
Unit name: C:\Users\edito\Documents\Embarcader
Search path:

Figure 3 search for the Ancestor



New Component

Create Unit

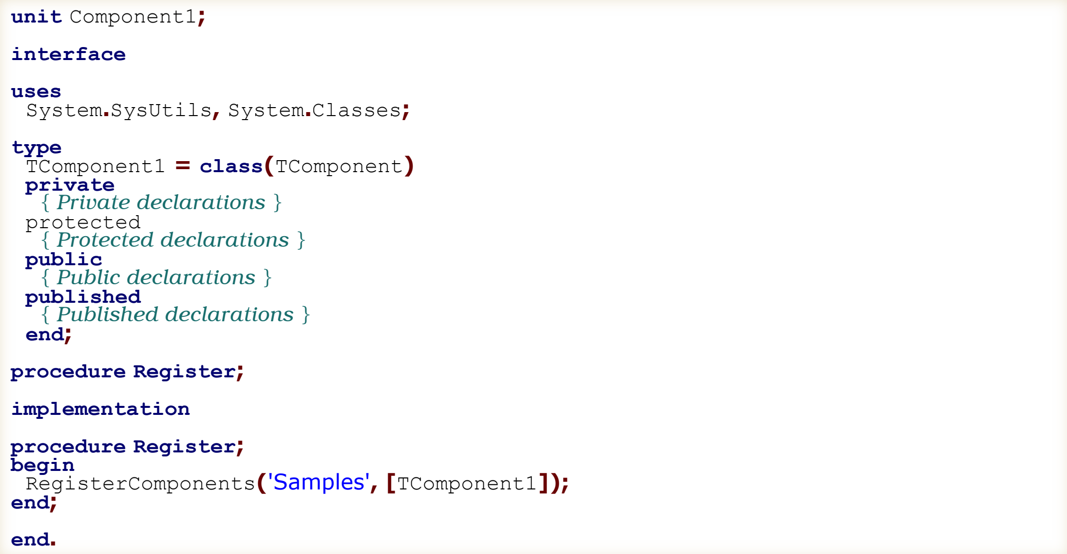
Choose to create a unit or add the created unit to an active package. After the unit is added to a package it can be installed through the Install Packages dialog.

● Create Unit
○ Install to Existing Package
○ Install to New Package

Figure 4 Create the unit of the component

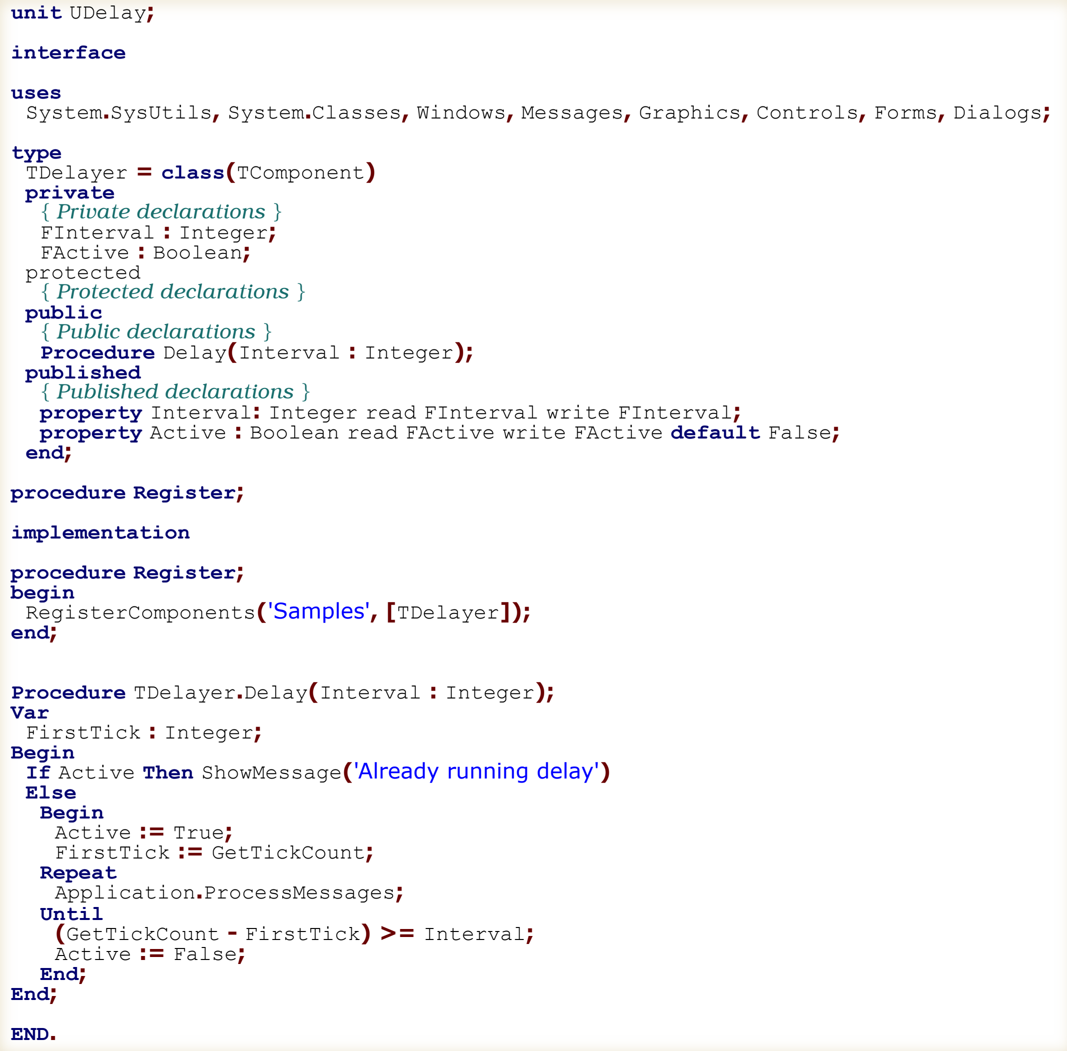
Once you want to create the **delay component listing** there is a **code example** on *page 4/x* of this article.
If you want to **rename the unit** you created because it has the wrong name, then you have to start the process again, or create the form manually without the wizard that sets it all up for you.
Just copy the code you already have or created and paste it in the form you saved with the correct name.
Save again.

Listing 1



```
unit Component1;

interface

uses
  System.SysUtils, System.Classes;

type
  TComponent1 = class(TComponent)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TComponent1]);
end;

end.
```

Listing 2



```
unit UDelay;

interface

uses
  System.SysUtils, System.Classes, Windows, Messages, Graphics, Controls, Forms, Dialogs;

type
  TDelayer = class(TComponent)
  private
    { Private declarations }
    FInterval : Integer;
    FActive : Boolean;
  protected
    { Protected declarations }
  public
    { Public declarations }
    Procedure Delay(Interval : Integer);
  published
    { Published declarations }
    property Interval: Integer read FInterval write FInterval;
    property Active : Boolean read FActive write FActive default False;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TDelayer]);
end;


Procedure TDelayer.Delay(Interval : Integer);
Var
  FirstTick : Integer;
Begin
  If Active Then ShowMessage('Already running delay')
  Else
    Begin
      Active := True;
      FirstTick := GetTickCount;
    Repeat
      Application.ProcessMessages;
    Until
      (GetTickCount - FirstTick) >= Interval;
      Active := False;
    End;
End;

END.
```

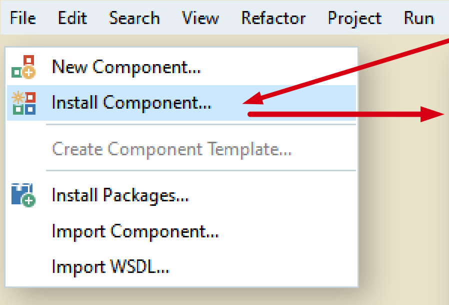
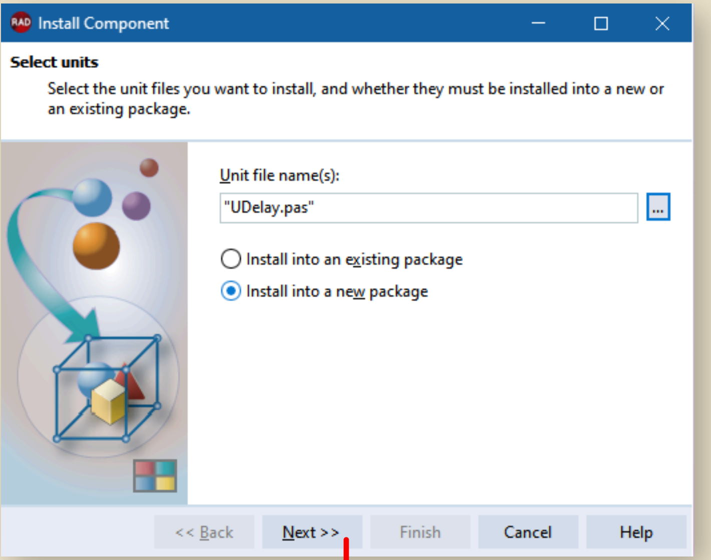


Figure 5 Install component



Select units

Select the unit files you want to install, and whether they must be installed into a new or an existing package.

Unit file name(s): "UDelay.pas"

○ Install into an existing package

● Install into a new package

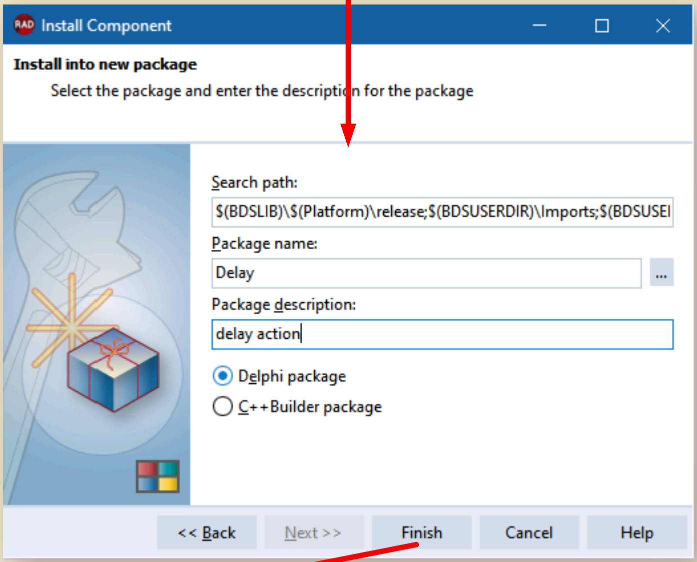
Now we can start the **installing of the component in the Delphi IDE**.

Go to:
Component > choose the next step: **Install Component.** Again a window appears: *See figure 5*.
By choosing the **three dotted button** (**Ellipsis**) the file system pops up and then go to where you saved your **UDelay.pas** (*or whatever name you have given it*).
Select → **Install to a new package.**
Click **Next**.
The search path appears (*See figure 7*.) and below that you can enter the package name.
The description should of course be some meaningful text. Click → **Finish**.

Its not all done yet:
Delphi wants to know what the **framework** will be:
See figure 8. There is little choice, so click OK and now the window (*See figure 9*) will appear. Package **...Delay.bpl has been installed.**

Now you can check if the component really is available:
Start creating a **new VCL program** and take a look in the
Palette → Sample → TDelayer

Figure 6 Unit File name



Install into new package

Select the package and enter the description for the package

Search path: $(BDSLIB)\$(Platform)\release;$(BDSUSERDIR)\Imports;$(BDSUSEI

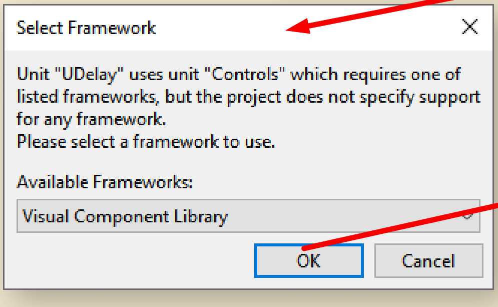
Package name: Delay

Package description: delay action

● Delphi package

○ C++Builder package

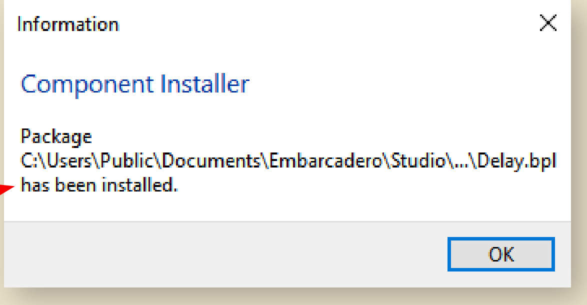
Figure 7: search path



Unit "UDelay" uses unit "Controls" which requires one of listed frameworks, but the project does not specify support for any framework.
Please select a framework to use.

Available Frameworks: Visual Component Library

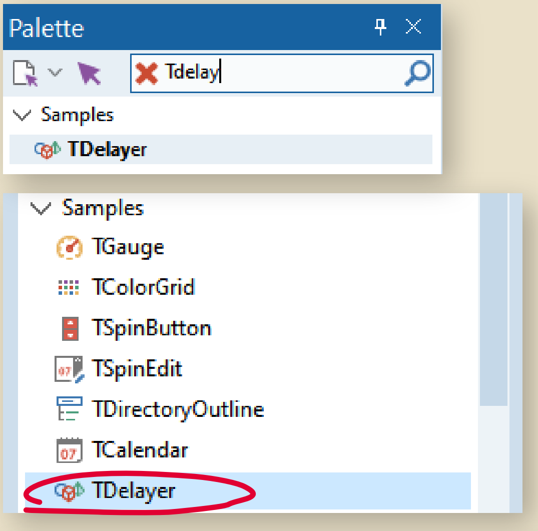
Figure 8: framework VCL



Component Installer

Package
C:\Users\Public\Documents\Embarcadero\Studio\...\Delay.bpl
has been installed.

Figure 9: final install worked



Palette

☒ Tdelay

Samples

TDelayer

Samples

TGauge

TColorGrid

TSpinButton

TSpinEdit

TDirectoryOutline

TCalendar

TDelayer

Figure 10:
the installed component

Furthermore, two fields: **`FInterval`** and **`FActive`** are defined of which the properties **`Interval`** and **`Active`** are included.

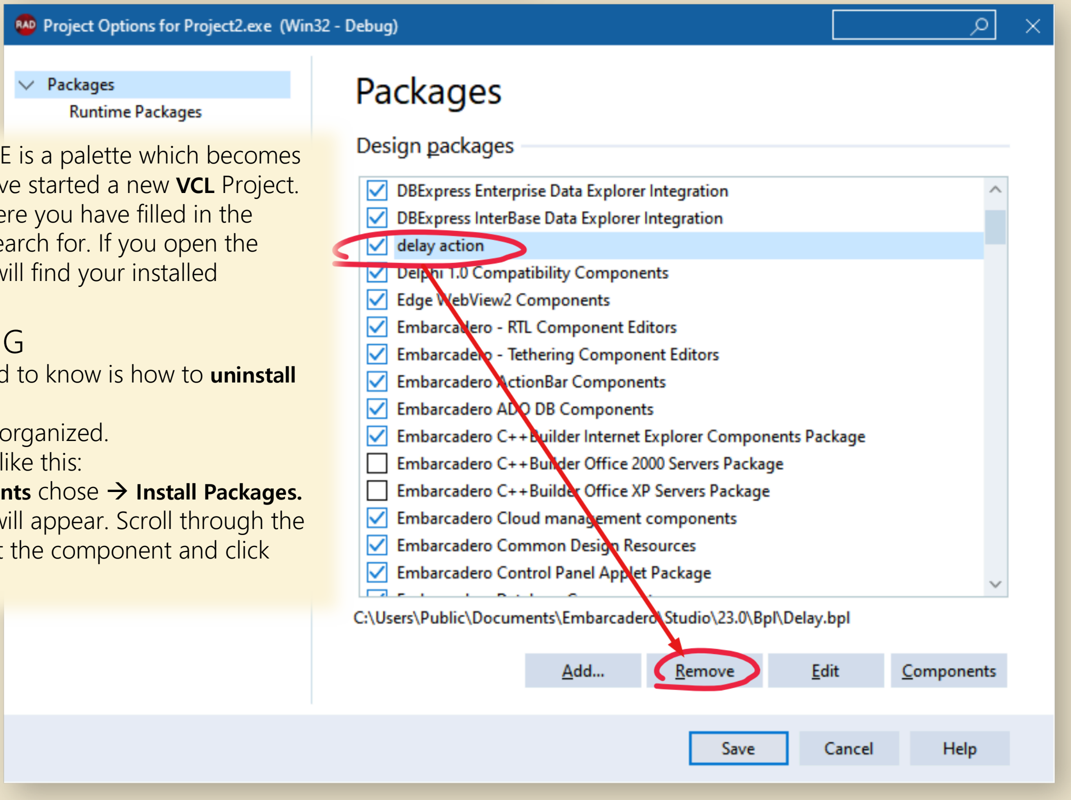
The property **`Active`** is intended to prevent a new delay from being started during delay, which is obviously not desirable and the correct interval not being used again!

Usually there is something wrong with the logic of a program. It is important for the writer of the program that an **error message** appears if such a thing is the case. If it is ok, the end-user of a program never gets this message to be seen.

Next, the unit is then saved with **File|Save** as choosing a more meaningful name, in my case **Udelay**.pas.

This is placed in a separate folder for components. I chose `c:\Components\TimeDelay\`.

Make your own choice



Project Options for Project2.exe (Win32 - Debug)

Packages
Runtime Packages

Packages

Design packages

☑ DBExpress Enterprise Data Explorer Integration
☑ DBExpress InterBase Data Explorer Integration
☑ delay action
☑ Delphi 1.0 Compatibility Components
☑ Edge WebView2 Components
☑ Embarcadero - RTL Component Editors
☑ Embarcadero - Tethering Component Editors
☑ Embarcadero ActionBar Components
☑ Embarcadero ADO DB Components
☑ Embarcadero C++Builder Internet Explorer Components Package
□ Embarcadero C++Builder Office 2000 Servers Package
□ Embarcadero C++Builder Office XP Servers Package
☑ Embarcadero Cloud management components
☑ Embarcadero Common Design Resources
☑ Embarcadero Control Panel Applet Package

C:\Users\Public\Documents\Embarcadero\Studio\23.0\Bpl\Delay.bpl

Figure 11:
uninstall is also important

Normally in the IDE is a palette which becomes available if you have started a new **VCL** Project. See image xxx where you have filled in the component you search for. If you open the **Samples** Tab you will find your installed component.

UNINSTALLING

What you still need to know is how to **uninstall** the component.

It's rather strange organized.

You need to do it like this:

go to → **Components** chose → **Install Packages.** The next window will appear. Scroll through the list and then select the component and click **remove**. That's all.

ANOTHER COMPONENT

We now describe, without explicitly repeating all the steps mentioned in the previous section mentioned, a second component:
`TClock`, which, like the previous one, is derived from **`TComponent`** and placed on the **Sample** page of the component palette. This component gets two functions and one property, and the entire unit is shown in **Listing 3:**

Listing 3. Source code of the unit `UClock`.
Here again we see the use of the `GetTickCount` function. The `GetStart` function is used to set the start time, while at the end of the period whose duration is to be measured, the function **`TClock.Time`** is called, which as a result at once gives the **string for the elapsed time.**

To also be able to determine intermediate times, e.g. in addition to the total time taken by a calculation, also that of intermediate operations, the array the array `st[0..3]` is used to record the individual start times in it. In the constructor `Create` its initial value is set to **`0.`**

Including the component on the component palette, including compiling proceeds exactly as described for the previous component. In this case, given the purpose of this component, there is no need for the presence of an adjustable property.
Although here you could therefore also use a unit instead of a component, inclusion in an application is, as already mentioned above, easier for a component than for a unit.

TEST PROGRAM

To **test** the components discussed above, we created a small program called **DelayTesting** with one unit: **`UDelay.pas`**. In it, we define the form **TestForm** and on it we place on a panel (`TestPanel`) the two components, six labels, two Buttons and an `Edit` field as ***shown in Figure 6.***
At the heart of the program are the procedures **`TestBtnClick`** and **`CalcButtonClick`** which are shown in **Listing 4,** *see next page*. Furthermore, we see the **`TestMemo`** in which the results of the **`ArithmeticButtonClick`** procedure are shown.

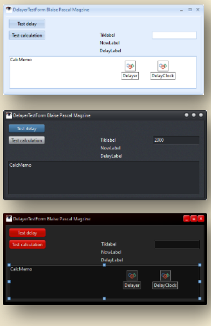


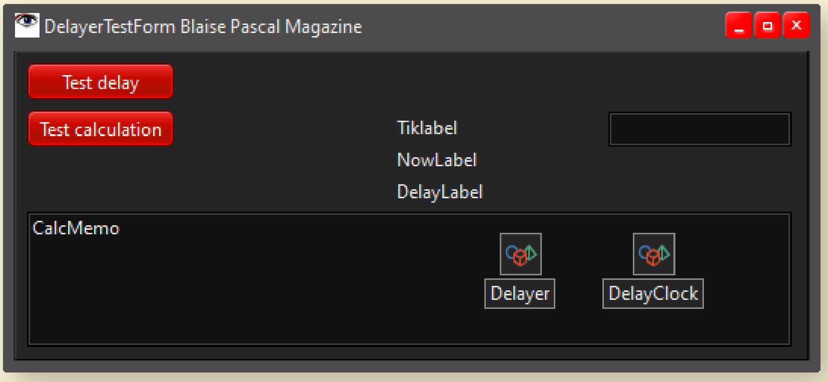
Figure
Delphi offers a special way of showing the graphical possibilities:

Go to → Project
→Options → Application
→Appearance

Listing 4. The procedures `FormCreate`, `TestBtnClick` and `CalculateButtonClick` from the unit `UTestTime`. (*See page 8 of this article*)
In the **`FormCreate`** procedure, the default value of the Interval in the **`IntervalEdit`** window. This or a modified value of this number is used as the interval for the **Delay**.
After expiration of that interval, the **`DelayLabel`** field displays the elapsed time.

When everything works correctly, the result is equal to the set delay. In most cases, this is correct within a few milliseconds and so that is apparently the accuracy of the time measurement.

In the **`ArithmeticButtonClick`** procedure is both - the **total time measured -** as well as the **time for each calculation** separately. This latter can be important, when between individual calculations intermediate operations take place and one also wants to determine their influence. In this case a delay as an example.



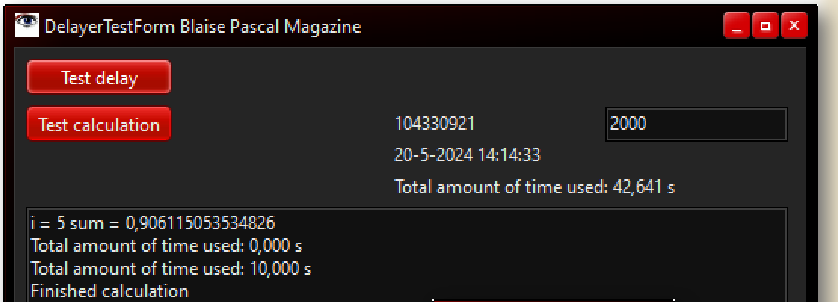
Listing 3

```
Unit UClock;

Interface

Uses
 System.SysUtils, System.Classes, Windows, Messages;

Type
 TDelayClock = class(TComponent)

 Private   { Private declarations }
   FStart : Integer;
   nPartTime: Integer;
   st : Array[0..3] OF Integer;

 Protected    { Protected declarations }
 Public   { Public declarations }
  Constructor Create(AOwner : TComponent); override;

  Function  GetStart : Integer;
  Function  Time : STRING;

  Property Start : Integer read GetStart;

 Published
  { Published declarations }
End;

Procedure Register;

Implementation

Procedure Register;
begin
 RegisterComponents('Samples', [TDelayClock]);
end;

Constructor TDelayClock.Create (AOwner : TComponent);
Begin
  Inherited Create(AOwner);
  nPartTime := 0;
End;

Function TDelayClock.GetStart : Integer;
Begin
  If nPartTime < 3 Then inc(nPartTime);
  st[nPartTime] := GetTickCount;
End;

Function TDelayClock.Time : String;
Var
 iExpireTime   : Integer;
 hour, min     : Word;
 sec           : extended;
 tmp           : String;

Begin
 iExpireTime := GetTickCount - st[nPartTime];
 hour        := iExpireTime  DIV 3600000;
 iExpireTime := iExpireTime  MOD 3600000;
 min         := iExpireTime  DIV 60000;
 iExpireTime := iExpireTime  MOD 60000;
 sec         := 0.001 * iExpireTime ;

 If hour > 0
 Then tmp := tmp + Format('%2.1d u %2.1d m %5.3f s', [hour, min, sec])
 Else
  If min > 0
  Then tmp := tmp + Format('%2.1d m %5.3f s', [min, sec])
  Else tmp := tmp + Format('%5.3f s', [sec]);

 Result := tmp;
 dec(nPartTime);
End;

end.
```

Listing 4

```
unit DelayTest;

interface

uses Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,
  Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls,
  UClock, UDelay;

type
 TTestForm = class(TForm)
  DelayClock: TDelayClock;
  TestPanel: TPanel;
  TestBtn: TButton;
  CalcButton: TButton;
  Tiklabel: TLabel;
  NowLabel: TLabel;
  DelayLabel: TLabel;
  Delayer: TDelayer;
  IntervalEdit: TEdit;
  CalcMemo: TMemo;
  procedure FormCreate(Sender: TObject);
  procedure TestBtnClick(Sender: TObject);
  procedure CalcButtonClick(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 TestForm: TTestForm;

implementation

{$R *.dfm}

Procedure TTestForm.FormCreate(Sender: TObject);
Begin
 IntervalEdit.Text := IntToStr(Delayer.Interval);
End;

Procedure TTestForm.TestBtnClick(Sender: TObject);
Begin
 NowLabel.Caption    := DateTimeToStr(Now);
 TikLabel.Caption    := IntToStr(GetTickCount);
 DelayLabel.Caption := '';
 Delayer.Interval    := StrToInt(IntervalEdit.Text);
 DelayClock.GetStart;
 Delayer.Delay(Delayer.Interval);
 DelayLabel.Caption := 'Total amount of time used: ' + DelayClock.Time;
End;

Procedure TTestForm.CalcButtonClick(Sender: TObject);
Var   i, j : Integer;  Sum : extended;
Begin
 DelayClock.GetStart;
 For i := 1 TO 5 DO
  Begin
   sum := 0.0;
   DelayClock.GetStart;

   FOR j := 1 TO 10000 DO
   sum := sum + i / (i + j) / (i + j);

   CalcMemo.Lines.Add('i = ' + IntToStr(i) + ' sum = ' + FloatToStr(sum));
   CalcMemo.Lines.Add('Total amount of time used: ' + DelayClock.Time);
   Delayer.Delay(Delayer.Interval);
  End;
 CalcMemo.Lines.Add('Total amount of time used: ' + DelayClock.Time);
 CalcMemo.Lines.Add('Finished calculation');
End;

end.
```

Figure 13:
the running project



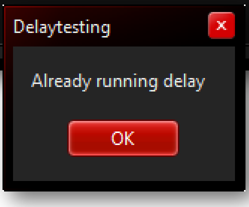
104330921

2000

20-5-2024 14:14:33

Total amount of time used: 42,641 s

```
i = 5 sum = 0,906115053534826
Total amount of time used: 0,000 s
Total amount of time used: 10,000 s
Finished calculation
```

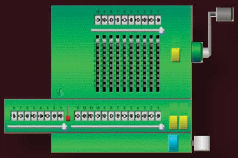


Already running delay

Figure 14: the error message

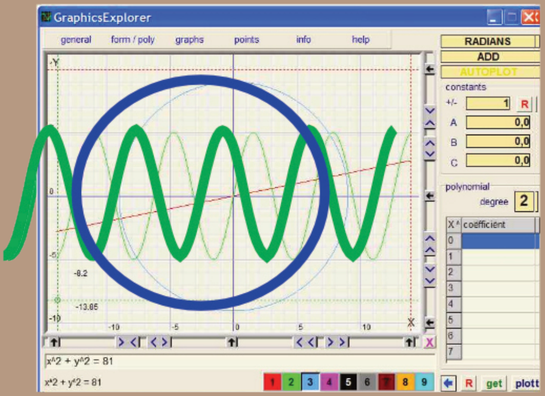
ADVERTISEMENT

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



DAVID DIRKSE

including 50 example projects



+/- 1

A 0,0

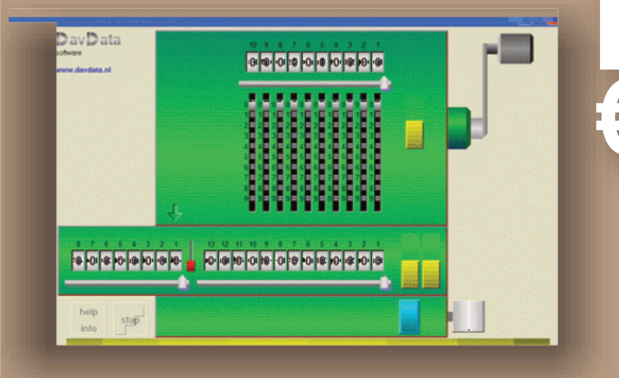
B 0,0

C 0,0

degree 2

x^2 + y^2 = 81

x^2 + y^2 = 81



Pocket
€ 50
ex Shipping
PDF
€ 35

COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

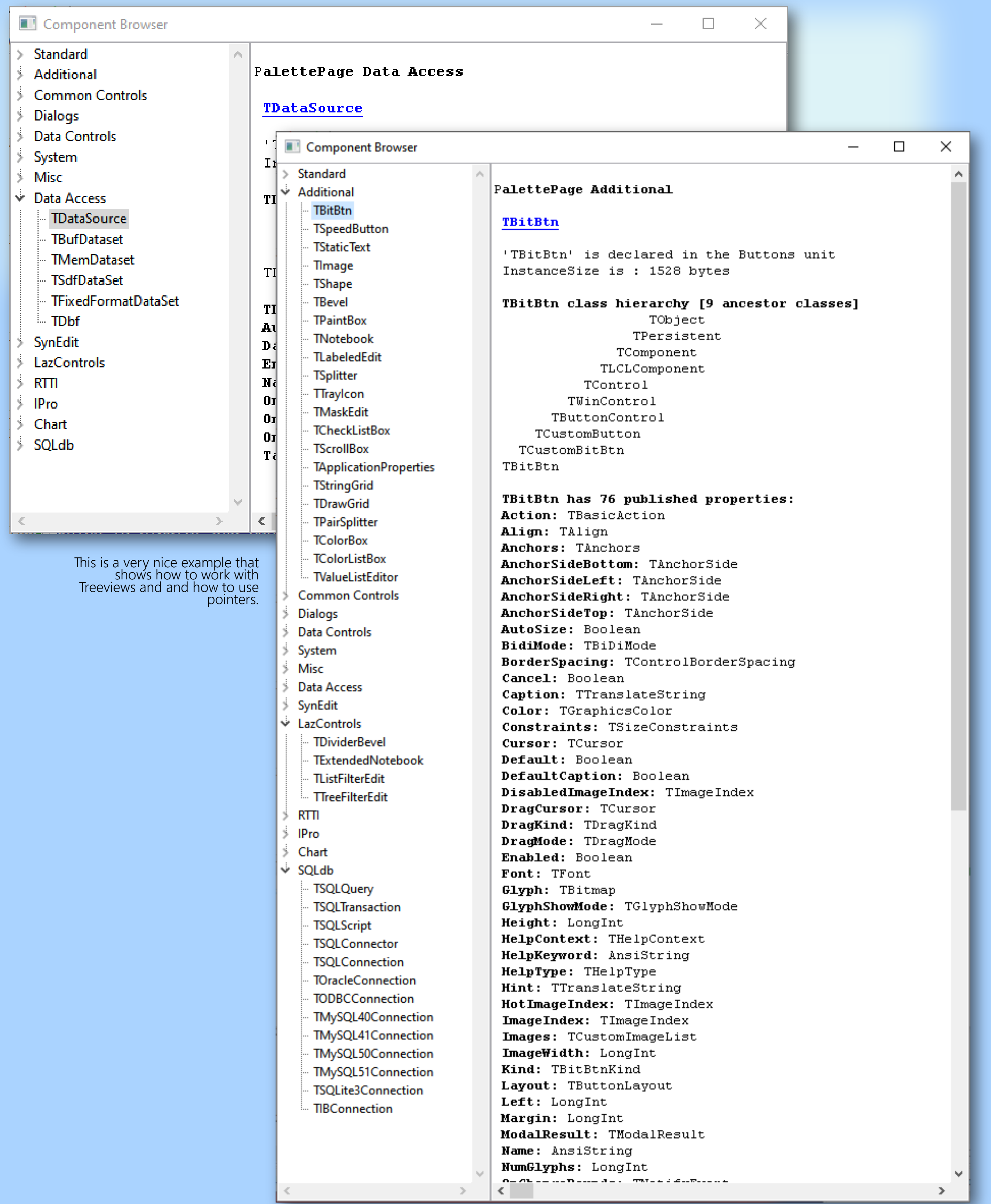
`https://www.blaisepascalmagazine.eu/product-category/books/`

PUTS:
PASCAL USERS TIPS & SOLUTIONS

Some weeks ago I had a reaction from **Ettore Cicinelli** from Italy
and he asked for a solution .He had a problem with a project from the book Learning to program using Lazarus: a very nice example, very useful but not yet checked with the latest version of Lazarus 3.2.
It did not compile in the right manner, so I asked **Ignace Peeters** from Belgium if he could take a look at it.
Ignace solved the problem and here it is:

There was an error in the code:
In the StringList, an object is added (***descending from*** `TStrings`) via `AddObject`.
When this object is requested afterwards, it is already an object, **it does not need to be converted** to `TObject`.
The Pointer conversion that used to be there: `TObject(`**`Pointer`**`(Length(APPI^.name))`
has been replaced by `TObject(`**`PtrUint`**`(Length(....)))` which is clearer and easier because this is an unsigned object. If you then do the reverse operation via **`PtrUnt`** afterwards when excavating, everything works.

```
procedure TForm1.DisplayComponentProperties;
var
    aPPI: PPropInfo;
    aPTI: PTypeInfo;
    aPTD: PTypeData;
    aPropList: PPropList;
    sortSL: TStringList;
    i: integer;
    s: string;

begin
  seViewer.Lines.Add(''); inc(lineNo);
  aPTI := PTypeInfo(compClass.ClassInfo);
  aPTD := GetTypeData(aPTI);
  s := Format(' %s has %d published properties:',[aPTI^.Name, aPTD^.PropCount]);
  hiliter.AddToken(lineNo, 1, tkText); hiliter.AddToken(lineNo, Length(s), atrBD);
  seViewer.Lines.Add(s); inc(lineNo);

  if (aPTD^.PropCount = 0)
    then seViewer.Lines.Add('   (no published properties)')
  else
    begin
      Getmem(aPropList, SizeOf(PPropInfo)* aPTD^.PropCount);
      sortSL := TStringList.Create;
      sortSL.Sorted:= true;

      try
        GetPropInfos(aPTI, aPropList);
        for i := 0 to aPTD^.PropCount - 1 do
          begin
            aPPI := aPropList^[i];
            sortSL.AddObject(Format(' %s: %s', [aPPI^.Name, aPPI^.PropType^.Name]),
    Commented // TObject(Pointer(Length(aPPI^.Name))));
                 TObject(PtrUInt(Length(aPPI^.Name))));
          end;

        for i := 0 to sortSL.Count - 1 do
          begin
            seViewer.Lines.Add(sortSL[i]);
  Comm/ // hiliter.AddToken(lineNo, Succ(Integer(sortSL.Objects[i])), atrBD);
           hiliter.AddToken(lineNo, Succ(PtrUInt(sortSL.Objects[i])), atrBD);
            hiliter.AddToken(lineNo, Length(sortSL[i]), tkText);
            inc(lineNo);
          end;
        finally
          Freemem(aPropList, SizeOf(PPropInfo)* aPTD^.PropCount);
          sortSL.Free;
        end;

     end;
end;
```



```
PalettePage Data Access

TDataSource

'T
Ir

TI

TI

TI
Au
Da
En
Na
On
On
On
Ta
```

```
PalettePage Additional

 TBitBtn

 'TBitBtn' is declared in the Buttons unit
 InstanceSize is : 1528 bytes

 TBitBtn class hierarchy [9 ancestor classes]
                  TObject
                TPersistent
              TComponent
            TLCLComponent
          TControl
        TWinControl
      TButtonControl
    TCustomButton
  TCustomBitBtn
 TBitBtn

 TBitBtn has 76 published properties:
 Action: TBasicAction
 Align: TAlign
 Anchors: TAnchors
 AnchorSideBottom: TAnchorSide
 AnchorSideLeft: TAnchorSide
 AnchorSideRight: TAnchorSide
 AnchorSideTop: TAnchorSide
 AutoSize: Boolean
 BidiMode: TBiDiMode
 BorderSpacing: TControlBorderSpacing
 Cancel: Boolean
 Caption: TTranslateString
 Color: TGraphicsColor
 Constraints: TSizeConstraints
 Cursor: TCursor
 Default: Boolean
 DefaultCaption: Boolean
 DisabledImageIndex: TImageIndex
 DragCursor: TCursor
 DragKind: TDragKind
 DragMode: TDragMode
 Enabled: Boolean
 Font: TFont
 Glyph: TBitmap
 GlyphShowMode: TGlyphShowMode
 Height: LongInt
 HelpContext: THelpContext
 HelpKeyword: AnsiString
 HelpType: THelpType
 Hint: TTranslateString
 HotImageIndex: TImageIndex
 ImageIndex: TImageIndex
 Images: TCustomImageList
 ImageWidth: LongInt
 Kind: TBitBtnKind
 Layout: TButtonLayout
 Left: LongInt
 Margin: LongInt
 ModalResult: TModalResult
 Name: AnsiString
 NumGlyphs: LongInt
```

This is a very nice example that shows how to work with Treeviews and and how to use pointers.

INTRODUCTION:

We introduced Database Workbench 6 back in 2022, a lot has changed since then.
But now, first things first:

WHAT IS DATABASE WORKBENCH?

A complete database development tool with **native support** for **Oracle, SQL Server, MySQL,MariaDB, PostgreSQL, Firebird, NexusDB, InterBase** and **SQLite.**
Database Workbench offers a well-ordered, clear and consistent user interface for different database systems and provides access to database system specific features. It is very user friendly and the GUI is made in away that people can understand most of it with simply using your expectations and logic. **Which is worth a great compliment.**

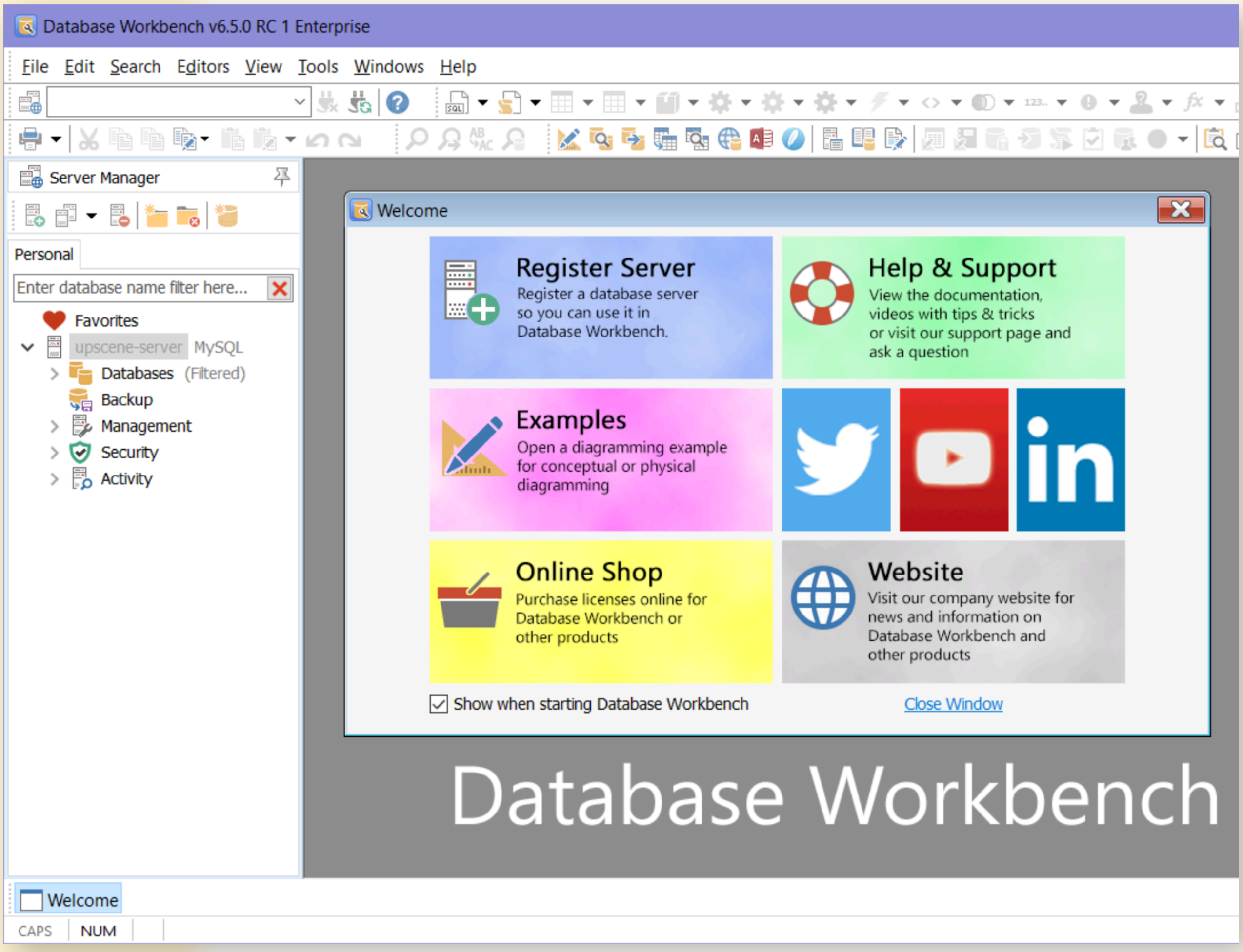

Database Workbench v6.5.0 RC 1 Enterprise
File Edit Search Editors View Tools Windows Help
Server Manager
Personal
Enter database name filter here...
Favorites
upscene-server MySQL
Databases (Filtered)
Backup
Management
Security
Activity
Welcome
Register Server
Register a database server so you can use it in Database Workbench.
Help & Support
View the documentation, videos with tips & tricks or visit our support page and ask a question
Examples
Open a diagramming example for conceptual or physical diagramming
Online Shop
Purchase licenses online for Database Workbench or other products
Website
Visit our company website for news and information on Database Workbench and other products
☑ Show when starting Database Workbench
Close Window
Database Workbench
Welcome
CAPS
NUM

Figure 1:
Database Workbench
main window / a larger

You can use this application for your database development from start to finish:
start with **designing graphically**, end with **testing queries** and **debugging stored procedures**.

And somewhere in between, **you can create schema objects** like tables, indices, constraints, **generate data or import data from legacy systems** for testing purposes, and document your database with printing or **maintain your database with to-do lists and version control**.

EDITIONS, MODULES, LICENSING AND SUPPORT

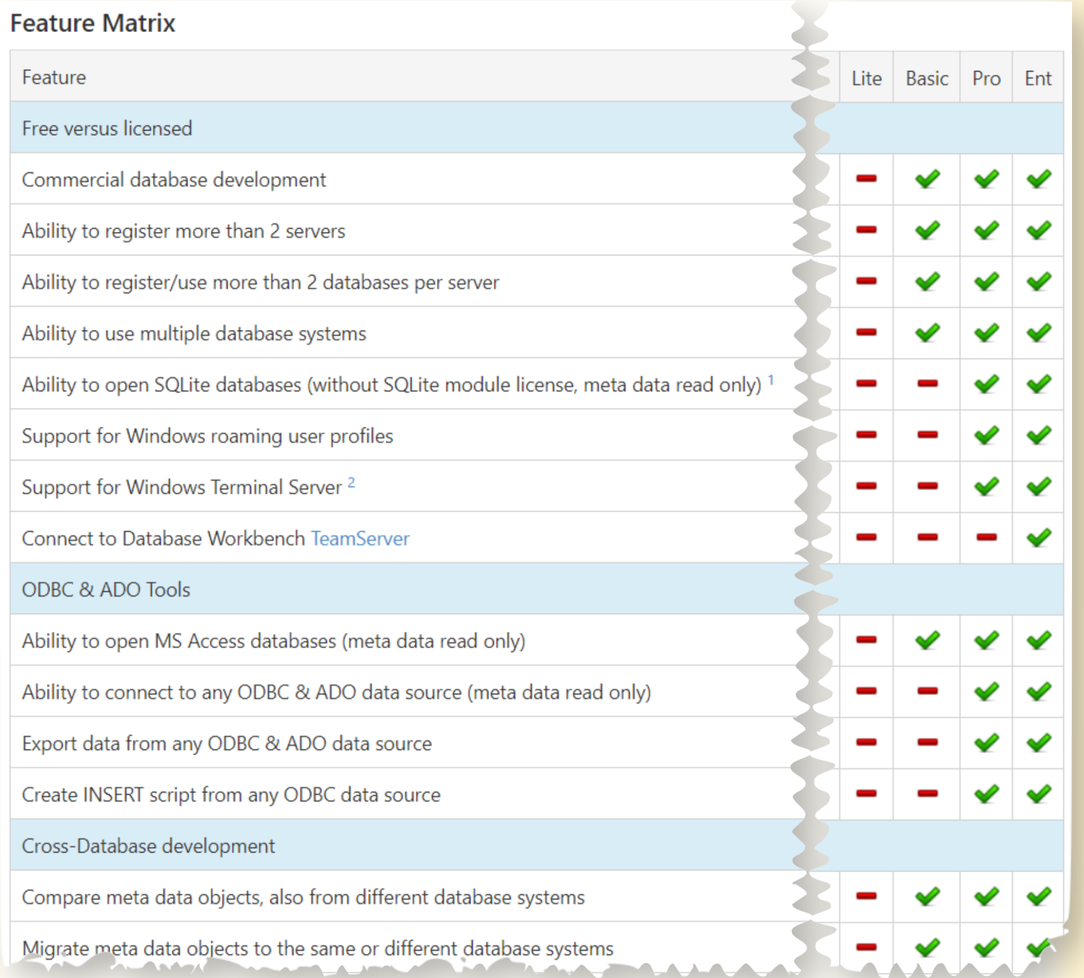
Database Workbench comes in **3 editions**: Basic, Pro and Enterprise.
The license itself is not time limited, updates are included for the first year.
You can purchase a subscription beyond your first year for additional updates for the current major version. Each license has to include at least 1 module for a specific database system.

Such a module provides **native access to a database** system and includes functionality to **design, create and modify databases**. Additional modules can be added at a later time.
The **Basic edition** covers the needs for most developers, but doesn't include visual database design, although you can reverse engineer an existing database for documentation purposes.
With the **Pro edition**, you can design your database with logical and physical models.
It adds the ability to debug stored procedures and triggers, and open any ADO or ODBC data source (*in meta data read only mode*) or convert from those to supported database systems.

The latest version of the Pro edition also allows you to open **SQLite databases** in meta data read only mode without licensing the SQLite module, there's **full SQLite support** with the module added to your license.
More features for the Pro Edition: transfer data to and from ADO or ODBC data sources, print schema objects, create custom reports that can be used for multiple databases, support for roaming Windows profiles and Windows Terminal Server, and additional productivity features like favorite databases, integrated To-Do lists and SQL/Code Catalogs.

The **Enterprise editio**n is created for development teams:
adding a **central repository or registered servers and databases**, specific multi-user features and a built-in **Version Control System** for database objects.
Modular licensing in combination with the different editions with different prices makes sure there's a suitable option for everyone.

Figure 2:
Feature matrix with detailed edition differences



Feature Matrix

Feature	Lite	Basic	Pro	Ent
Free versus licensed				
Commercial database development	−	✓	✓	✓
Ability to register more than 2 servers	−	✓	✓	✓
Ability to register/use more than 2 databases per server	−	✓	✓	✓
Ability to use multiple database systems	−	✓	✓	✓
Ability to open SQLite databases (without SQLite module license, meta data read only) [1]	−	−	✓	✓
Support for Windows roaming user profiles	−	−	✓	✓
Support for Windows Terminal Server [2]	−	−	✓	✓
Connect to Database Workbench TeamServer	−	−	−	✓
ODBC & ADO Tools				
Ability to open MS Access databases (meta data read only)	−	✓	✓	✓
Ability to connect to any ODBC & ADO data source (meta data read only)	−	−	✓	✓
Export data from any ODBC & ADO data source	−	−	✓	✓
Create INSERT script from any ODBC data source	−	−	✓	✓
Cross-Database development				
Compare meta data objects, also from different database systems	−	✓	✓	✓
Migrate meta data objects to the same or different database systems	−	✓	✓	✓

A detailed feature matrix is available here:
`https://www.upscene.com/database_workbench/editions`

CONCEPTUAL AND PHYSICAL DATABASE DESIGN

A data modeling tool can be useful during the database design phase and **Database Workbench** includes a **diagram editor** with which you can create 2 types of model, a "logical model" and a "physical model".

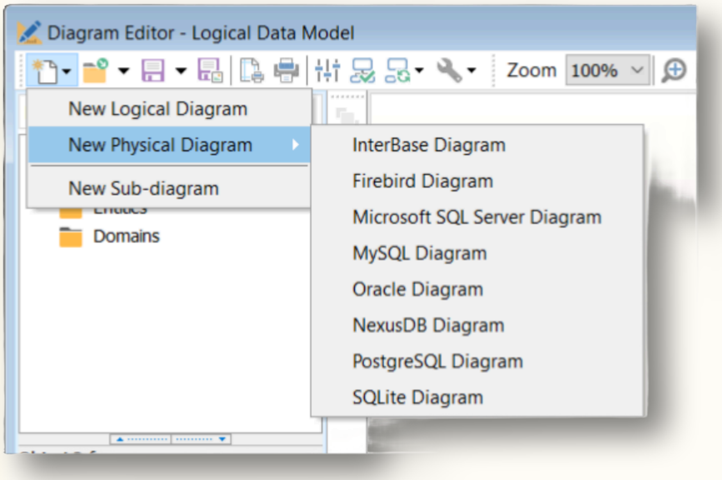


Figure3
New Diagram menu

The **physical** model is the actual database, it contains tables, views and their relationships via foreign key constraints. You can **visually** design a database this way or **reverse engineer** an existing database, for example for documentation purposes.

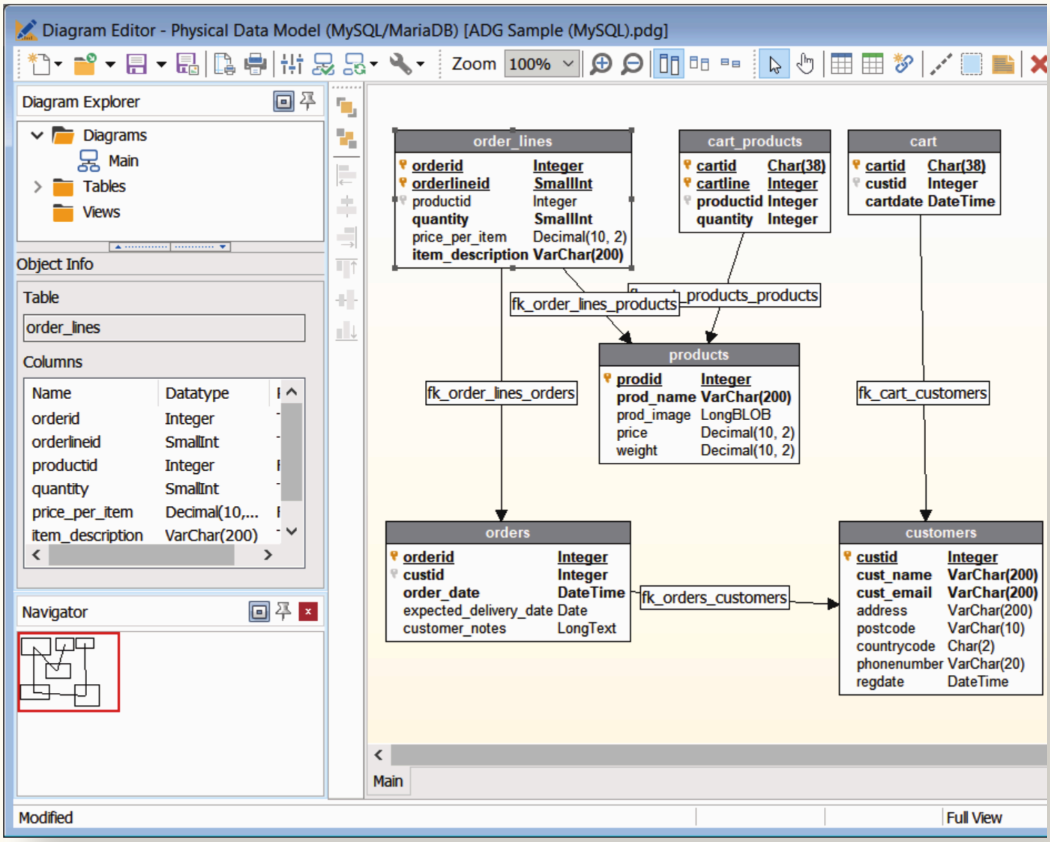


Table
order_lines

Columns

Name	Datatype
orderid	Integer
orderlineid	SmallInt
productid	Integer
quantity	SmallInt
price_per_item	Decimal(10,...
item_description	VarChar(200)

order_lines	
orderid	Integer
orderlineid	SmallInt
productid	Integer
quantity	SmallInt
price_per_item	Decimal(10, 2)
item_description	VarChar(200)

cart_products	
cartid	Char(38)
cartline	Integer
productid	Integer
quantity	Integer

cart	
cartid	Char(38)
custid	Integer
cartdate	DateTime

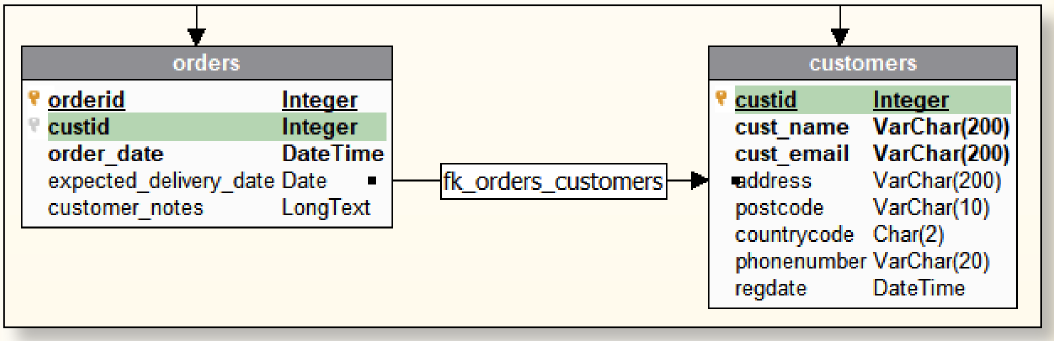
products	
prodid	Integer
prod_name	VarChar(200)
prod_image	LongBLOB
price	Decimal(10, 2)
weight	Decimal(10, 2)

orders	
orderid	Integer
custid	Integer
order_date	DateTime
expected_delivery_date	Date
customer_notes	LongText

customers	
custid	Integer
cust_name	VarChar(200)
cust_email	VarChar(200)
address	VarChar(200)
postcode	VarChar(10)
countrycode	Char(2)
phonenumber	VarChar(20)
regdate	DateTime

Figure 4:
A physical data model in the Diagram Editor.

The **physical model** (the database) is an implementation of certain business logic, eg in a webshop database, each "customer" can have orders. Each "order" record should point to an existing "customers" record.
This logic is physically implemented by a CUSTOMERS table, an ORDERS table and a foreign key constraints between those tables with a CUSTOMERID column in the ORDERS table.

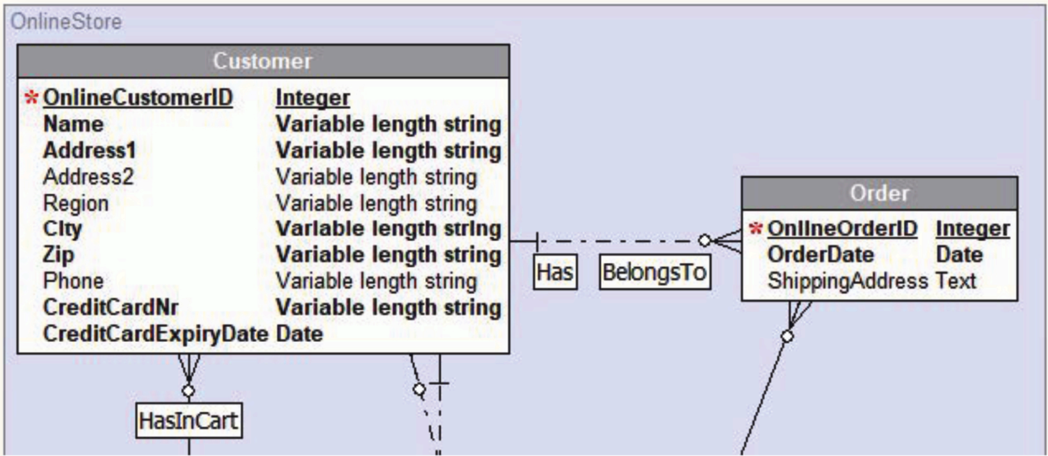


orders	
orderid	Integer
custid	Integer
order_date	DateTime
expected_delivery_date	Date
customer_notes	LongText

customers	
custid	Integer
cust_name	VarChar(200)
cust_email	VarChar(200)
address	VarChar(200)
postcode	VarChar(10)
countrycode	Char(2)
phonenumber	VarChar(20)
regdate	DateTime

Figure 5:
A physical model foreign key constraint from ORDERS to CUSTOMERS

The **"logical model"** is database agnostic, more abstract and it does not care about actual implementation, you're only modeling the business logic.
A logical model can then be used to generate the physical model: the actual database.
Logical data models consist of entities and their relationships, a so-called entity-relationship model (ER-model). A "customer" is an entity which has a relation to the "order" entity.

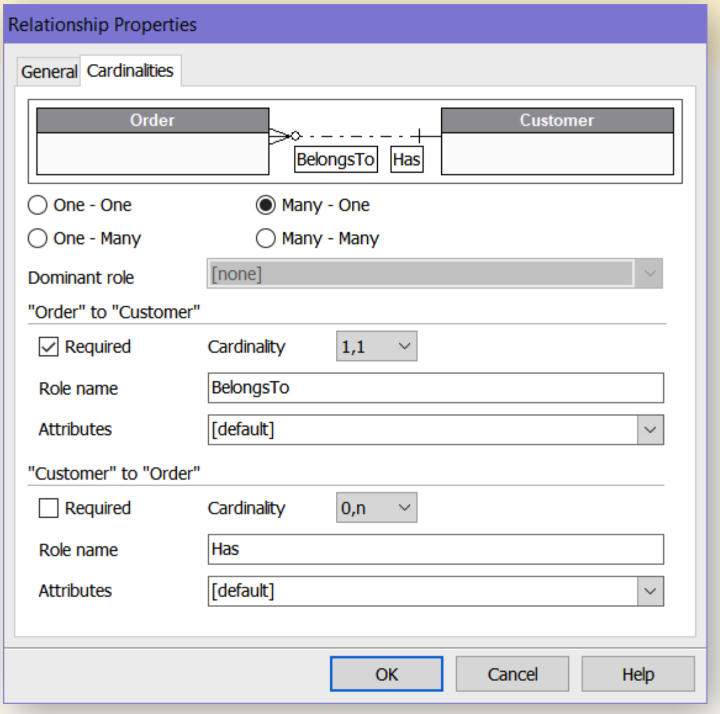


Customer	
*OnlineCustomerID	Integer
Name	Variable length string
Address1	Variable length string
Address2	Variable length string
Region	Variable length string
City	Variable length string
Zip	Variable length string
Phone	Variable length string
CreditCardNr	Variable length string
CreditCardExpiryDate	Date

Order	
*OnlineOrderID	Integer
OrderDate	Date
ShippingAddress	Text

Figure 6:
A logical model relationship

As you can see here, you don't have a reference to the CUSTOMER entity in the ORDER entity, but there is only a logical relationship between the two. This makes for a cleaner model with the focus on the actual logic instead of how to implement such a relationship in your database.



Relationship Properties

General | Cardinalities

○ One - One ● Many - One

○ One - Many ○ Many - Many

Dominant role: [none]

"Order" to "Customer"

☑ Required — Cardinality: 1,1

Role name: BelongsTo

Attributes: [default]

"Customer" to "Order"

□ Required — Cardinality: 0,n

Role name: Has

Attributes: [default]

Figure 7: Relationship properties in a logical model

There are different types of relationships in an ER-model and these relationships have different implementations in a database.
A relationship can be "**identifying**" or "**non-identifying**" and it has cardinality,
with the number ratio expressed in symbols, like one-to-one or one-to-many. An identifying relationship means the dependent entity can only be identified when also using the owner entity identifier. When generating the physical model, the identity identifier becomes the primary key and for an identifying relationship, the primary key of the parent will be part of the primary key of the childtable.

With a non-identifying relationship the child entity can be identified without using the owner entity. When generating the physical model, the primary key of the owner will be referred to in the child, but the child will have its own primary key.

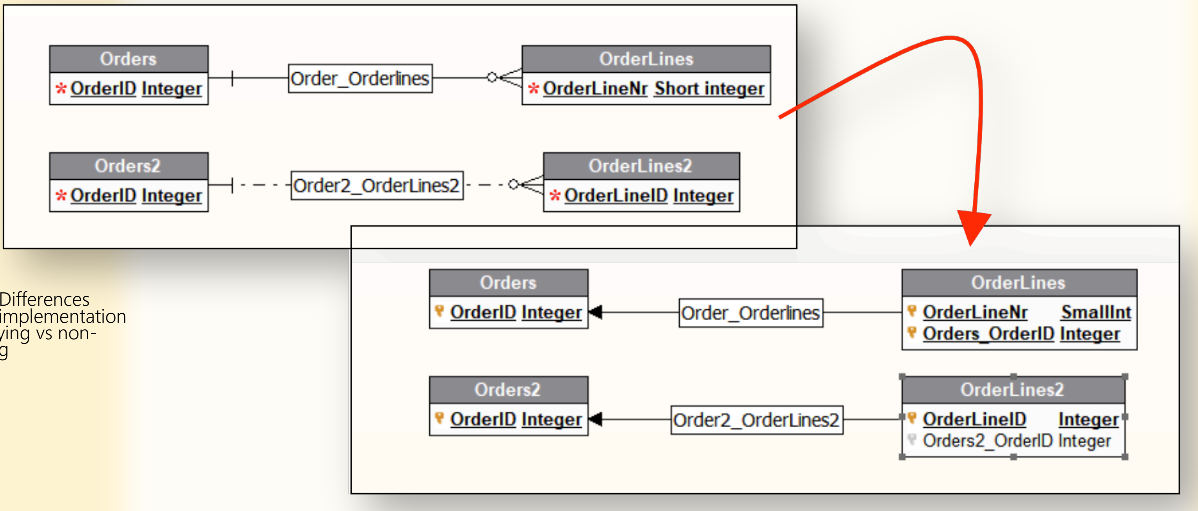
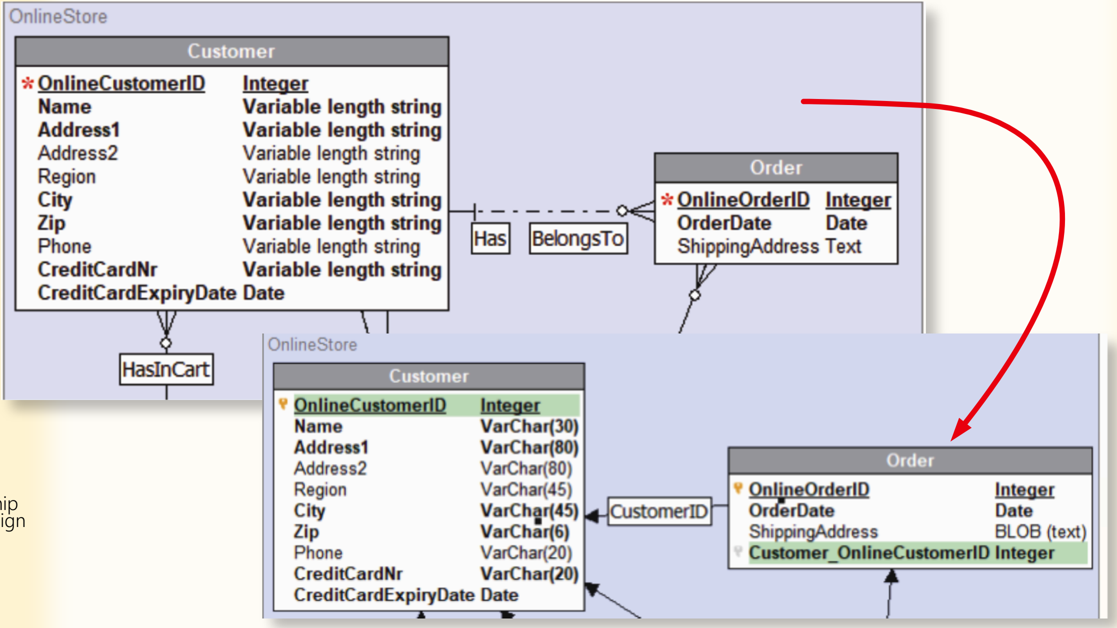


Figure 8: Differences between implementation of identifying vs non-identifying

Additionally, a **non-identifying relationship** can be mandatory or optional, resulting in a child table with either a non-null column for the parent identifier, or a nullable column.
This optionality is also an aspect of cardinality, as it possibly defines "one-or-zero"-to-many relationships, for example. When you generate the physical model, columns for the relationship will be added to the table automatically.



Customer

* OnlineCustomerID	Integer
Name	Variable length string
Address1	Variable length string
Address2	Variable length string
Region	Variable length string
City	Variable length string
Zip	Variable length string
Phone	Variable length string
CreditCardNr	Variable length string
CreditCardExpiryDate	Date

Order

* OnlineOrderID	Integer
OrderDate	Date
ShippingAddress	Text

Customer

OnlineCustomerID	Integer
Name	VarChar(30)
Address1	VarChar(80)
Address2	VarChar(80)
Region	VarChar(45)
City	VarChar(45)
Zip	VarChar(6)
Phone	VarChar(20)
CreditCardNr	VarChar(20)
CreditCardExpiryDate	Date

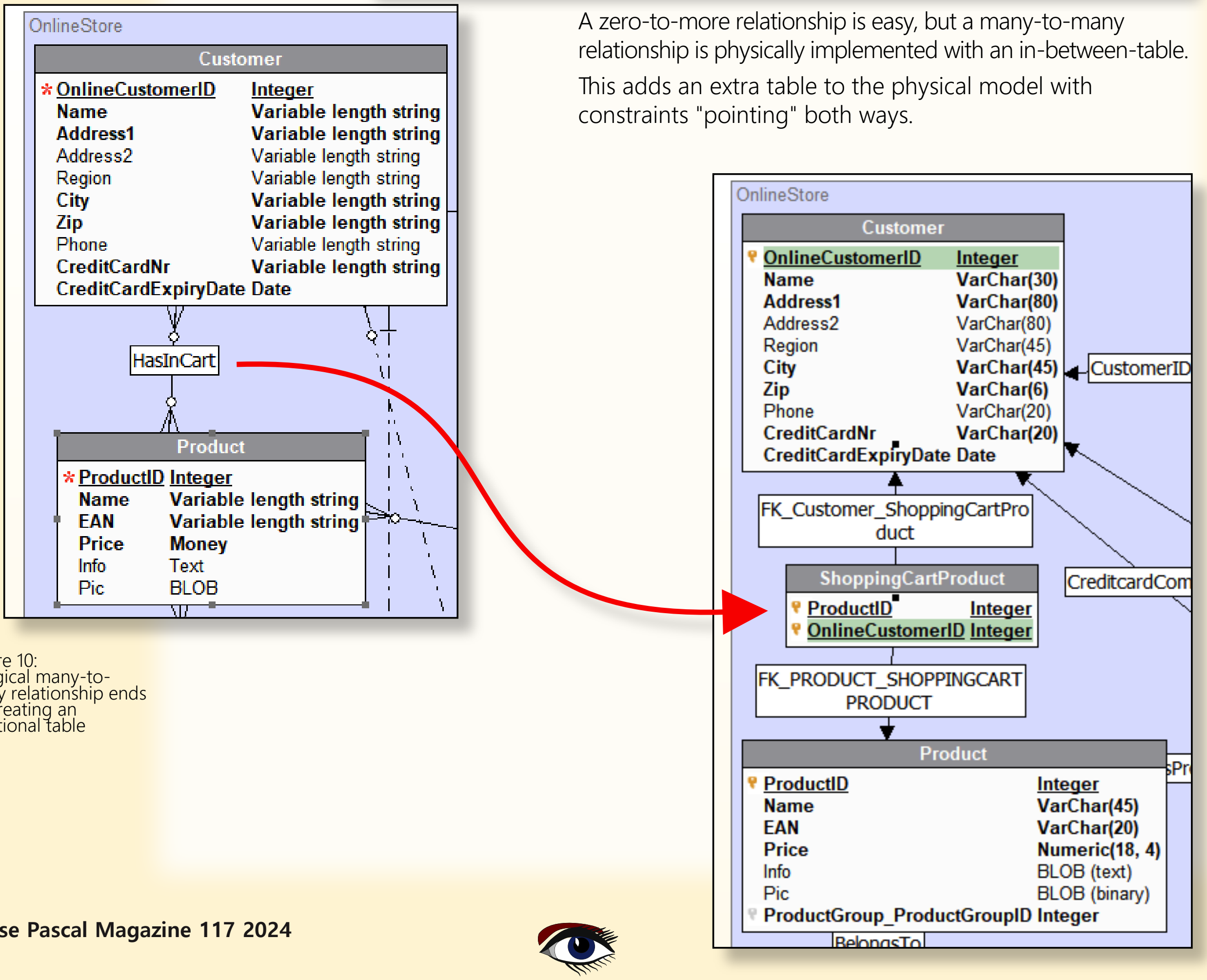
Order

OnlineOrderID	Integer
OrderDate	Date
ShippingAddress	BLOB (text)
Customer_OnlineCustomerID	Integer

Figure 9:
A logical relationship translates to a foreign key constraint

A zero-to-more relationship is easy, but a many-to-many relationship is physically implemented with an in-between-table.

This adds an extra table to the physical model with constraints "pointing" both ways.



Customer

* OnlineCustomerID	Integer
Name	Variable length string
Address1	Variable length string
Address2	Variable length string
Region	Variable length string
City	Variable length string
Zip	Variable length string
Phone	Variable length string
CreditCardNr	Variable length string
CreditCardExpiryDate	Date

Product

* ProductID	Integer
Name	Variable length string
EAN	Variable length string
Price	Money
Info	Text
Pic	BLOB

Customer

OnlineCustomerID	Integer
Name	VarChar(30)
Address1	VarChar(80)
Address2	VarChar(80)
Region	VarChar(45)
City	VarChar(45)
Zip	VarChar(6)
Phone	VarChar(20)
CreditCardNr	VarChar(20)
CreditCardExpiryDate	Date

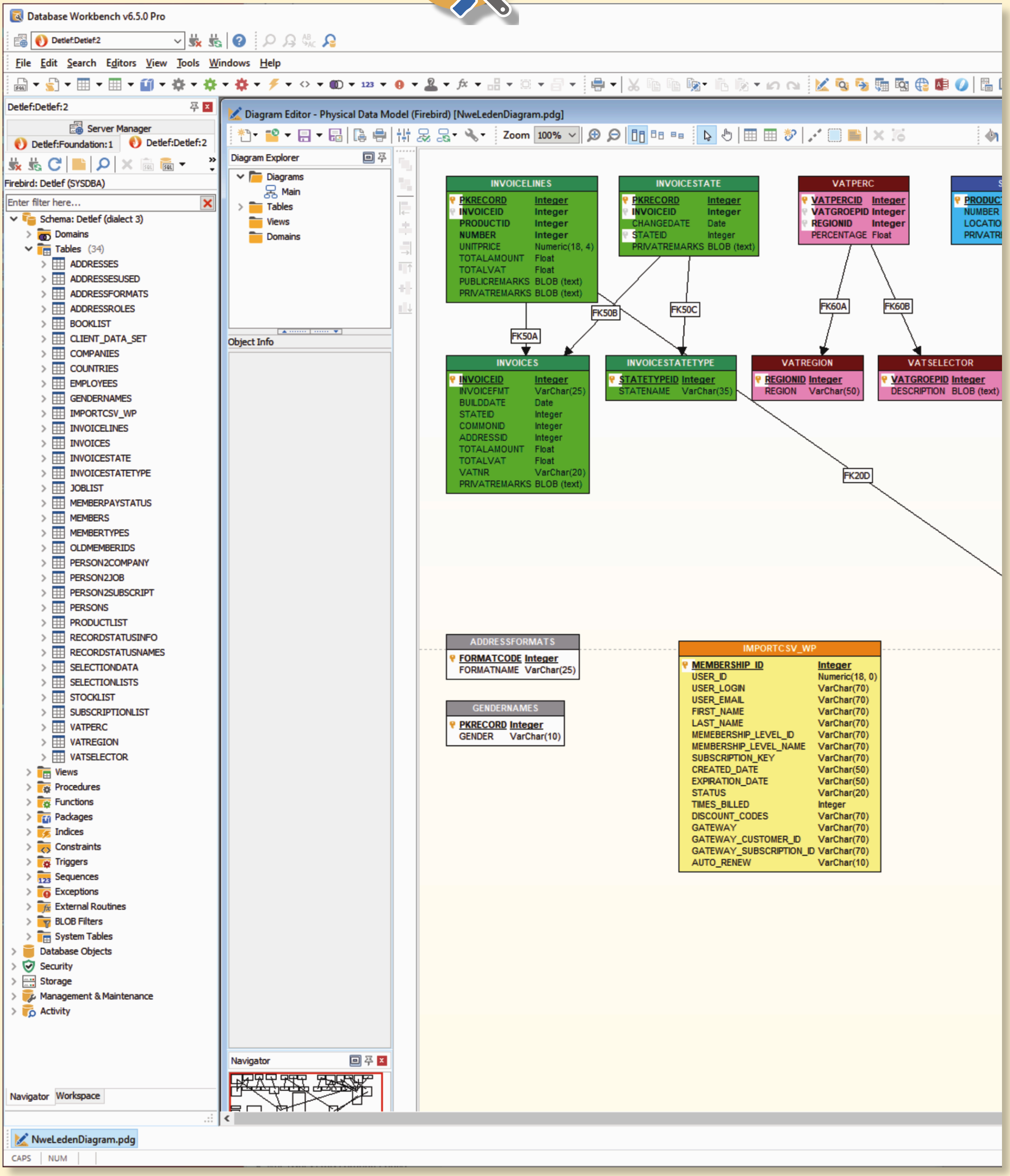
ShoppingCartProduct

ProductID	Integer
OnlineCustomerID	Integer

Product

ProductID	Integer
Name	VarChar(45)
EAN	VarChar(20)
Price	Numeric(18, 4)
Info	BLOB (text)
Pic	BLOB (binary)
ProductGroup_ProductGroupID	Integer

Figure 10:
A logical many-to-many relationship ends up creating an additional table



Database Workbench v6.5.0 Pro

Detlef:Detlef:2

Detlef:Detlef:2

Server Manager

Detlef:Foundation:1 Detlef:Detlef:2

Firebird: Detlef (SYSDBA)

Enter filter here...

- Schema: Detlef (dialect 3)
  - Domains
  - Tables (34)
    - ADDRESSES
    - ADDRESSESUSED
    - ADDRESSFORMATS
    - ADDRESSROLES
    - BOOKLIST
    - CLIENT_DATA_SET
    - COMPANIES
    - COUNTRIES
    - EMPLOYEES
    - GENDERNAMES
    - IMPORTCSV_WP
    - INVOICELINES
    - INVOICES
    - INVOICESTATE
    - INVOICESTATETYPE
    - JOBLIST
    - MEMBERPAYSTATUS
    - MEMBERS
    - MEMBERTYPES
    - OLDMEMBERIDS
    - PERSON2COMPANY
    - PERSON2JOB
    - PERSON2SUBSCRIPT
    - PERSONS
    - PRODUCTLIST
    - RECORDSTATUSINFO
    - RECORDSTATUSNAMES
    - SELECTIONDATA
    - SELECTIONLISTS
    - STOCKLIST
    - SUBSCRIPTIONLIST
    - VATPERC
    - VATREGION
    - VATSELECTOR
  - Views
  - Procedures
  - Functions
  - Packages
  - Indices
  - Constraints
  - Triggers
  - Sequences
  - Exceptions
  - External Routines
  - BLOB Filters
  - System Tables
- Database Objects
- Security
- Storage
- Management & Maintenance
- Activity

Navigator Workspace

Diagram Editor - Physical Data Model (Firebird) [NweLedenDiagram.pdg]

Zoom 100%

Diagram Explorer

- Diagrams
  - Main
- Tables
- Views
- Domains

Object Info

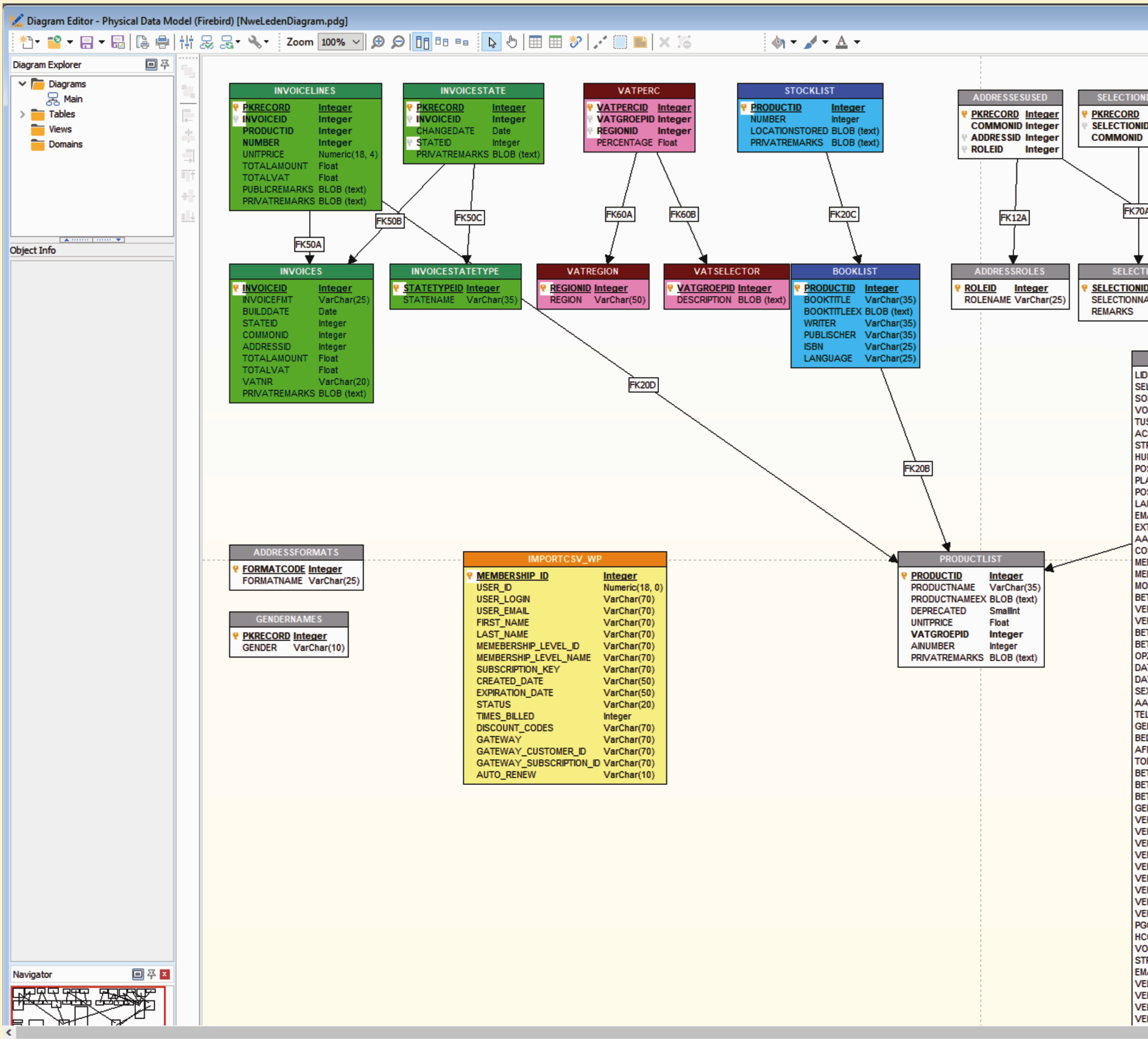
Navigator

```
INVOICELINES
PKRECORD         Integer
INVOICEID        Integer
PRODUCTID        Integer
NUMBER           Integer
UNITPRICE        Numeric(18, 4)
TOTALAMOUNT      Float
TOTALVAT         Float
PUBLICREMARKS    BLOB (text)
PRIVATREMARKS    BLOB (text)

INVOICESTATE
PKRECORD         Integer
INVOICEID        Integer
CHANGEDATE       Date
STATEID          Integer
PRIVATREMARKS    BLOB (text)

VATPERC
VATPERCID        Integer
VATGROEPID       Integer
REGIONID         Integer
PERCENTAGE       Float

PRODUC
NUMBER
LOCATIO
PRIVATR

INVOICES
INVOICEID        Integer
INVOICEFMT       VarChar(25)
BUILDDATE        Date
STATEID          Integer
COMMONID         Integer
ADDRESSID        Integer
TOTALAMOUNT      Float
TOTALVAT         Float
VATNR            VarChar(20)
PRIVATREMARKS    BLOB (text)

INVOICESTATETYPE
STATETYPEID      Integer
STATENAME        VarChar(35)

VATREGION
REGIONID         Integer
REGION           VarChar(50)

VATSELECTOR
VATGROEPID       Integer
DESCRIPTION      BLOB (text)

ADDRESSFORMATS
FORMATCODE       Integer
FORMATNAME       VarChar(25)

GENDERNAMES
PKRECORD         Integer
GENDER           VarChar(10)

IMPORTCSV_WP
MEMBERSHIP_ID                Integer
USER_ID                      Numeric(18, 0)
USER_LOGIN                   VarChar(70)
USER_EMAIL                   VarChar(70)
FIRST_NAME                   VarChar(70)
LAST_NAME                    VarChar(70)
MEMEBERSHIP_LEVEL_ID         VarChar(70)
MEMBERSHIP_LEVEL_NAME        VarChar(70)
SUBSCRIPTION_KEY             VarChar(70)
CREATED_DATE                 VarChar(50)
EXPIRATION_DATE              VarChar(50)
STATUS                       VarChar(20)
TIMES_BILLED                 Integer
DISCOUNT_CODES               VarChar(70)
GATEWAY                      VarChar(70)
GATEWAY_CUSTOMER_ID          VarChar(70)
GATEWAY_SUBSCRIPTION_ID      VarChar(70)
AUTO_RENEW                   VarChar(10)
```

NweLedenDiagram.pdg

CAPS NUM

Figure 11: You can add colors



INVOICELINES	
PKRECORD	Integer
INVOICEID	Integer
PRODUCTID	Integer
NUMBER	Integer
UNITPRICE	Numeric(18, 4)
TOTALAMOUNT	Float
TOTALVAT	Float
PUBLICREMARKS	BLOB (text)
PRIVATREMARKS	BLOB (text)

INVOICESTATE	
PKRECORD	Integer
INVOICEID	Integer
CHANGEDATE	Date
STATEID	Integer
PRIVATREMARKS	BLOB (text)

VATPERC	
VATPERCID	Integer
VATGROEPID	Integer
REGIONID	Integer
PERCENTAGE	Float

STOCKLIST	
PRODUCTID	Integer
NUMBER	Integer
LOCATIONSTORED	BLOB (text)
PRIVATREMARKS	BLOB (text)

ADDRESSESUSED	
PKRECORD	Integer
COMMONID	Integer
ADDRESSID	Integer
ROLEID	Integer

SELECTIONI	
PKRECORD	
SELECTIONID	
COMMONID	

INVOICES	
INVOICEID	Integer
INVOICEFMT	VarChar(25)
BUILDDATE	Date
STATEID	Integer
COMMONID	Integer
ADDRESSID	Integer
TOTALAMOUNT	Float
TOTALVAT	Float
VATNR	VarChar(20)
PRIVATREMARKS	BLOB (text)

INVOICESTATETYPE	
STATETYPEID	Integer
STATENAME	VarChar(35)

VATREGION	
REGIONID	Integer
REGION	VarChar(50)

VATSELECTOR	
VATGROEPID	Integer
DESCRIPTION	BLOB (text)

BOOKLIST	
PRODUCTID	Integer
BOOKTITLE	VarChar(35)
BOOKTITLEEX	BLOB (text)
WRITER	VarChar(35)
PUBLISCHER	VarChar(35)
ISBN	VarChar(25)
LANGUAGE	VarChar(25)

ADDRESSROLES	
ROLEID	Integer
ROLENAME	VarChar(25)

SELECTI	
SELECTIONID	
SELECTIONNA	
REMARKS	

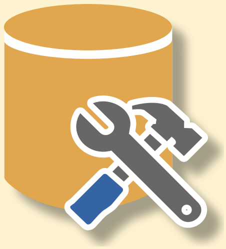
ADDRESSFORMATS	
FORMATCODE	Integer
FORMATNAME	VarChar(25)

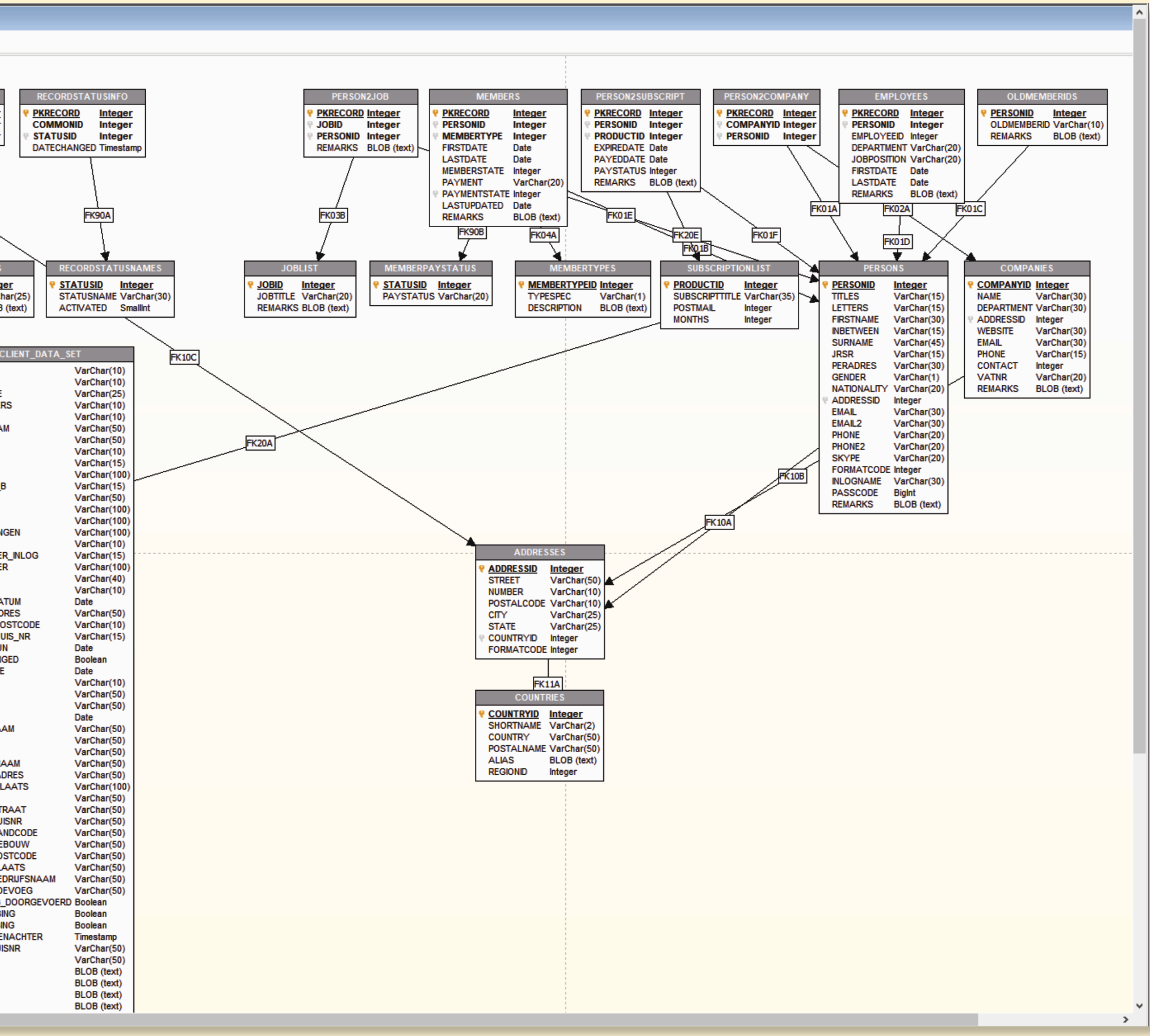
GENDERNAMES	
PKRECORD	Integer
GENDER	VarChar(10)

IMPORTCSV_WP	
MEMBERSHIP_ID	Integer
USER_ID	Numeric(18, 0)
USER_LOGIN	VarChar(70)
USER_EMAIL	VarChar(70)
FIRST_NAME	VarChar(70)
LAST_NAME	VarChar(70)
MEMEBERSHIP_LEVEL_ID	VarChar(70)
MEMBERSHIP_LEVEL_NAME	VarChar(70)
SUBSCRIPTION_KEY	VarChar(70)
CREATED_DATE	VarChar(50)
EXPIRATION_DATE	VarChar(50)
STATUS	VarChar(20)
TIMES_BILLED	Integer
DISCOUNT_CODES	VarChar(70)
GATEWAY	VarChar(70)
GATEWAY_CUSTOMER_ID	VarChar(70)
GATEWAY_SUBSCRIPTION_ID	VarChar(70)
AUTO_RENEW	VarChar(10)

PRODUCTLIST	
PRODUCTID	Integer
PRODUCTNAME	VarChar(35)
PRODUCTNAMEEX	BLOB (text)
DEPRECATED	SmallInt
UNITPRICE	Float
VATGROEPID	Integer
AINUMBER	Integer
PRIVATREMARKS	BLOB (text)

Figure: 12 If you open the pdf file with an opposite page you will see the whole picture over two pages





RECORDSTATUSINFO	
PKRECORD	**Integer**
COMMONID	**Integer**
STATUSID	**Integer**
DATECHANGED	Timestamp

PERSON2JOB	
PKRECORD	**Integer**
JOBID	**Integer**
PERSONID	**Integer**
REMARKS	BLOB (text)

MEMBERS	
PKRECORD	**Integer**
PERSONID	**Integer**
MEMBERTYPE	**Integer**
FIRSTDATE	Date
LASTDATE	Date
MEMBERSTATE	Integer
PAYMENT	VarChar(20)
PAYMENTSTATE	Integer
LASTUPDATED	Date
REMARKS	BLOB (text)

PERSON2SUBSCRIPT	
PKRECORD	**Integer**
PERSONID	**Integer**
PRODUCTID	**Integer**
EXPIREDATE	Date
PAYEDDATE	Date
PAYSTATUS	Integer
REMARKS	BLOB (text)

PERSON2COMPANY	
PKRECORD	**Integer**
COMPANYID	**Integer**
PERSONID	**Integer**

EMPLOYEES	
PKRECORD	**Integer**
PERSONID	**Integer**
EMPLOYEEID	Integer
DEPARTMENT	VarChar(20)
JOBPOSITION	VarChar(20)
FIRSTDATE	Date
LASTDATE	Date
REMARKS	BLOB (text)

OLDMEMBERIDS	
PERSONID	**Integer**
OLDMEMBERID	VarChar(10)
REMARKS	BLOB (text)

ger
har(25)
B (text)

RECORDSTATUSNAMES	
STATUSID	**Integer**
STATUSNAME	VarChar(30)
ACTIVATED	SmallInt

JOBLIST	
JOBID	**Integer**
JOBTITLE	VarChar(20)
REMARKS	BLOB (text)

MEMBERPAYSTATUS	
STATUSID	**Integer**
PAYSTATUS	VarChar(20)

MEMBERTYPES	
MEMBERTYPEID	**Integer**
TYPESPEC	VarChar(1)
DESCRIPTION	BLOB (text)

SUBSCRIPTIONLIST	
PRODUCTID	**Integer**
SUBSCRIPTTITLE	VarChar(35)
POSTMAIL	Integer
MONTHS	Integer

PERSONS	
PERSONID	**Integer**
TITLES	VarChar(15)
LETTERS	VarChar(15)
FIRSTNAME	VarChar(30)
INBETWEEN	VarChar(15)
SURNAME	VarChar(45)
JRSR	VarChar(15)
PERADRES	VarChar(30)
GENDER	VarChar(1)
NATIONALITY	VarChar(20)
ADDRESSID	Integer
EMAIL	VarChar(30)
EMAIL2	VarChar(30)
PHONE	VarChar(20)
PHONE2	VarChar(20)
SKYPE	VarChar(20)
FORMATCODE	Integer
INLOGNAME	VarChar(30)
PASSCODE	BigInt
REMARKS	BLOB (text)

COMPANIES	
COMPANYID	**Integer**
NAME	VarChar(30)
DEPARTMENT	VarChar(30)
ADDRESSID	Integer
WEBSITE	VarChar(30)
EMAIL	VarChar(30)
PHONE	VarChar(15)
CONTACT	Integer
VATNR	VarChar(20)
REMARKS	BLOB (text)

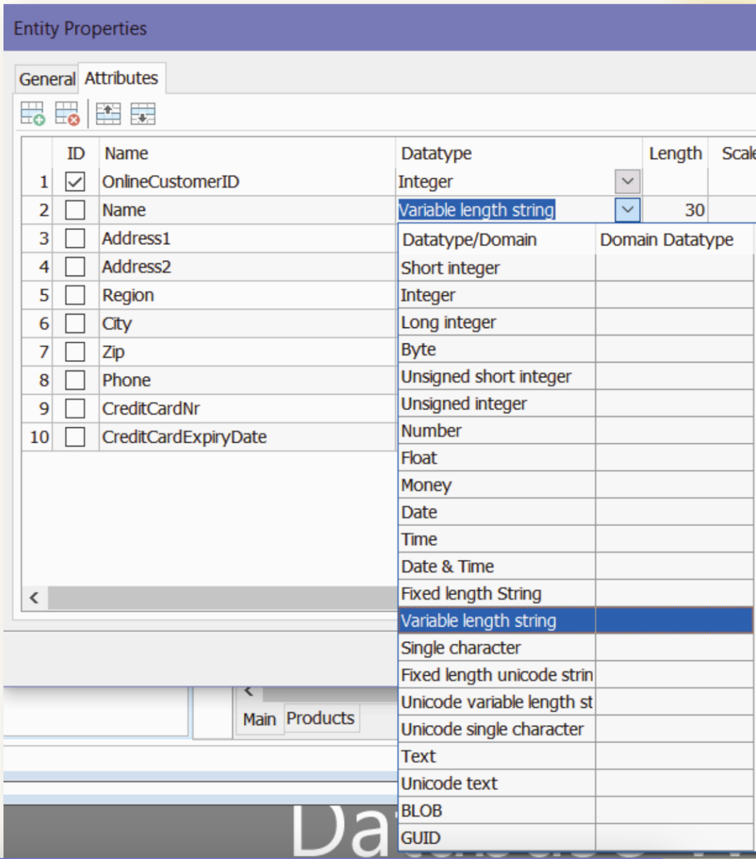
CLIENT_DATA_SET	
	VarChar(10)
	VarChar(10)
E	VarChar(25)
RS	VarChar(10)
	VarChar(10)
AM	VarChar(50)
	VarChar(50)
	VarChar(10)
	VarChar(15)
	VarChar(100)
B	VarChar(15)
	VarChar(50)
	VarChar(100)
	VarChar(100)
NGEN	VarChar(100)
	VarChar(10)
ER_INLOG	VarChar(15)
ER	VarChar(100)
	VarChar(40)
	VarChar(10)
ATUM	Date
DRES	VarChar(50)
OSTCODE	VarChar(10)
UIS_NR	VarChar(15)
UN	Date
IGED	Boolean
E	Date
	VarChar(10)
	VarChar(50)
	VarChar(50)
	Date
AAM	VarChar(50)
	VarChar(50)
	VarChar(50)
AAM	VarChar(50)
ADRES	VarChar(50)
LAATS	VarChar(100)
	VarChar(50)
TRAAT	VarChar(50)
UISNR	VarChar(50)
ANDCODE	VarChar(50)
EBOUW	VarChar(50)
OSTCODE	VarChar(50)
LAATS	VarChar(50)
EDRIJFSNAAM	VarChar(50)
DEVOEG	VarChar(50)
_DOORGEVOERD	Boolean
SING	Boolean
SING	Boolean
ENACHTER	Timestamp
UISNR	VarChar(50)
	VarChar(50)
	BLOB (text)
	BLOB (text)
	BLOB (text)
	BLOB (text)

ADDRESSES	
ADDRESSID	**Integer**
STREET	VarChar(50)
NUMBER	VarChar(10)
POSTALCODE	VarChar(10)
CITY	VarChar(25)
STATE	VarChar(25)
COUNTRYID	Integer
FORMATCODE	Integer

COUNTRIES	
COUNTRYID	**Integer**
SHORTNAME	VarChar(2)
COUNTRY	VarChar(50)
POSTALNAME	VarChar(50)
ALIAS	BLOB (text)
REGIONID	Integer

Figure 13:
The logical data types get converted to physical, database system dependent types

Right:
Focusing on modeling the business logic first, is easier than trying to implement a database right away and gives you more visual information about the relationships compared to visualized foreign key constraints. See the next page for the other half op the picture.



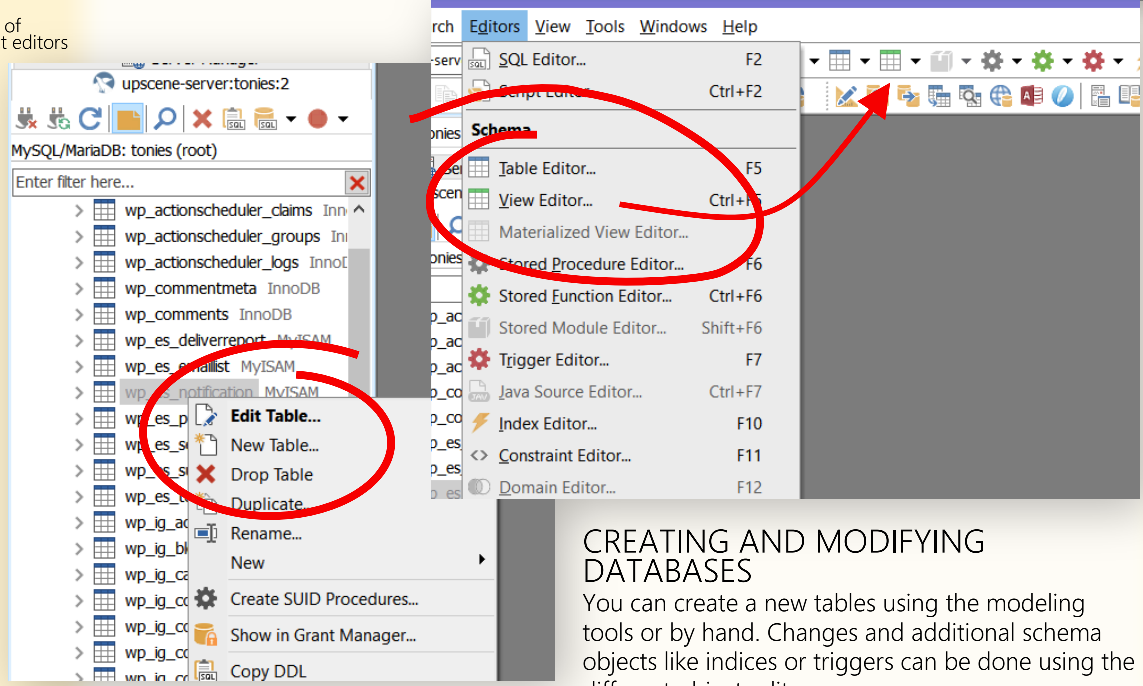
Entity Properties

General | Attributes

	ID	Name	Datatype	Length	Scale
1	☑	OnlineCustomerID	Integer		
2	☐	Name	Variable length string	30	
3	☐	Address1			
4	☐	Address2			
5	☐	Region			
6	☐	City			
7	☐	Zip			
8	☐	Phone			
9	☐	CreditCardNr			
10	☐	CreditCardExpiryDate			

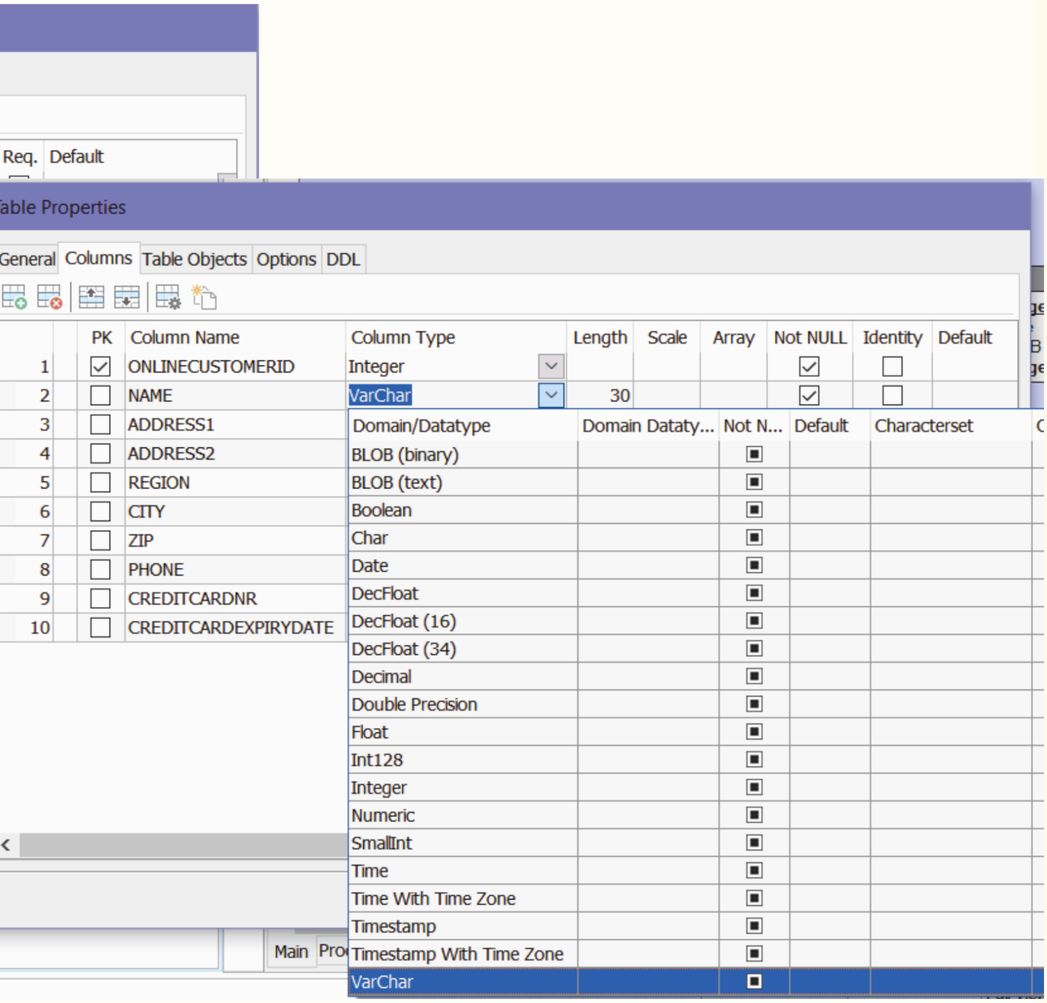
Datatype/Domain	Domain Datatype
Short integer	
Integer	
Long integer	
Byte	
Unsigned short integer	
Unsigned integer	
Number	
Float	
Money	
Date	
Time	
Date & Time	
Fixed length String	
Variable length string	
Single character	
Fixed length unicode strin	
Unicode variable length st	
Unicode single character	
Text	
Unicode text	
BLOB	
GUID	

Figure 14:
Different ways of opening object editors



CREATING AND MODIFYING DATABASES

You can create a new tables using the modeling tools or by hand. Changes and additional schema objects like indices or triggers can be done using the different object editors.
Object editors can be reached from the Database Navigator using double click or the context menu, via the buttons in the toolbar or the main menu.

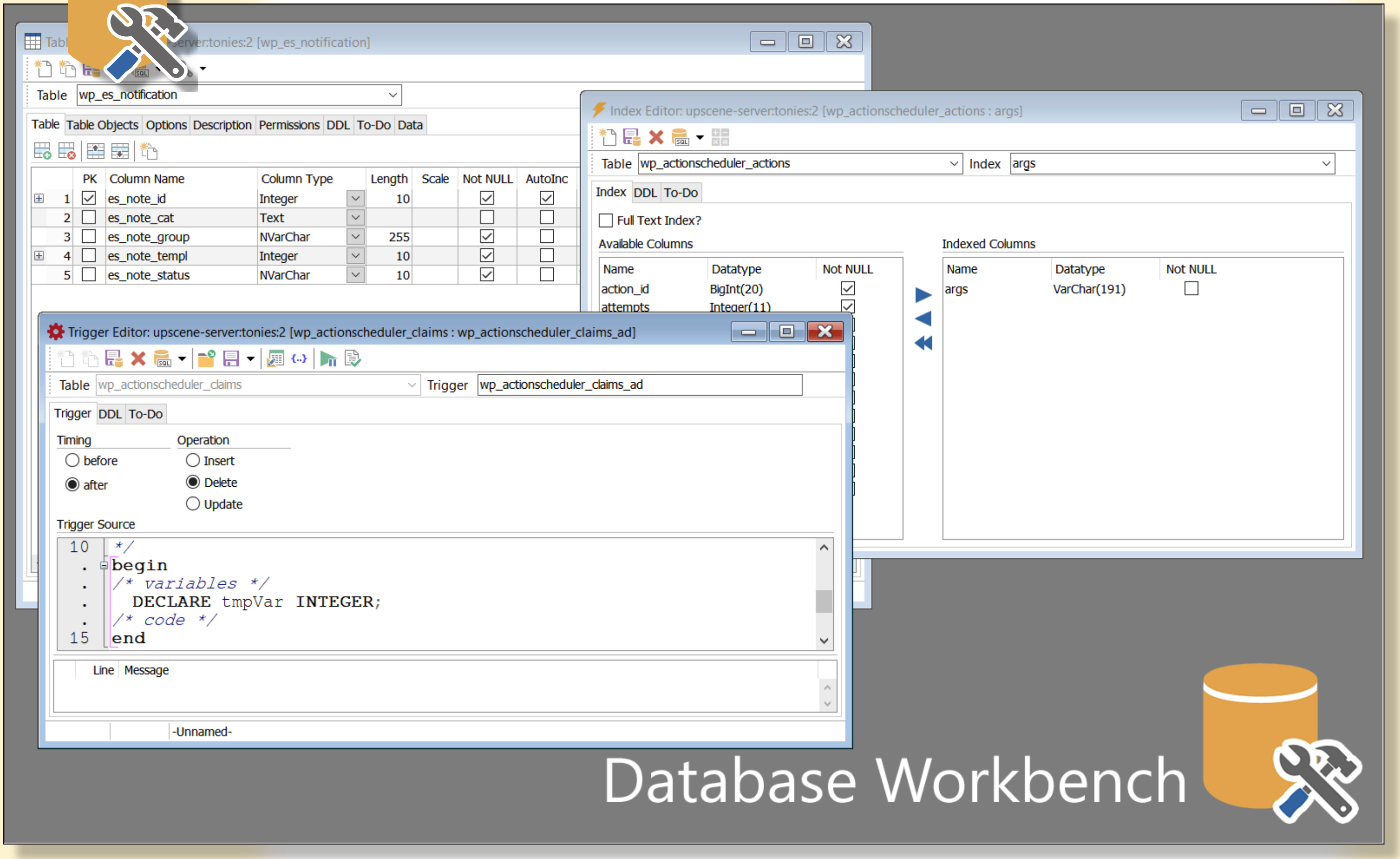


	PK	Column Name	Column Type	Length	Scale	Array	Not NULL	Identity	Default
1	☑	ONLINECUSTOMERID	Integer				☑	☐	
2	☐	NAME	VarChar	30			☑	☐	
3	☐	ADDRESS1							
4	☐	ADDRESS2							
5	☐	REGION							
6	☐	CITY							
7	☐	ZIP							
8	☐	PHONE							
9	☐	CREDITCARDNR							
10	☐	CREDITCARDEXPIRYDATE							

Domain/Datatype	Domain Dataty...	Not N...	Default	Characterset	C
BLOB (binary)		■			
BLOB (text)		■			
Boolean		■			
Char		■			
Date		■			
DecFloat		■			
DecFloat (16)		■			
DecFloat (34)		■			
Decimal		■			
Double Precision		■			
Float		■			
Int128		■			
Integer		■			
Numeric		■			
SmallInt		■			
Time		■			
Time With Time Zone		■			
Timestamp		■			
Timestamp With Time Zone		■			
VarChar		■			

Figure: 12 If you open the pdf file with an opposite page you will see the whole picture over two pages

Each type of object has a different editor available. And although different database systems have different features for each object type, the user interface shows you consistent object editors for each system.



Tabl… server:tonies:2 [wp_es_notification]

Table: wp_es_notification

	PK	Column Name	Column Type	Length	Scale	Not NULL	AutoInc
1	☑	es_note_id	Integer	10		☑	☑
2	□	es_note_cat	Text			□	□
3	□	es_note_group	NVarChar	255		☑	□
4	□	es_note_templ	Integer	10		☑	□
5	□	es_note_status	NVarChar	10		☑	□

Index Editor: upscene-server:tonies:2 [wp_actionscheduler_actions : args]

Table: wp_actionscheduler_actions Index: args

□ Full Text Index?

Available Columns

Name	Datatype	Not NULL
action_id	BigInt(20)	☑
attempts	Integer(11)	☑

Indexed Columns

Name	Datatype	Not NULL
args	VarChar(191)	□

Trigger Editor: upscene-server:tonies:2 [wp_actionscheduler_claims : wp_actionscheduler_claims_ad]

Table: wp_actionscheduler_claims Trigger: wp_actionscheduler_claims_ad

Timing: ○ before ● after

Operation: ○ Insert ● Delete ○ Update

Trigger Source

```
*/
begin
/* variables */
  DECLARE tmpVar INTEGER;
/* code */
end
```

Line Message

-Unnamed-

Figure 15: Several types of object editors

From each object editor, you can create new objects or modify existing objects. Depending on the type of object, additional tabs can be available that show, for example, the data for tables.

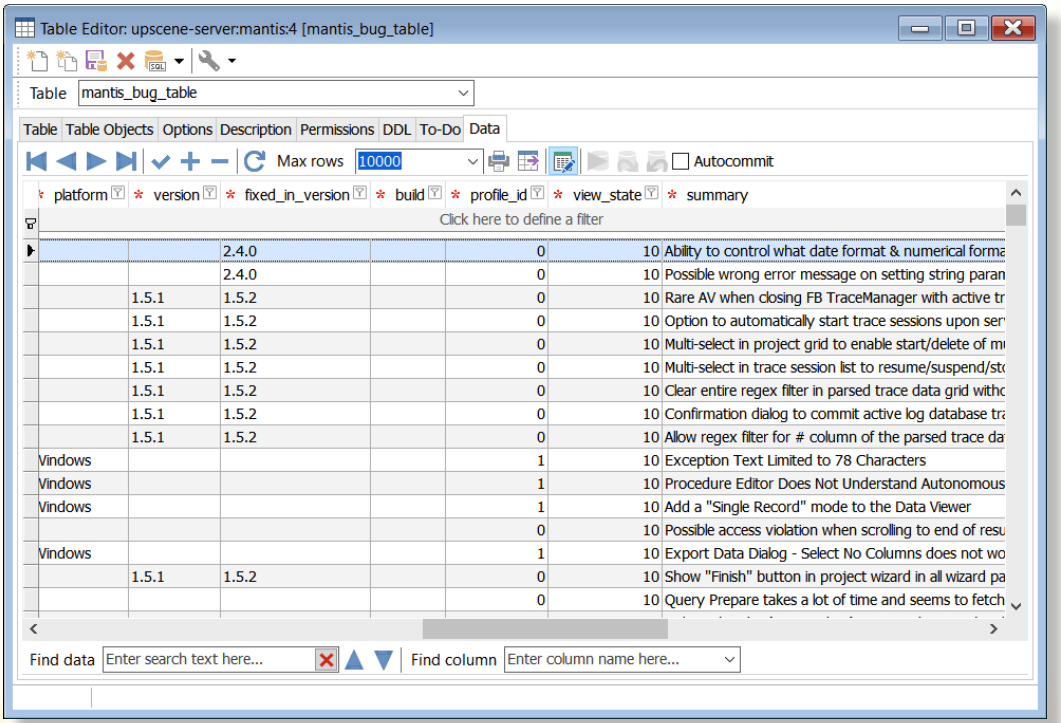


Table Editor: upscene-server:mantis:4 [mantis_bug_table]

Table: mantis_bug_table

Max rows 10000 □ Autocommit

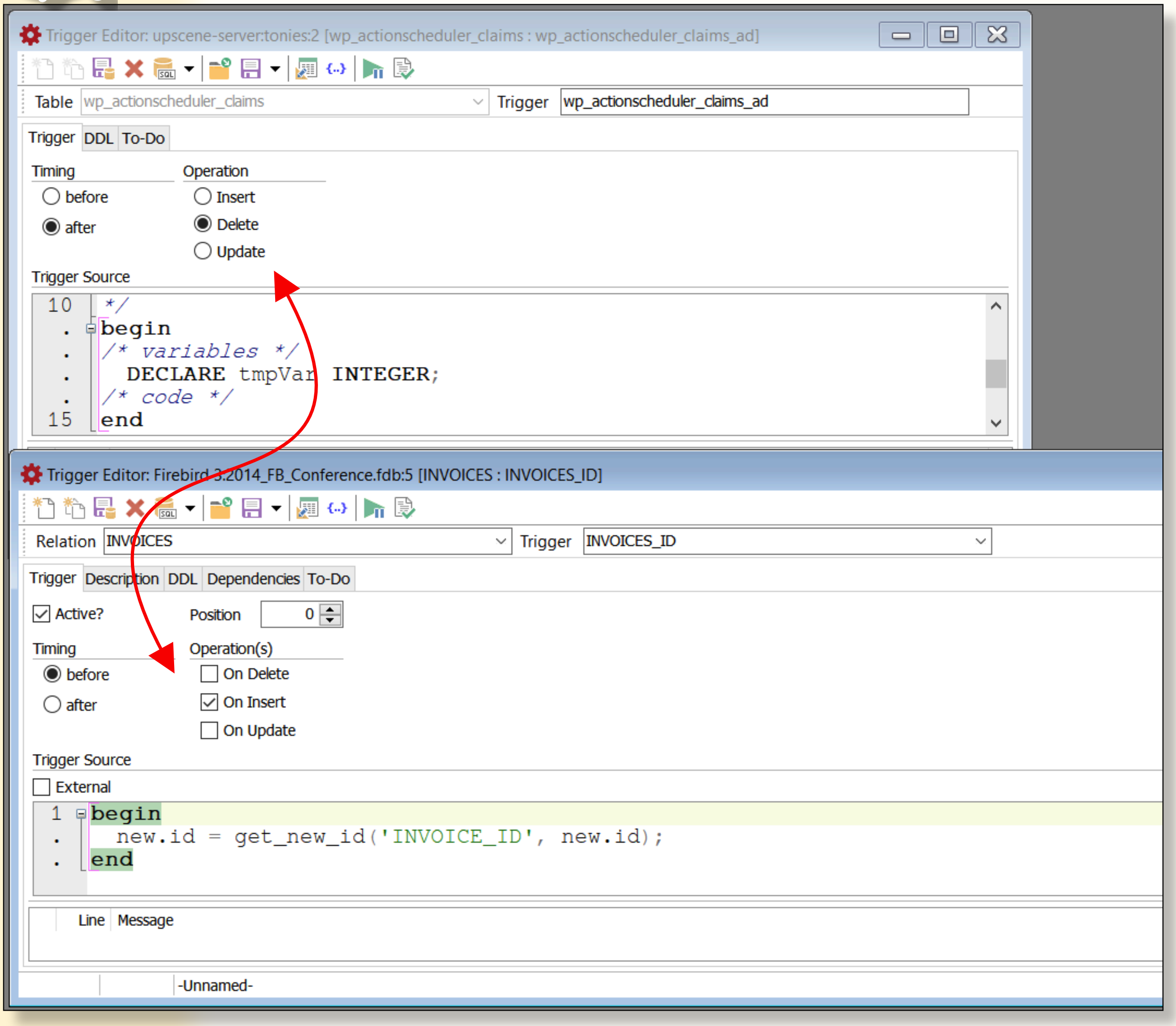
Click here to define a filter

platform	version	fixed_in_version	build	profile_id	view_state	summary
		2.4.0		0	10	Ability to control what date format & numerical forma
		2.4.0		0	10	Possible wrong error message on setting string paran
	1.5.1	1.5.2		0	10	Rare AV when closing FB TraceManager with active tr
	1.5.1	1.5.2		0	10	Option to automatically start trace sessions upon ser
	1.5.1	1.5.2		0	10	Multi-select in project grid to enable start/delete of mu
	1.5.1	1.5.2		0	10	Multi-select in trace session list to resume/suspend/sto
	1.5.1	1.5.2		0	10	Clear entire regex filter in parsed trace data grid witho
	1.5.1	1.5.2		0	10	Confirmation dialog to commit active log database tra
	1.5.1	1.5.2		0	10	Allow regex filter for # column of the parsed trace da
Vindows				1	10	Exception Text Limited to 78 Characters
Vindows				1	10	Procedure Editor Does Not Understand Autonomous
Vindows				1	10	Add a "Single Record" mode to the Data Viewer
				0	10	Possible access violation when scrolling to end of resu
Vindows				1	10	Export Data Dialog - Select No Columns does not wo
	1.5.1	1.5.2		0	10	Show "Finish" button in project wizard in all wizard pa
				0	10	Query Prepare takes a lot of time and seems to fetch

Find data: Enter search text here... Find column: Enter column name here...

Figure 16: Table Editor with additional Data tab

Different database systems all use a slightly different syntax for these statements, from within the editors, you can easily modify the object properties and **Database Workbench** will generate system specific **Data Definition Language (DDL)** statements for you.



Trigger Editor: upscene-server:tonies:2 [wp_actionscheduler_claims : wp_actionscheduler_claims_ad]

Table: wp_actionscheduler_claims — Trigger: wp_actionscheduler_claims_ad

Tabs: Trigger | DDL | To-Do

Timing: ○ before ● after

Operation: ○ Insert ● Delete ○ Update

Trigger Source

```
*/
begin
/* variables */
  DECLARE tmpVar INTEGER;
/* code */
end
```

Trigger Editor: Firebird 3.2014_FB_Conference.fdb:5 [INVOICES : INVOICES_ID]

Relation: INVOICES — Trigger: INVOICES_ID

Tabs: Trigger | Description | DDL | Dependencies | To-Do

☑ Active? Position: 0

Timing: ● before ○ after

Operation(s): ☐ On Delete ☑ On Insert ☐ On Update

Trigger Source

☐ External

```
begin
  new.id = get_new_id('INVOICE_ID', new.id);
end
```

Line | Message

-Unnamed-

Figure 17: Trigger Editor for MySQL and Firebird, similar interface, small differences

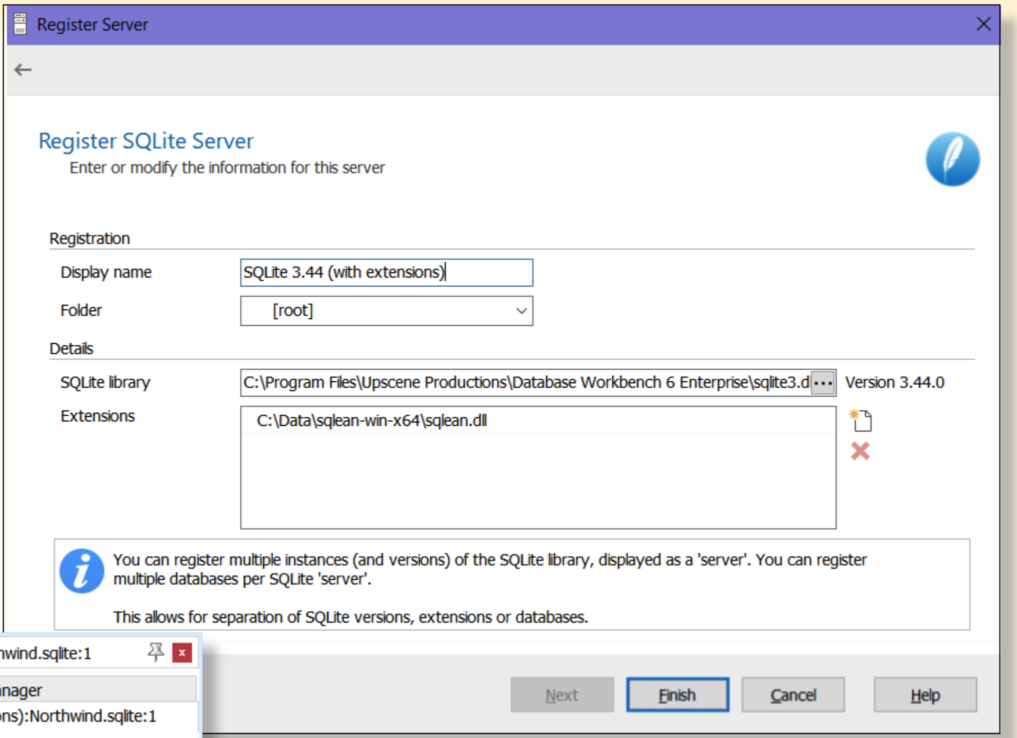
The user interface shows you the different possible data types, options for indices, triggers and constraints. It allows you to create complex objects using mouse and keyboard, all without having to know the exact DDL syntax or available options. It's fast and easy.

SQLITE WITH DATABASE WORKBENCH

The **SQLite module** was recently added to Database Workbench, it fully supports all SQLite features include third party extensions.

Although **SQLite** is not server-based, it fits neatly into the application, instead of registering a server, you register the **SQLite library and optional extensions**. After that, you can register your databases. You can register multiple versions of the library or with different sets of extensions depending on your requirements.

Figure 18:
Registering an SQLite library with an extension



Register Server

Register SQLite Server

Enter or modify the information for this server

Registration

Display name: SQLite 3.44 (with extensions)

Folder: [root]

Details

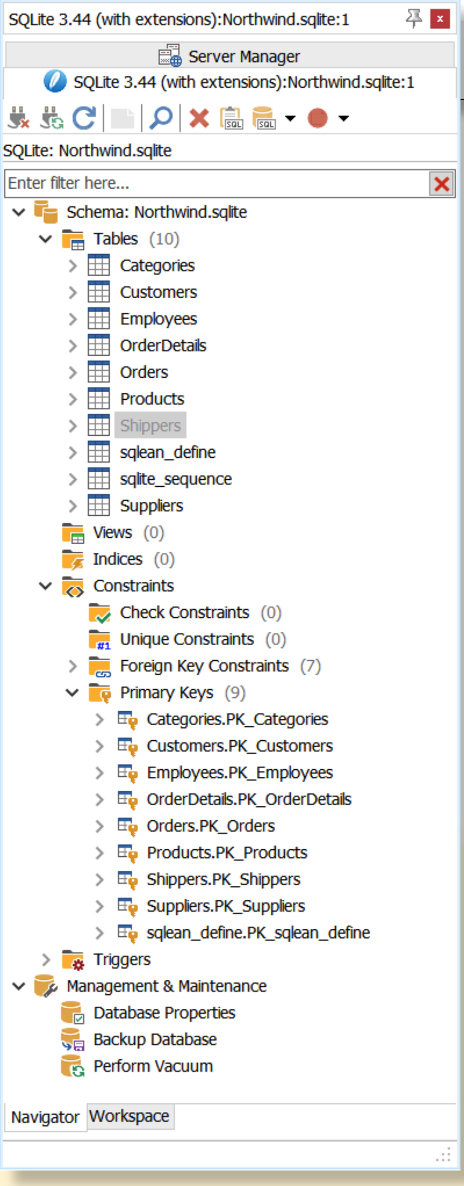
SQLite library: C:\Program Files\Upscene Productions\Database Workbench 6 Enterprise\sqlite3.d Version 3.44.0

Extensions: C:\Data\sqlean-win-x64\sqlean.dll

You can register multiple instances (and versions) of the SQLite library, displayed as a 'server'. You can register multiple databases per SQLite 'server'.

This allows for separation of SQLite versions, extensions or databases.

Figure 19:
The "Northwind" SQLite database



SQLite 3.44 (with extensions):Northwind.sqlite:1

Server Manager

SQLite 3.44 (with extensions):Northwind.sqlite:1

SQLite: Northwind.sqlite

Enter filter here...

- Schema: Northwind.sqlite
  - Tables (10)
    - Categories
    - Customers
    - Employees
    - OrderDetails
    - Orders
    - Products
    - Shippers
    - sqlean_define
    - sqlite_sequence
    - Suppliers
  - Views (0)
  - Indices (0)
  - Constraints
    - Check Constraints (0)
    - Unique Constraints (0)
    - Foreign Key Constraints (7)
    - Primary Keys (9)
      - Categories.PK_Categories
      - Customers.PK_Customers
      - Employees.PK_Employees
      - OrderDetails.PK_OrderDetails
      - Orders.PK_Orders
      - Products.PK_Products
      - Shippers.PK_Shippers
      - Suppliers.PK_Suppliers
      - sqlean_define.PK_sqlean_define
  - Triggers
- Management & Maintenance
  - Database Properties
  - Backup Database
  - Perform Vacuum

Registering or creating a database is as easy as selecting an existing file or entering a new filename. You can now open the database and view or create the meta data and data.

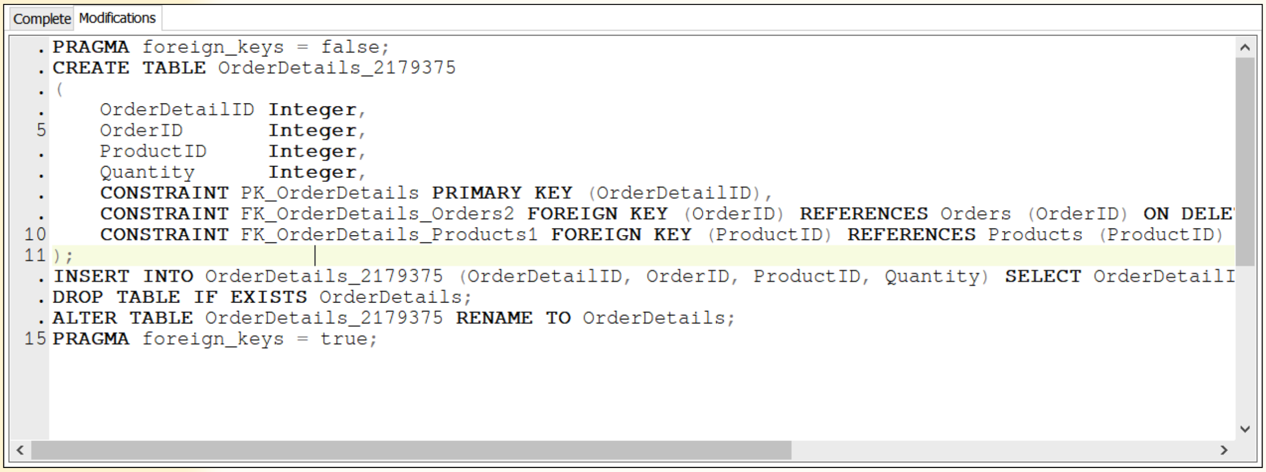
The Database **Navigator** displays the tree with schema objects and several **"Management & Maintenance"** options for commonly used database information and tasks.
Although **SQLite** is easy to use, you'll notice that during development you can run into problems.

For example, **SQLite does not support adding or removing constraints for existing tables**. There's a number of steps to take:

❶ Create a new table with the constraints
❷ Transfer the data from the old table to the new
❸ Drop views that rely on the old table
❹ Rename the new table
❺ Recreate the triggers on the table
❻ Recreate the views

Database Workbench has this all automated for you.

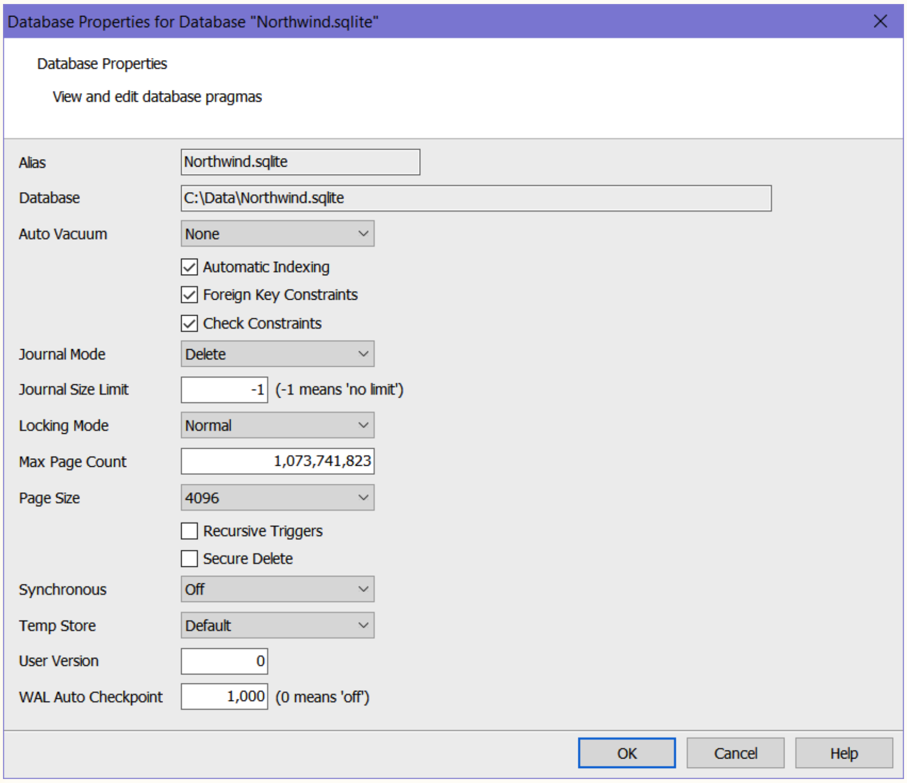
In the **Table Editor** you can add or drop constraints
and the application will generate the required **SQL** statements for you.
This makes modifying your SQLite database much easier.
On the next page of this article (12 /33) is an example,
no views or triggers available.



```
PRAGMA foreign_keys = false;
CREATE TABLE OrderDetails_2179375
(
    OrderDetailID Integer,
    OrderID       Integer,
    ProductID     Integer,
    Quantity      Integer,
    CONSTRAINT PK_OrderDetails PRIMARY KEY (OrderDetailID),
    CONSTRAINT FK_OrderDetails_Orders2 FOREIGN KEY (OrderID) REFERENCES Orders (OrderID) ON DELE
    CONSTRAINT FK_OrderDetails_Products1 FOREIGN KEY (ProductID) REFERENCES Products (ProductID)
);
INSERT INTO OrderDetails_2179375 (OrderDetailID, OrderID, ProductID, Quantity) SELECT OrderDetailI
DROP TABLE IF EXISTS OrderDetails;
ALTER TABLE OrderDetails_2179375 RENAME TO OrderDetails;
PRAGMA foreign_keys = true;
```

Figure 20:
The statements after modifying a foreign key constraint for a table

SQLite database properties, or so-called **PRAGMAS**, can be easily modified in **Database Workbench** as well: it displays the available options, just click and select and use OK to finish the job.



Database Properties

View and edit database pragmas

Alias: Northwind.sqlite

Database: C:\Data\Northwind.sqlite

Auto Vacuum: None

☑ Automatic Indexing

☑ Foreign Key Constraints

☑ Check Constraints

Journal Mode: Delete

Journal Size Limit: -1 (-1 means 'no limit')

Locking Mode: Normal

Max Page Count: 1,073,741,823

Page Size: 4096

☐ Recursive Triggers

☐ Secure Delete

Synchronous: Off

Temp Store: Default

User Version: 0

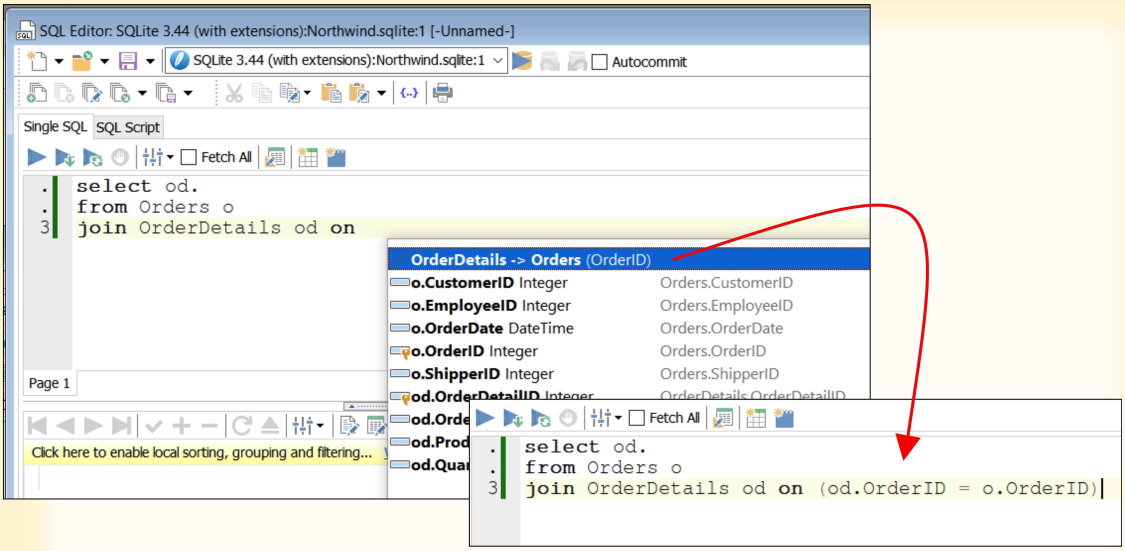
WAL Auto Checkpoint: 1,000 (0 means 'off')

Figure 21:
Editing database properties for SQLite

TOOLS FOR WRITING QUERIES

Writing **SQL queries** is part of day-to-day database development. **Database Workbench** offers several tools to make this easier for you.

The **SQL Insight tool** can help you when writing queries by hand. As is the case for other text editors for programming, this tools parses what you've written so far and offers suggestions in a drop down box. **This helps you to quickly select tables, columns and write JOINs.**



SQL Editor: SQLite 3.44 (with extensions):Northwind.sqlite:1 [-Unnamed-]

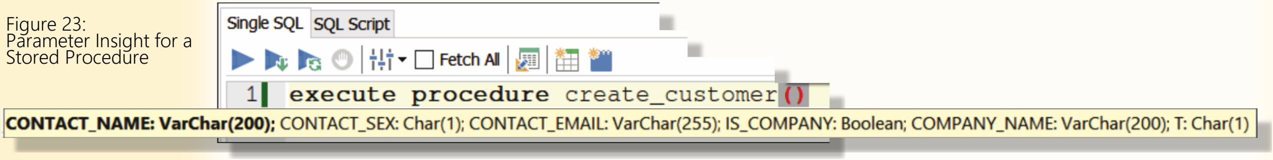
```
select od.
from Orders o
join OrderDetails od on
```

OrderDetails -> Orders (OrderID)	
o.CustomerID Integer	Orders.CustomerID
o.EmployeeID Integer	Orders.EmployeeID
o.OrderDate DateTime	Orders.OrderDate
o.OrderID Integer	Orders.OrderID
o.ShipperID Integer	Orders.ShipperID
od.OrderDetailID Integer	OrderDetails.OrderDetailID
od.Orde	
od.Prod	
od.Quan	

```
select od.
from Orders o
join OrderDetails od on (od.OrderID = o.OrderID)
```

Figure 22: SQL Insight with Join Completion

SQL Insight understands table aliases when selecting columns, parses definition of temporary views and uses **foreign key constraints** available in the database for **JOIN-completion**. Additionally, **Parameter Insight** shows the parameter names and data types when trying to execute stored procedures from SQL or when calling built-in **DBMS functions**.

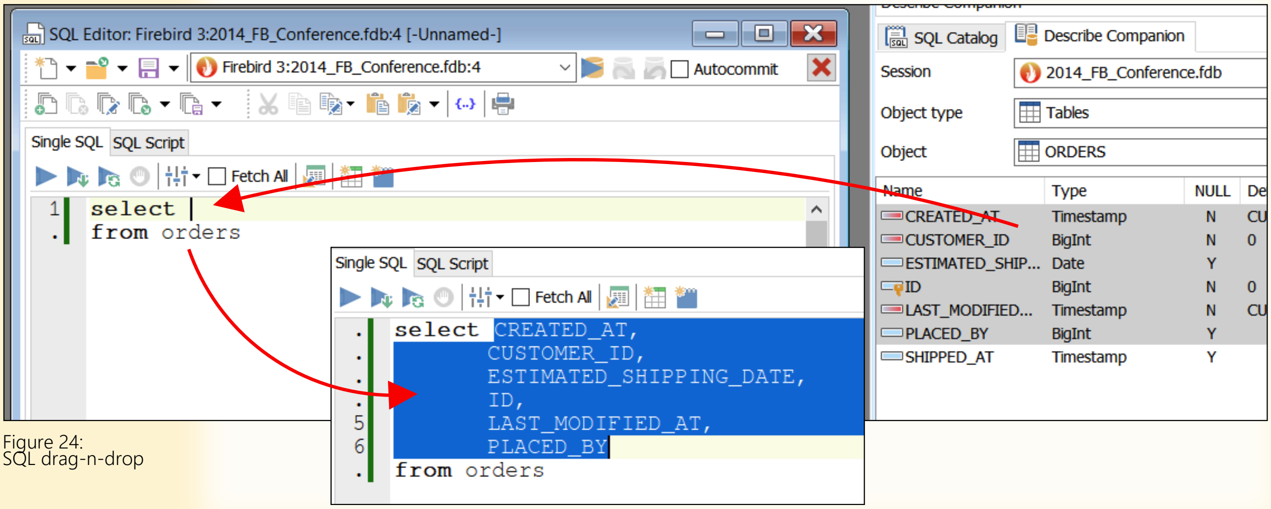


```
execute procedure create_customer()
```

CONTACT_NAME: VarChar(200); CONTACT_SEX: Char(1); CONTACT_EMAIL: VarChar(255); IS_COMPANY: Boolean; COMPANY_NAME: VarChar(200); T: Char(1)

Figure 23: Parameter Insight for a Stored Procedure

Another query writing tool is the ability to **drag and drop object names**, like tables and columns. You can **drag lists of columns** as well. You can drag from the **Database Navigator** or the **Describe Companion**, the latter supports selecting multiple items from the list of available columns.
These 3 tools are available in all code editors as well.



SQL Editor: Firebird 3:2014_FB_Conference.fdb:4 [-Unnamed-]

```
select 
from orders
```

Describe Companion

Session: 2014_FB_Conference.fdb
Object type: Tables
Object: ORDERS

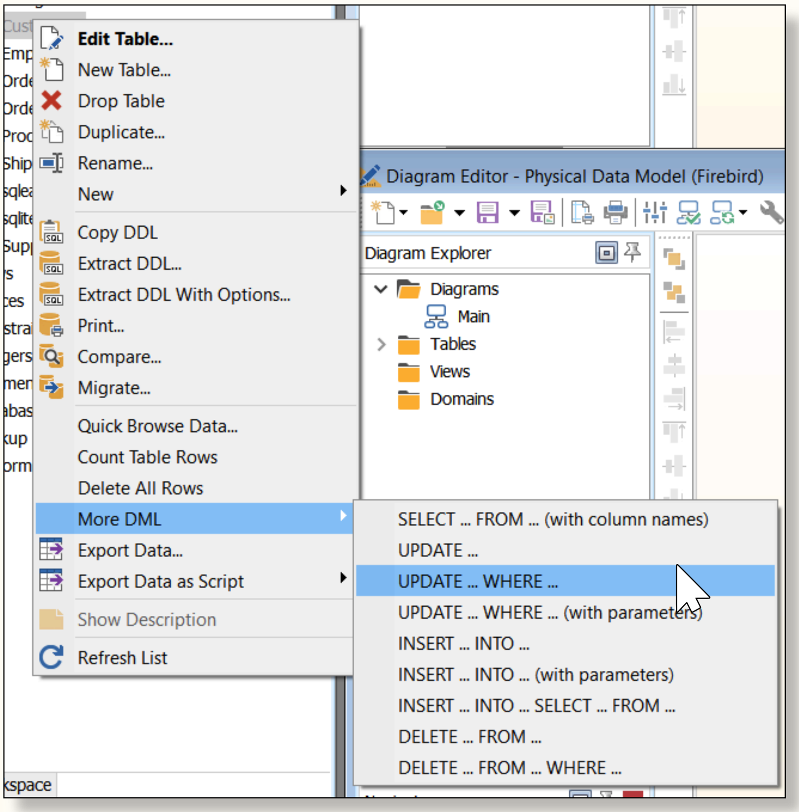
Name	Type	NULL	De
CREATED_AT	Timestamp	N	CU
CUSTOMER_ID	BigInt	N	0
ESTIMATED_SHIP...	Date	Y	
ID	BigInt	N	0
LAST_MODIFIED...	Timestamp	N	CU
PLACED_BY	BigInt	Y	
SHIPPED_AT	Timestamp	Y	

```
select CREATED_AT,
       CUSTOMER_ID,
       ESTIMATED_SHIPPING_DATE,
       ID,
       LAST_MODIFIED_AT,
       PLACED_BY
from orders
```

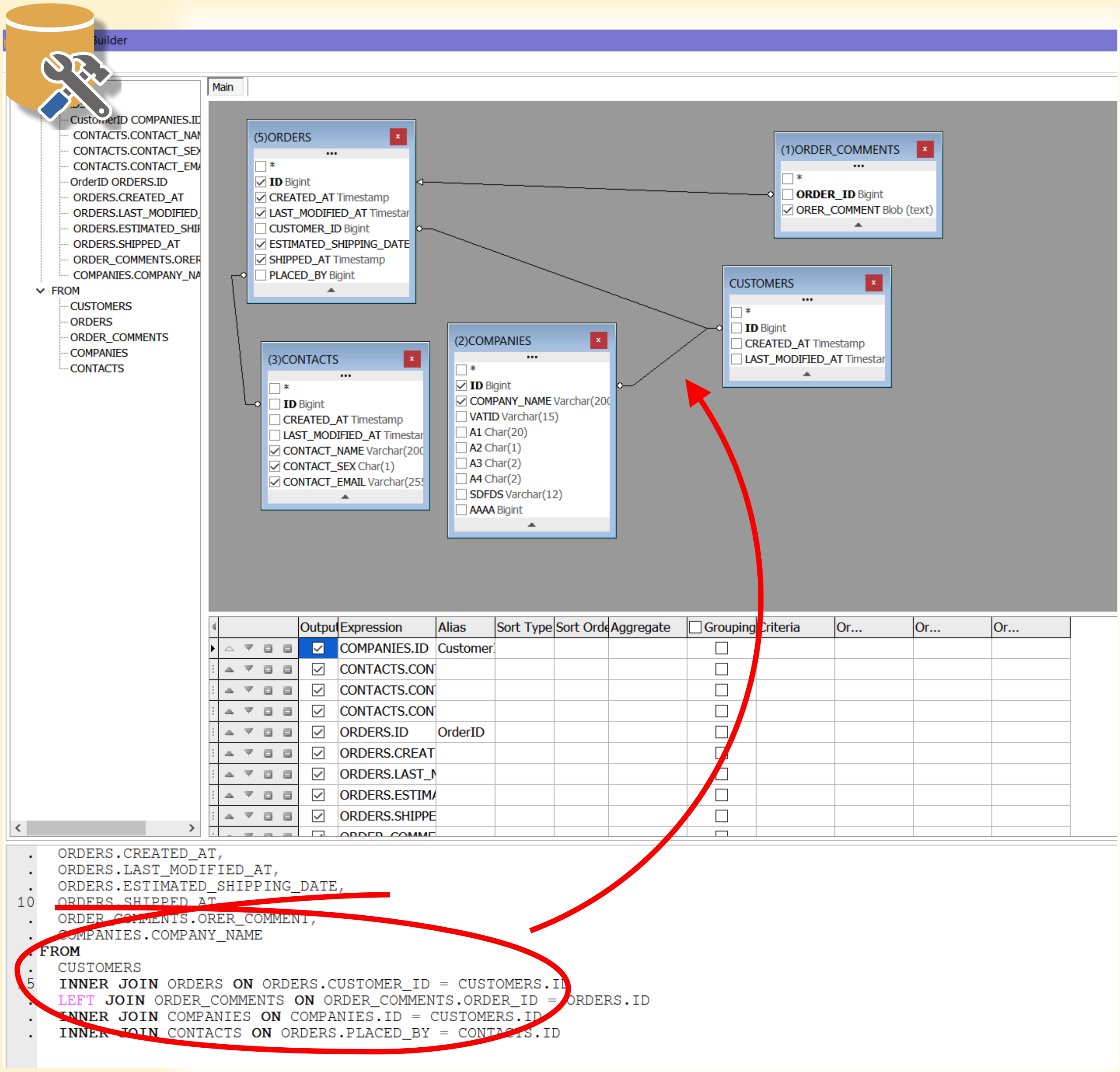
Figure 24: SQL drag-n-drop

Figure 25:
The "More DML" context menu item offers quick SQL

Alternatively, using the context menu on a table allows you to quickly create an SQL statement for SELECT, INSERT, UPDATE or DELETE. Just fill in the blanks and execute!



The **Visual Query Builder** offers another environment to **write your SQL queries.**
If you open the tool with a query available, the query is parsed and displayed in the builder.
If not, you'll start with an empty canvas.
Double click tables or views in the tree to add them or drag them onto the canvas.
It will **automatically add JOIN**s if foreign keys are available in the database.
You can also drag from one column to another to create a JOIN.
Use the context menu on the line to display the join options.



- CustomerID COMPANIES.ID
- CONTACTS.CONTACT_NAI
- CONTACTS.CONTACT_SEX
- CONTACTS.CONTACT_EM/
- OrderID ORDERS.ID
- ORDERS.CREATED_AT
- ORDERS.LAST_MODIFIED_
- ORDERS.ESTIMATED_SHIF
- ORDERS.SHIPPED_AT
- ORDER_COMMENTS.ORER
- COMPANIES.COMPANY_NA
- FROM
  - CUSTOMERS
  - ORDERS
  - ORDER_COMMENTS
  - COMPANIES
  - CONTACTS

(5)ORDERS
- □ *
- ☑ ID Bigint
- ☑ CREATED_AT Timestamp
- ☑ LAST_MODIFIED_AT Timestar
- □ CUSTOMER_ID Bigint
- ☑ ESTIMATED_SHIPPING_DATE
- ☑ SHIPPED_AT Timestamp
- □ PLACED_BY Bigint

(1)ORDER_COMMENTS
- □ *
- □ ORDER_ID Bigint
- ☑ ORER_COMMENT Blob (text)

CUSTOMERS
- □ *
- □ ID Bigint
- □ CREATED_AT Timestamp
- □ LAST_MODIFIED_AT Timestar

(3)CONTACTS
- □ *
- □ ID Bigint
- □ CREATED_AT Timestamp
- □ LAST_MODIFIED_AT Timestar
- ☑ CONTACT_NAME Varchar(20(
- ☑ CONTACT_SEX Char(1)
- ☑ CONTACT_EMAIL Varchar(25!

(2)COMPANIES
- □ *
- ☑ ID Bigint
- ☑ COMPANY_NAME Varchar(20(
- □ VATID Varchar(15)
- □ A1 Char(20)
- □ A2 Char(1)
- □ A3 Char(2)
- □ A4 Char(2)
- □ SDFDS Varchar(12)
- □ AAAA Bigint

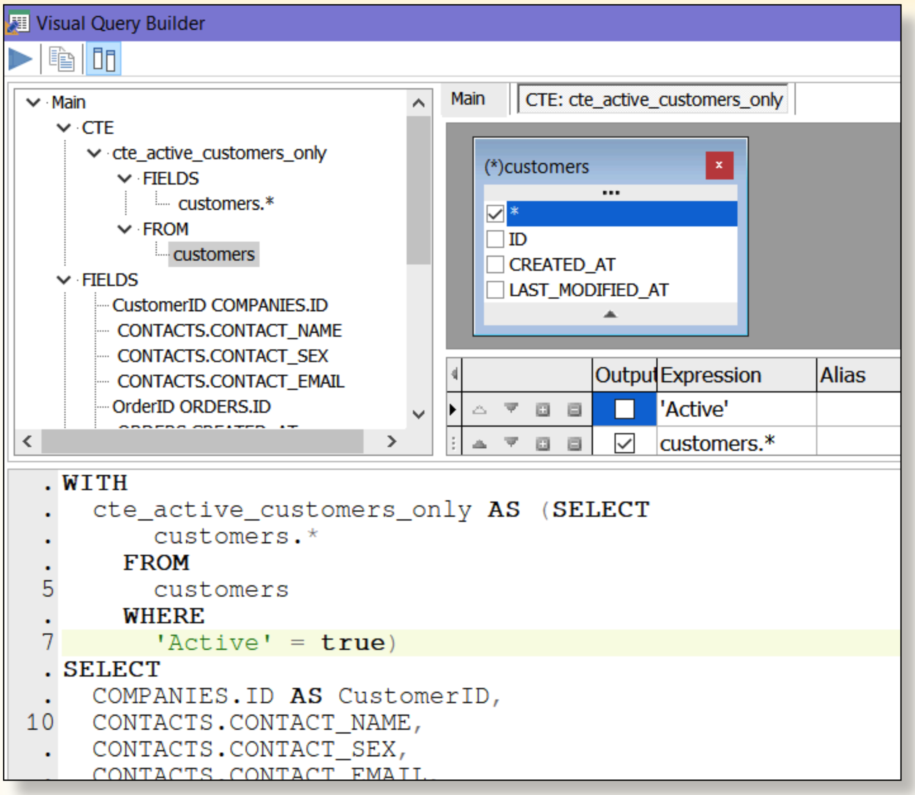
Output	Expression	Alias	Sort Type	Sort Orde	Aggregate	□ Grouping	Criteria	Or...	Or...	Or...
☑	COMPANIES.ID	Customer]				□				
☑	CONTACTS.CON					□				
☑	CONTACTS.CON					□				
☑	CONTACTS.CON					□				
☑	ORDERS.ID	OrderID				□				
☑	ORDERS.CREAT					□				
☑	ORDERS.LAST_N					□				
☑	ORDERS.ESTIM/					□				
☑	ORDERS.SHIPPE					□				
☑	ORDER_COMME					□				

```
    ORDERS.CREATED_AT,
    ORDERS.LAST_MODIFIED_AT,
    ORDERS.ESTIMATED_SHIPPING_DATE,
    ORDERS.SHIPPED_AT
    ORDER_COMMENTS.ORER_COMMENT,
    COMPANIES.COMPANY_NAME
FROM
    CUSTOMERS
    INNER JOIN ORDERS ON ORDERS.CUSTOMER_ID = CUSTOMERS.ID
    LEFT JOIN ORDER_COMMENTS ON ORDER_COMMENTS.ORDER_ID = ORDERS.ID
    INNER JOIN COMPANIES ON COMPANIES.ID = CUSTOMERS.ID
    INNER JOIN CONTACTS ON ORDERS.PLACED_BY = CONTACTS.ID
```

Figure 26:
Visual Query Builder with several visualized joins

Simply checking a colum adds it to the output, additional options can be modified in the grid. **WITH THIS TOOL YOU CAN WRITE SQL QUERIES WITHOUT KNOWLEDGE OF SQL**.

So for example, it's ideal for people writing ad-hoc reports (*more on that later*).
Queries can be complex, you can go beyond a single **SELECT** and also add **UNIONs**, or add a **Common Table Expression (CTE)**, all these will show up in the query tree outline and selecting a particular query or **CTE** in the tree will display it on the canvas.



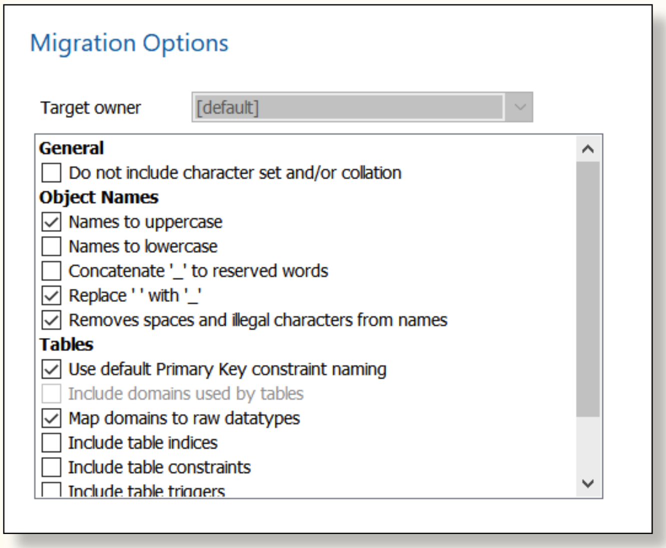
Output	Expression	Alias
□	'Active'	
☑	customers.*	

```
WITH
 cte_active_customers_only AS (SELECT
   customers.*
  FROM
   customers
  WHERE
   'Active' = true)
SELECT
 COMPANIES.ID AS CustomerID,
 CONTACTS.CONTACT_NAME,
 CONTACTS.CONTACT_SEX,
 CONTACTS.CONTACT_EMAIL,
```

Figure 27:
Visual Query Builder with a CTE on a seperate tab

CROSS DATABASE DEVELOPMENT: MIGRATING, COMPARING AND DATA-TRANSFER.

As **Database Workbench** supports multiple database systems, it also features **special cross database development tools.** With the **Database Migration tools** you can convert tables, views, indices and constraints from one database system to another.



Migration Options

Target owner: [default]

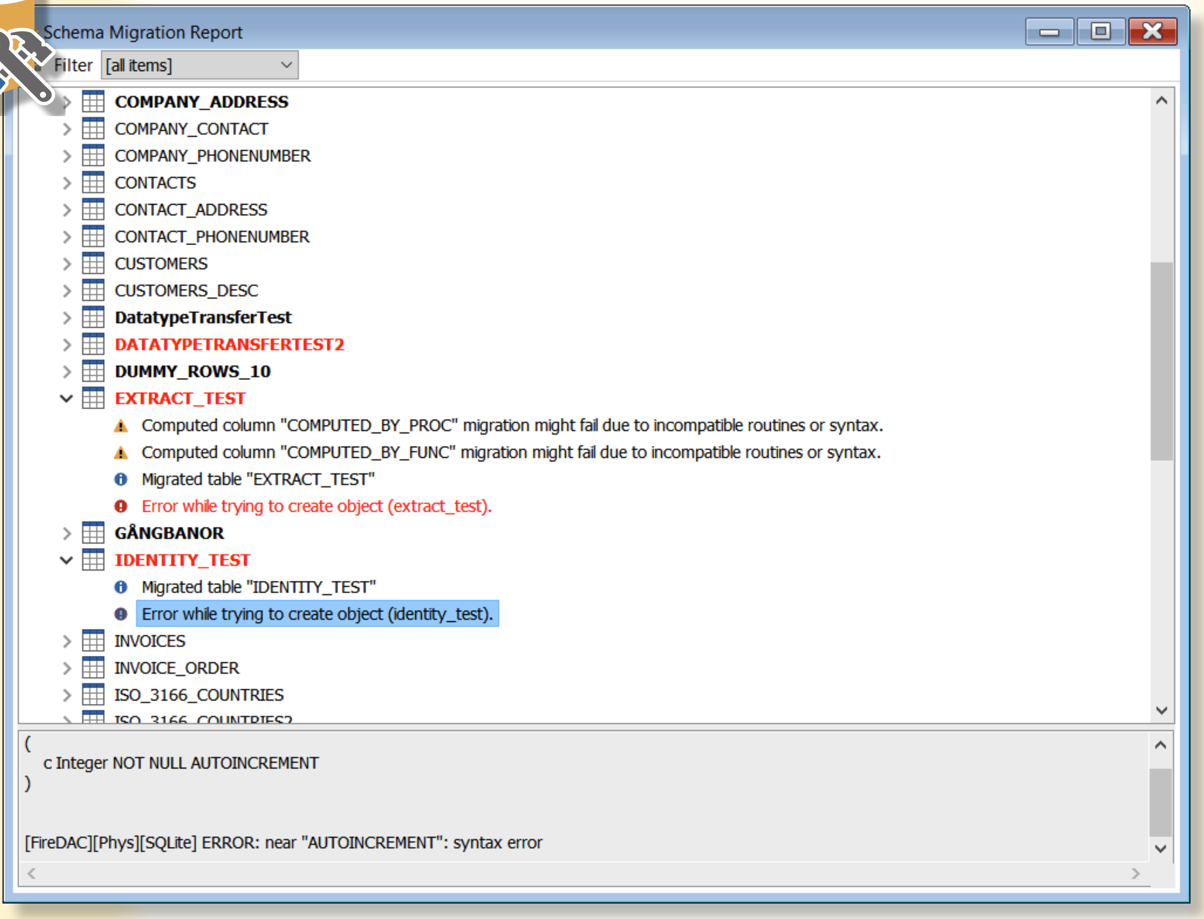
General
- □ Do not include character set and/or collation

Object Names
- ☑ Names to uppercase
- □ Names to lowercase
- □ Concatenate '_' to reserved words
- ☑ Replace ' ' with '_'
- ☑ Removes spaces and illegal characters from names

Tables
- ☑ Use default Primary Key constraint naming
- □ Include domains used by tables
- ☑ Map domains to raw datatypes
- □ Include table indices
- □ Include table constraints
- □ Include table triggers

Figure 28:
Options for database migration

Data types and other options will be automatically converted.
For example, while **MySQL** uses **VARCHAR** for character data, **Oracle** uses **VARCHAR2**.
Firebird uses **BLOB SUB_TYPE TEXT**, while **SQL Server** uses TEXT. Some systems use **AUTOINC** as a data type, while others use it as an attribute to any **INTEGER-based** column. No need to remember these differences, **Database Workbench** does it all for you. Migrating an existing database like this help speed up your cross-database development efforts a lot. Very helpful.

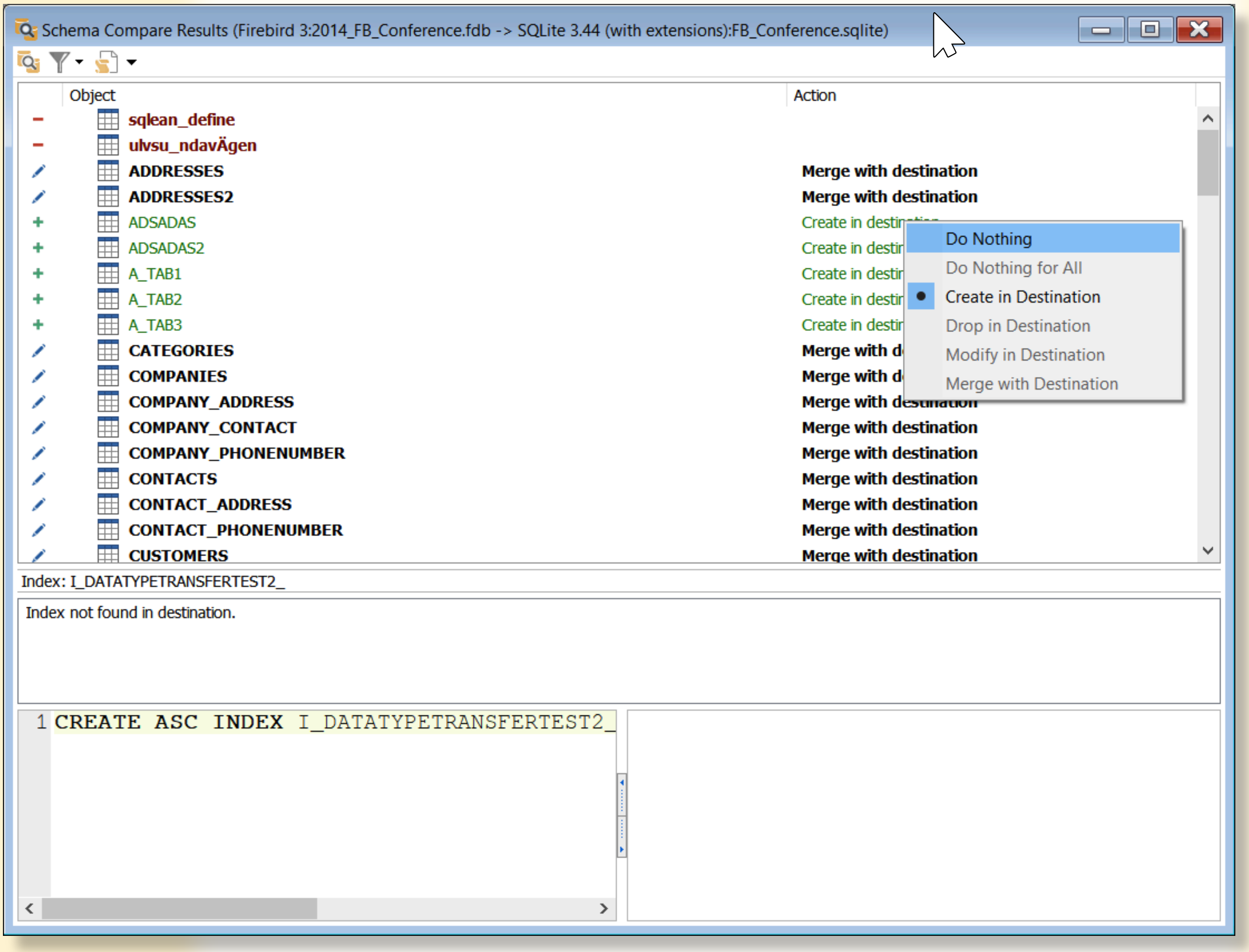


```
COMPANY_ADDRESS
COMPANY_CONTACT
COMPANY_PHONENUMBER
CONTACTS
CONTACT_ADDRESS
CONTACT_PHONENUMBER
CUSTOMERS
CUSTOMERS_DESC
DatatypeTransferTest
DATATYPETRANSFERTEST2
DUMMY_ROWS_10
EXTRACT_TEST
    Computed column "COMPUTED_BY_PROC" migration might fail due to incompatible routines or syntax.
    Computed column "COMPUTED_BY_FUNC" migration might fail due to incompatible routines or syntax.
    Migrated table "EXTRACT_TEST"
    Error while trying to create object (extract_test).
GÅNGBANOR
IDENTITY_TEST
    Migrated table "IDENTITY_TEST"
    Error while trying to create object (identity_test).
INVOICES
INVOICE_ORDER
ISO_3166_COUNTRIES
ISO_3166_COUNTRIES2
```

```
(
  c Integer NOT NULL AUTOINCREMENT
)

[FireDAC][Phys][SQLite] ERROR: near "AUTOINCREMENT": syntax error
```

Figure 29:
Migration report with detailed warnings and error messages

If you want to check meta data of one database against another, you can use the **Database Compare Tool**.
Useful for detecting changes between development and production database, or checking database versions for deployment.
It can even compare databases between different database systems, using the same data type conversions as the **Database Migration Tool**.



	Object	Action
−	sqlean_define	
−	ulvsu_ndavÄgen	
✎	ADDRESSES	Merge with destination
✎	ADDRESSES2	Merge with destination
+	ADSADAS	Create in destin
+	ADSADAS2	Create in destir
+	A_TAB1	Create in destir
+	A_TAB2	Create in destir
+	A_TAB3	Create in destir
✎	CATEGORIES	Merge with d
✎	COMPANIES	Merge with d
✎	COMPANY_ADDRESS	Merge with destination
✎	COMPANY_CONTACT	Merge with destination
✎	COMPANY_PHONENUMBER	Merge with destination
✎	CONTACTS	Merge with destination
✎	CONTACT_ADDRESS	Merge with destination
✎	CONTACT_PHONENUMBER	Merge with destination
✎	CUSTOMERS	Merge with destination

Index: I_DATATYPETRANSFERTEST2_

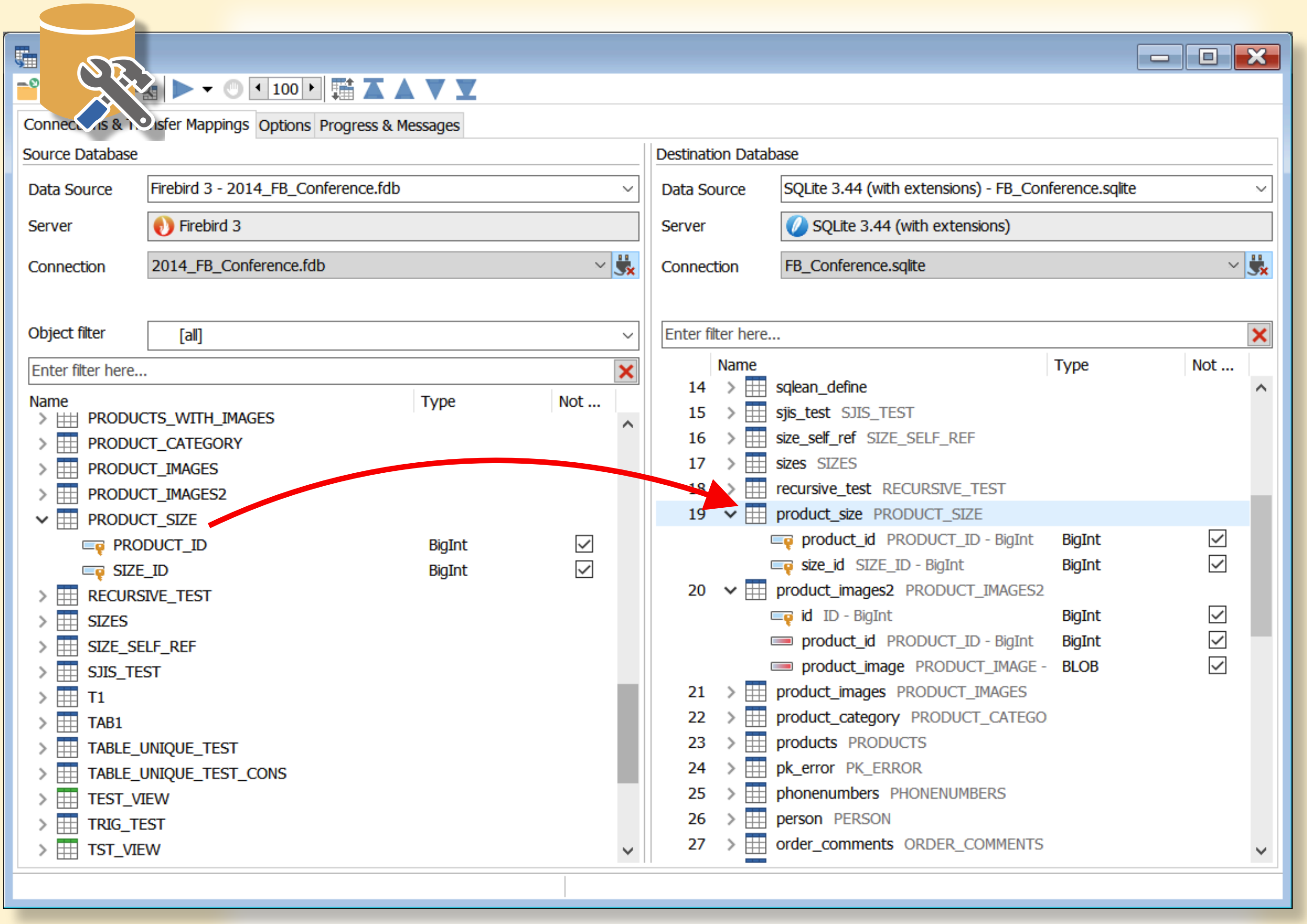
Index not found in destination.

```
CREATE ASC INDEX I_DATATYPETRANSFERTEST2_
```

Figure 30:
Compare result, you can define what action to take

The results are shown after **comparing** and **Database Workbench** can generate a change script you can execute on the **destination database** to **modify the schema** objects.

To move data between databases, the **DataPump tool** can be used.
It supports transferring all data, even large binary data, as fast as possible.
After selecting both a source and destination database, the tables can automatically be arranged in the correct order as per the foreign key constraints in the database.
For example, an **ORDER record** can only exist for a particular **CUSTOMER record**, so the data in table **CUSTOMER** needs to be moved first.



Connec...s & T...sfer Mappings | Options | Progress & Messages

Source Database

Data Source: Firebird 3 - 2014_FB_Conference.fdb
Server: Firebird 3
Connection: 2014_FB_Conference.fdb
Object filter: [all]
Enter filter here...

Name	Type	Not ...
PRODUCTS_WITH_IMAGES		
PRODUCT_CATEGORY		
PRODUCT_IMAGES		
PRODUCT_IMAGES2		
PRODUCT_SIZE		
PRODUCT_ID	BigInt	☑
SIZE_ID	BigInt	☑
RECURSIVE_TEST		
SIZES		
SIZE_SELF_REF		
SJIS_TEST		
T1		
TAB1		
TABLE_UNIQUE_TEST		
TABLE_UNIQUE_TEST_CONS		
TEST_VIEW		
TRIG_TEST		
TST_VIEW		

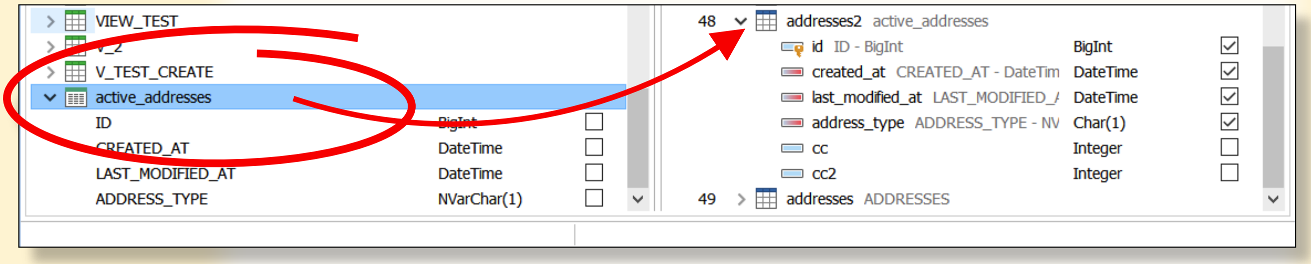
Destination Database

Data Source: SQLite 3.44 (with extensions) - FB_Conference.sqlite
Server: SQLite 3.44 (with extensions)
Connection: FB_Conference.sqlite
Enter filter here...

	Name	Type	Not ...
14	sqlean_define		
15	sjis_test SJIS_TEST		
16	size_self_ref SIZE_SELF_REF		
17	sizes SIZES		
18	recursive_test RECURSIVE_TEST		
19	product_size PRODUCT_SIZE		
	product_id PRODUCT_ID - BigInt	BigInt	☑
	size_id SIZE_ID - BigInt	BigInt	☑
20	product_images2 PRODUCT_IMAGES2		
	id ID - BigInt	BigInt	☑
	product_id PRODUCT_ID - BigInt	BigInt	☑
	product_image PRODUCT_IMAGE -	BLOB	☑
21	product_images PRODUCT_IMAGES		
22	product_category PRODUCT_CATEGO		
23	products PRODUCTS		
24	pk_error PK_ERROR		
25	phonenumbers PHONENUMBERS		
26	person PERSON		
27	order_comments ORDER_COMMENTS		

Figure 31: DataPump with linked transfers

You can either simply transfer data from one table to another, or modify how the DataPump fetches the data from the source table:
this can be done by setting a **WHERE-clause** for the source, or even more complex, by using a completely self-written SQL statement to fetch data.
This can be useful if the data for the target table needs to come from multiple source tables, for example, or from a selectable stored procedure.



Name	Type	Not ...
VIEW_TEST		
V_2		
V_TEST_CREATE		
active_addresses		
ID	BigInt	□
CREATED_AT	DateTime	□
LAST_MODIFIED_AT	DateTime	□
ADDRESS_TYPE	NVarChar(1)	□

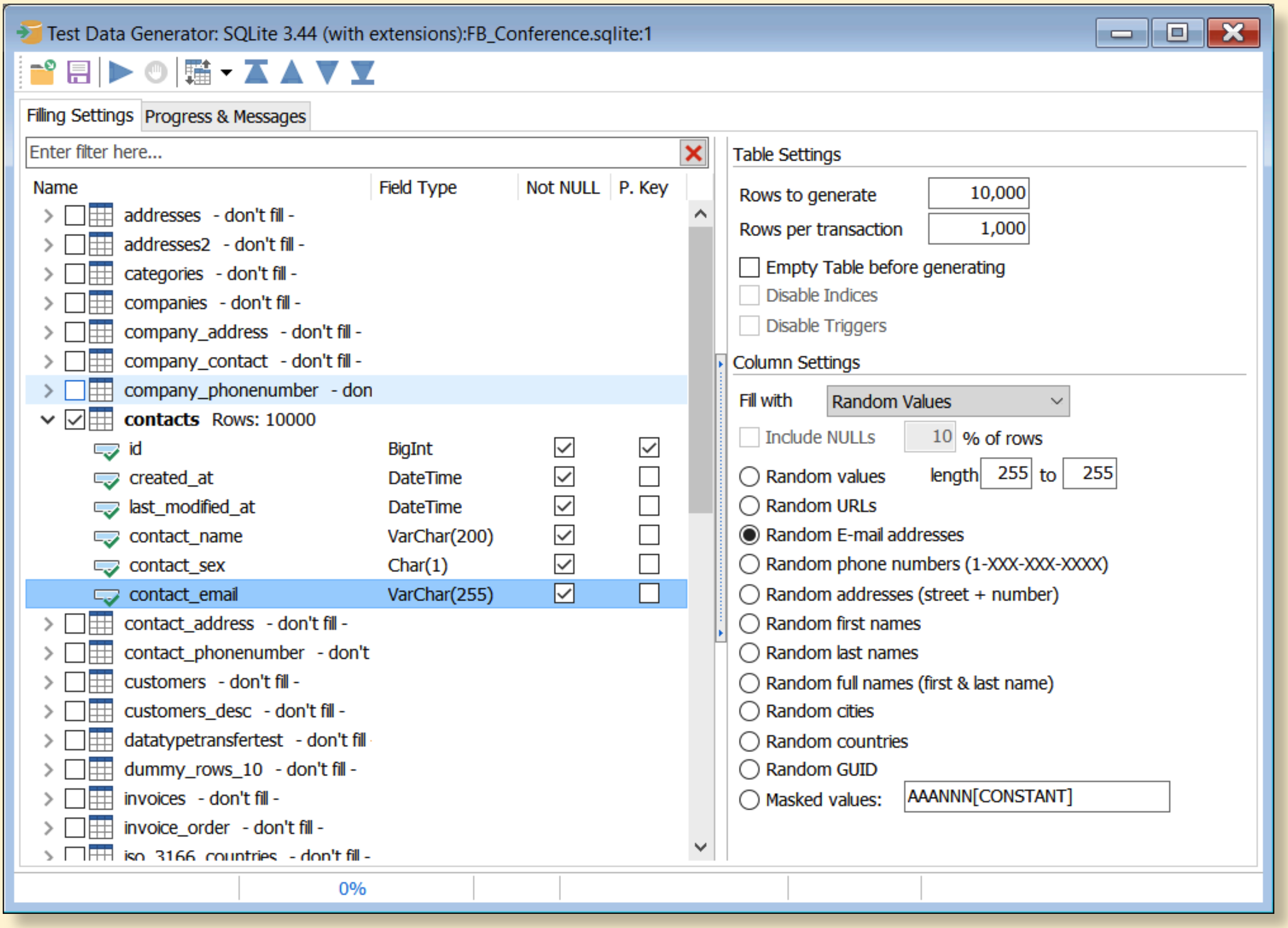
	Name	Type	Not ...
48	addresses2 active_addresses		
	id ID - BigInt	BigInt	☑
	created_at CREATED_AT - DateTim	DateTime	☑
	last_modified_at LAST_MODIFIED_/	DateTime	☑
	address_type ADDRESS_TYPE - NV	Char(1)	☑
	cc	Integer	□
	cc2	Integer	□
49	addresses ADDRESSES		

Figure 32: SQL based source to transfer data to destination table

The **DataPump** can disable triggers in the destination databases to **avoid business logic coded into the destination database being executed.** It can also disable indices and enabled them afterwards, to increase data transfer speed.

TESTING & DEBUGGING

If you haven't got existing data, the **Test Data Generato**r tool can help you to create fake data for testing purposes. You need data to test your performance, reporting, user interface and so on, **never test with a (*near*) empty database.**



Test Data Generator: SQLite 3.44 (with extensions):FB_Conference.sqlite:1

Filling Settings | Progress & Messages

Enter filter here...

Name	Field Type	Not NULL	P. Key
□ addresses - don't fill -			
□ addresses2 - don't fill -			
□ categories - don't fill -			
□ companies - don't fill -			
□ company_address - don't fill -			
□ company_contact - don't fill -			
□ company_phonenumber - don			
☑ **contacts** Rows: 10000			
id	BigInt	☑	☑
created_at	DateTime	☑	□
last_modified_at	DateTime	☑	□
contact_name	VarChar(200)	☑	□
contact_sex	Char(1)	☑	□
contact_email	VarChar(255)	☑	□
□ contact_address - don't fill -			
□ contact_phonenumber - don't			
□ customers - don't fill -			
□ customers_desc - don't fill -			
□ datatypetransfertest - don't fill			
□ dummy_rows_10 - don't fill -			
□ invoices - don't fill -			
□ invoice_order - don't fill -			
□ iso_3166_countries - don't fill -			

Table Settings

Rows to generate: 10,000

Rows per transaction: 1,000

□ Empty Table before generating

□ Disable Indices

□ Disable Triggers

Column Settings

Fill with: Random Values

□ Include NULLs 10 % of rows

○ Random values length 255 to 255

○ Random URLs

● Random E-mail addresses

○ Random phone numbers (1-XXX-XXX-XXXX)

○ Random addresses (street + number)

○ Random first names

○ Random last names

○ Random full names (first & last name)

○ Random cities

○ Random countries

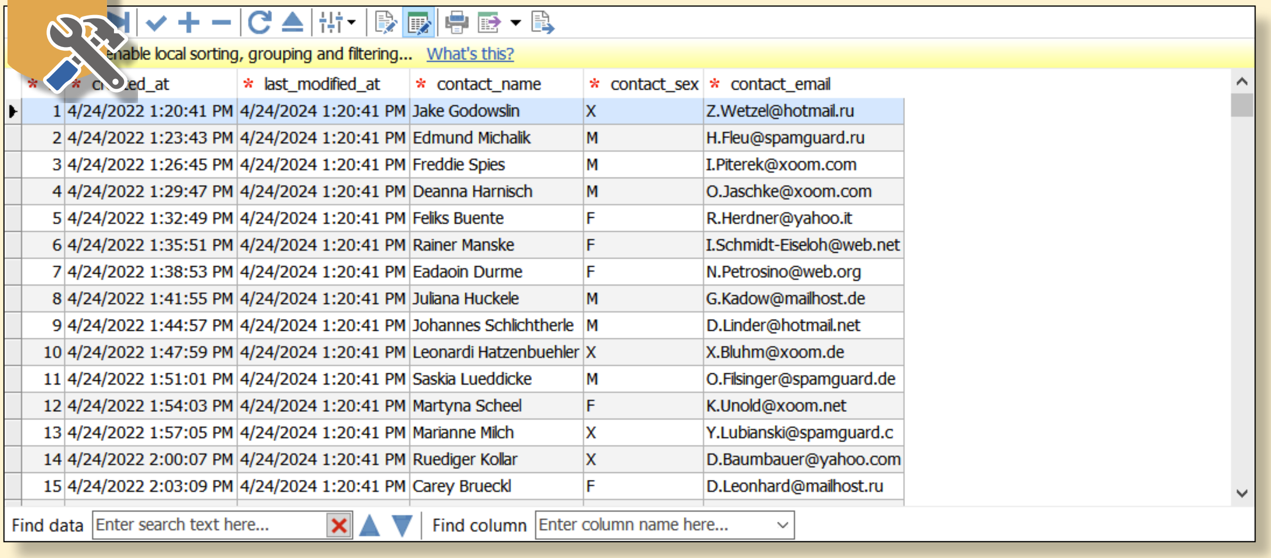
○ Random GUID

○ Masked values: AAANNN[CONSTANT]

0%

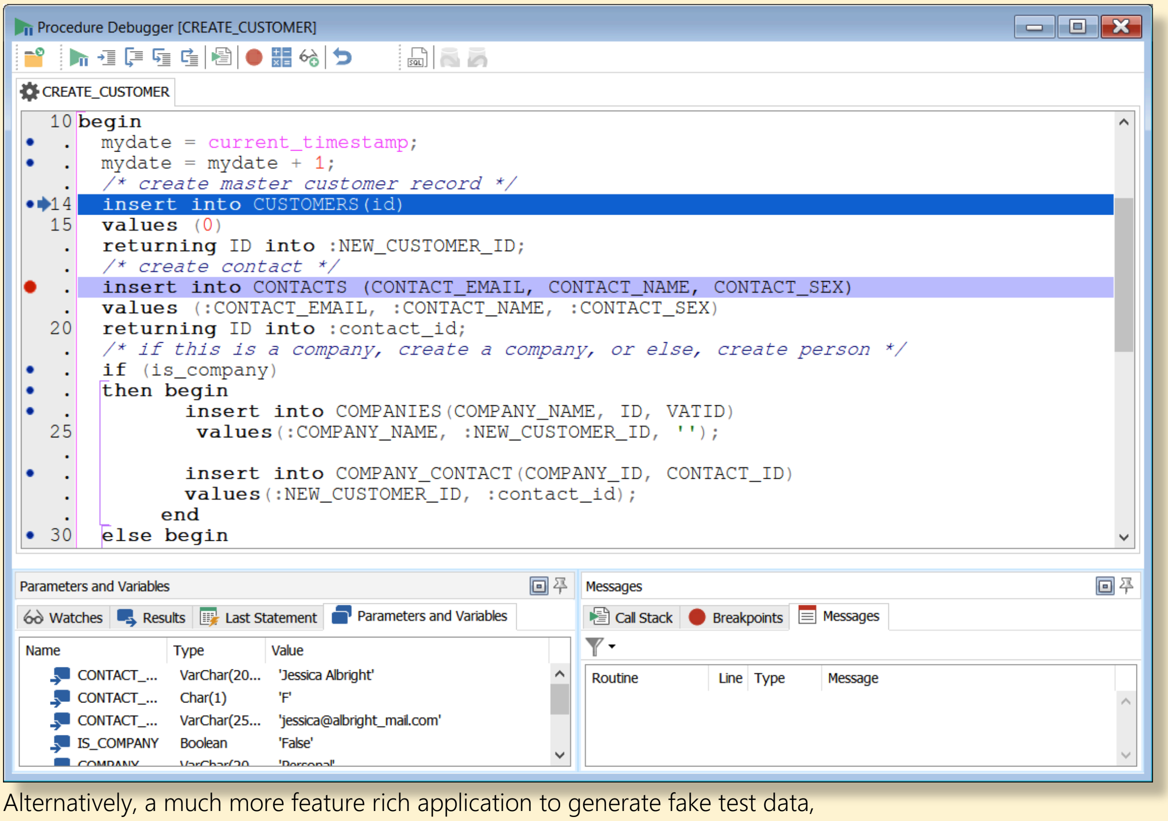
Figure 33:
Test Data Generator with example options

In order to **fill the database with real-life-like data**, it can generate fake e-mail addresses, street names, first and last names, postal codes, it can use existing images, generate large pieces of text, and more.


Enable local sorting, grouping and filtering... What's this?
Find data
Enter search text here...
Find column
Enter column name here...

	* created_at	* last_modified_at	* contact_name	* contact_sex	* contact_email
1	4/24/2022 1:20:41 PM	4/24/2024 1:20:41 PM	Jake Godowslin	X	Z.Wetzel@hotmail.ru
2	4/24/2022 1:23:43 PM	4/24/2024 1:20:41 PM	Edmund Michalik	M	H.Fleu@spamguard.ru
3	4/24/2022 1:26:45 PM	4/24/2024 1:20:41 PM	Freddie Spies	M	I.Piterek@xoom.com
4	4/24/2022 1:29:47 PM	4/24/2024 1:20:41 PM	Deanna Harnisch	M	O.Jaschke@xoom.com
5	4/24/2022 1:32:49 PM	4/24/2024 1:20:41 PM	Feliks Buente	F	R.Herdner@yahoo.it
6	4/24/2022 1:35:51 PM	4/24/2024 1:20:41 PM	Rainer Manske	F	I.Schmidt-Eiseloh@web.net
7	4/24/2022 1:38:53 PM	4/24/2024 1:20:41 PM	Eadaoin Durme	F	N.Petrosino@web.org
8	4/24/2022 1:41:55 PM	4/24/2024 1:20:41 PM	Juliana Huckele	M	G.Kadow@mailhost.de
9	4/24/2022 1:44:57 PM	4/24/2024 1:20:41 PM	Johannes Schlichtherle	M	D.Linder@hotmail.net
10	4/24/2022 1:47:59 PM	4/24/2024 1:20:41 PM	Leonardi Hatzenbuehler	X	X.Bluhm@xoom.de
11	4/24/2022 1:51:01 PM	4/24/2024 1:20:41 PM	Saskia Lueddicke	M	O.Filsinger@spamguard.de
12	4/24/2022 1:54:03 PM	4/24/2024 1:20:41 PM	Martyna Scheel	F	K.Unold@xoom.net
13	4/24/2022 1:57:05 PM	4/24/2024 1:20:41 PM	Marianne Milch	X	Y.Lubianski@spamguard.c
14	4/24/2022 2:00:07 PM	4/24/2024 1:20:41 PM	Ruediger Kollar	X	D.Baumbauer@yahoo.com
15	4/24/2022 2:03:09 PM	4/24/2024 1:20:41 PM	Carey Brueckl	F	D.Leonhard@mailhost.ru

Figure 34:
Result of the above settings for data generation



```
begin
  mydate = current_timestamp;
  mydate = mydate + 1;
  /* create master customer record */
  insert into CUSTOMERS(id)
  values (0)
  returning ID into :NEW_CUSTOMER_ID;
  /* create contact */
  insert into CONTACTS (CONTACT_EMAIL, CONTACT_NAME, CONTACT_SEX)
  values (:CONTACT_EMAIL, :CONTACT_NAME, :CONTACT_SEX)
  returning ID into :contact_id;
  /* if this is a company, create a company, or else, create person */
  if (is_company)
  then begin
         insert into COMPANIES(COMPANY_NAME, ID, VATID)
          values(:COMPANY_NAME, :NEW_CUSTOMER_ID, '');

         insert into COMPANY_CONTACT(COMPANY_ID, CONTACT_ID)
         values(:NEW_CUSTOMER_ID, :contact_id);
       end
  else begin
```

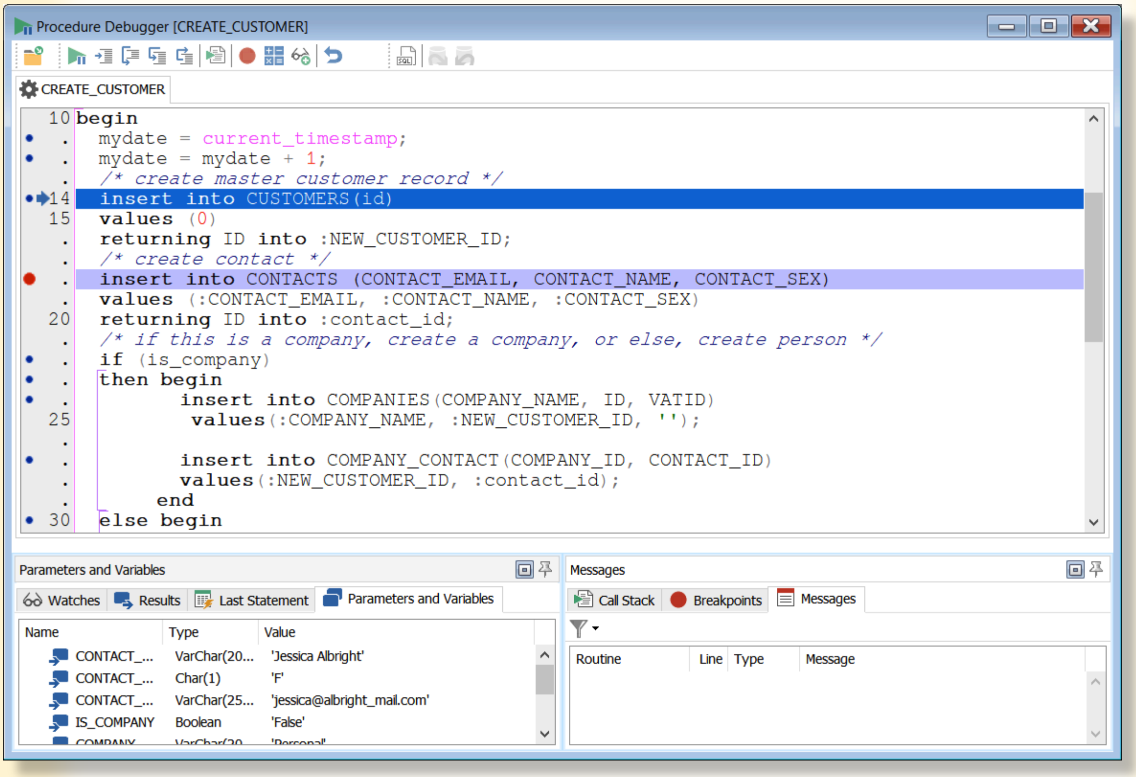
Name	Type	Value
CONTACT_...	VarChar(20...	'Jessica Albright'
CONTACT_...	Char(1)	'F'
CONTACT_...	VarChar(25...	'jessica@albright_mail.com'
IS_COMPANY	Boolean	'False'
COMPANY	VarChar(20	'Personal'

Routine	Line	Type	Message

Figure 35:
Result of the above settings for data generation

Alternatively, a much more feature rich application to generate fake test data, named **Advanced Data Generator** is available as well, see **`https://www.upscene.com/advanced_data_generator/`**

If you use code on the database itself, **triggers, stored procedures, stored functions or packages**, **Database Workbench** offers a **debugger** for this code. While **Oracle and PostgreSQL** provide a debugging interface, InterBase, Firebird and MySQL do not. In order to debug code for those database systems, Database Workbench emulates the code as if it were executed on the database, step by step, line by line.



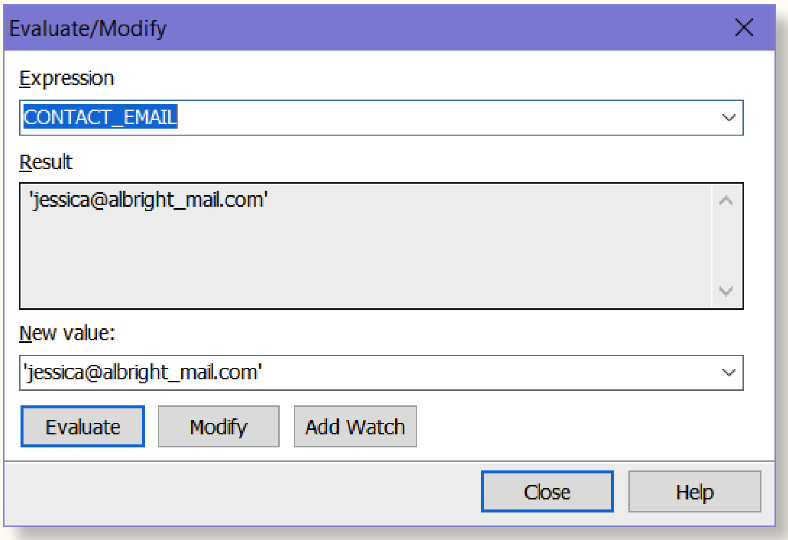
```
begin
  mydate = current_timestamp;
  mydate = mydate + 1;
  /* create master customer record */
  insert into CUSTOMERS(id)
  values (0)
  returning ID into :NEW_CUSTOMER_ID;
  /* create contact */
  insert into CONTACTS (CONTACT_EMAIL, CONTACT_NAME, CONTACT_SEX)
  values (:CONTACT_EMAIL, :CONTACT_NAME, :CONTACT_SEX)
  returning ID into :contact_id;
  /* if this is a company, or else, create person */
  if (is_company)
  then begin
         insert into COMPANIES(COMPANY_NAME, ID, VATID)
          values(:COMPANY_NAME, :NEW_CUSTOMER_ID, '');

         insert into COMPANY_CONTACT(COMPANY_ID, CONTACT_ID)
         values(:NEW_CUSTOMER_ID, :contact_id);
       end
  else begin
```

Name	Type	Value
CONTACT_...	VarChar(20...	'Jessica Albright'
CONTACT_...	Char(1)	'F'
CONTACT_...	VarChar(25...	'jessica@albright_mail.com'
IS_COMPANY	Boolean	'False'
COMPANY	VarChar(20	'Personal'

Routine	Line	Type	Message

Figure 36: Stored Procedure debugger, with breakpoint set

When you start a stored routine, the debugger will prompt you for input values.
If it's a trigger, you can browse the table for values.
Continue to start the actual routine. From here going forward, you're able to execute the stored routine statement by statement, just like with **Delphi or Lazarus**. You can modify variable or parameter values or execute SQL statements to check the current state of data.



Expression: CONTACT_EMAIL

Result: 'jessica@albright_mail.com'

New value: 'jessica@albright_mail.com'

Figure 37: Evaluate/Modify dialog to modify values of parameters and variables

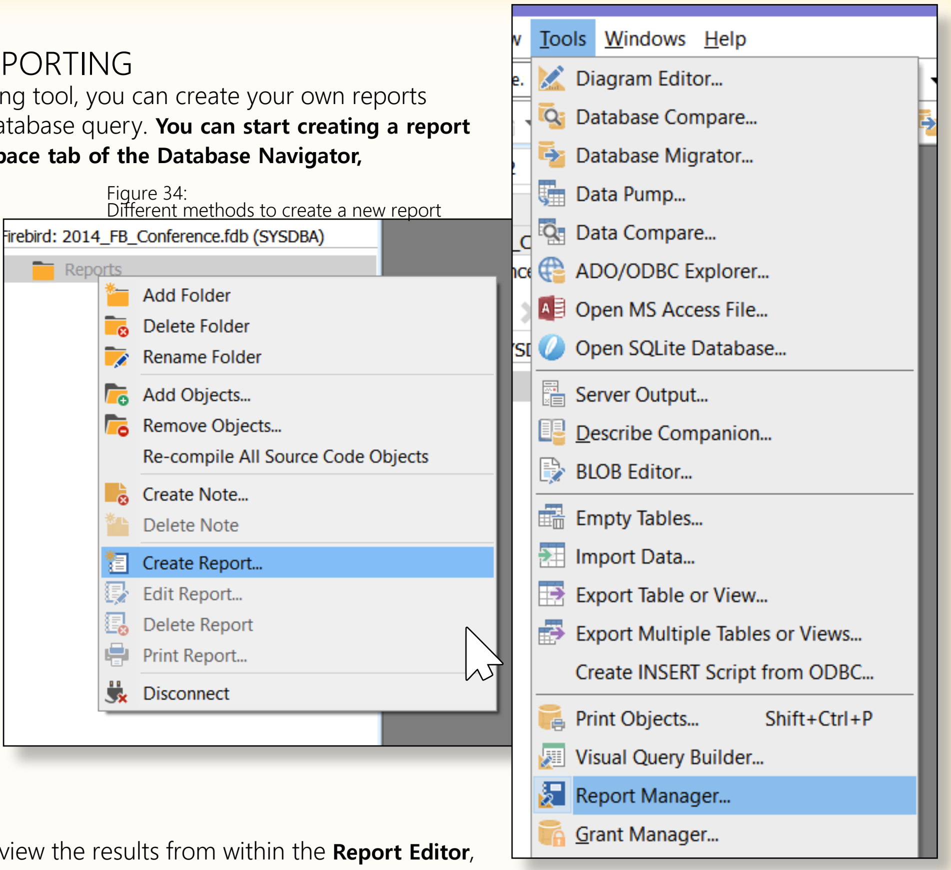
After the routine has finished, you can use the **Debugger SQL Editor** to rollback or commit the transaction.

AD-HOC REPORTING

With the reporting tool, you can create your own reports based on any database query. **You can start creating a report from the Workspace tab of the Database Navigator,** or from the **Report Manager** under the **Tools menu item**. While reports in the Workspace are saved for that particular database configuration, a report in the **Report Manager** has the added benefit of possibly being used for multiple databases if these include the same tables used as a source for data.

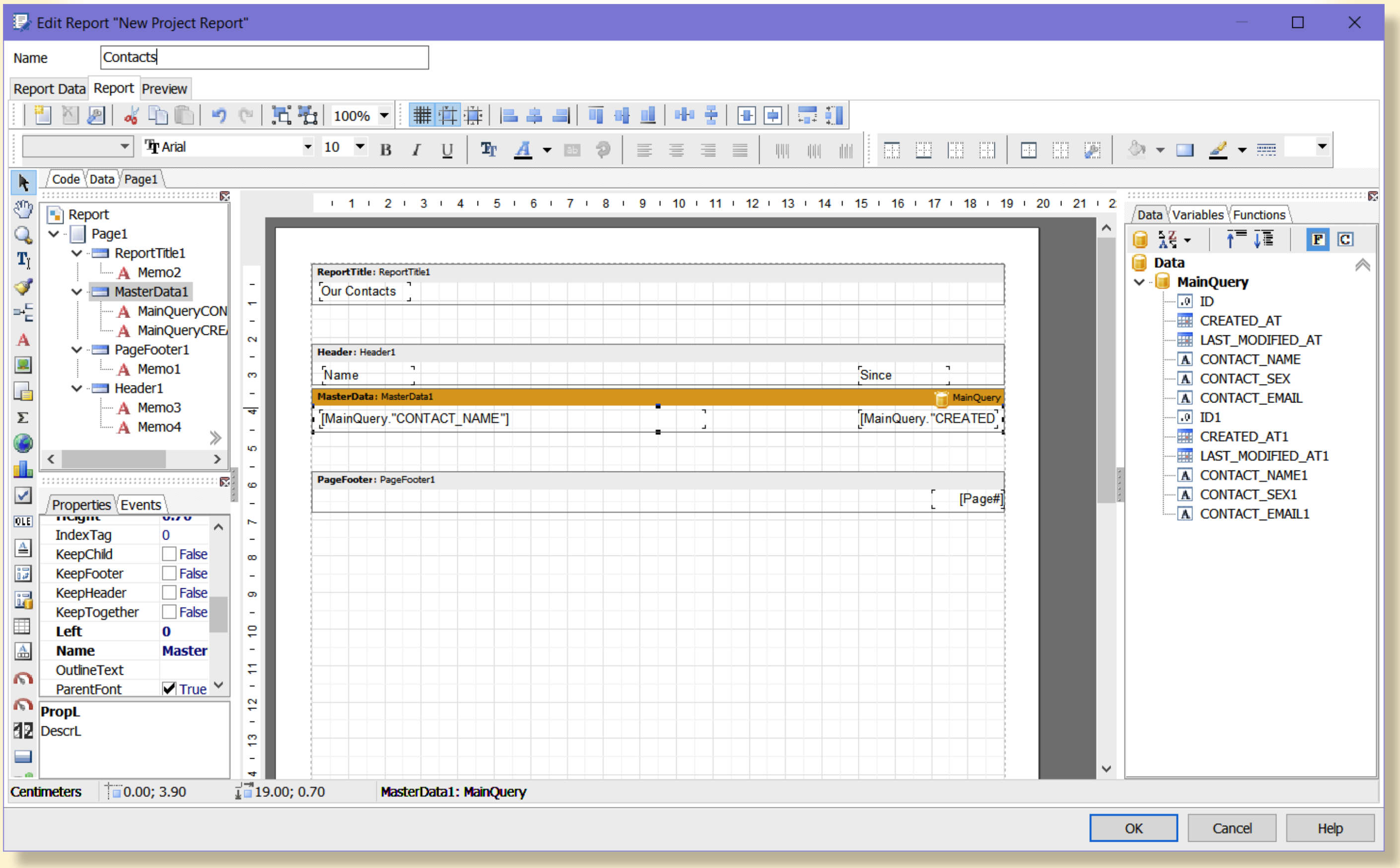
In order to create and test your query, the **Visual Query Builder** is opened. After composing your query, click OK and you'll see the Report Editor.

Figure 34:
Different methods to create a new report



You can preview the results from within the **Report Editor**, just click the tab **Preview** and the report will fetch the data and display it.

Figure 38:
Report Editor

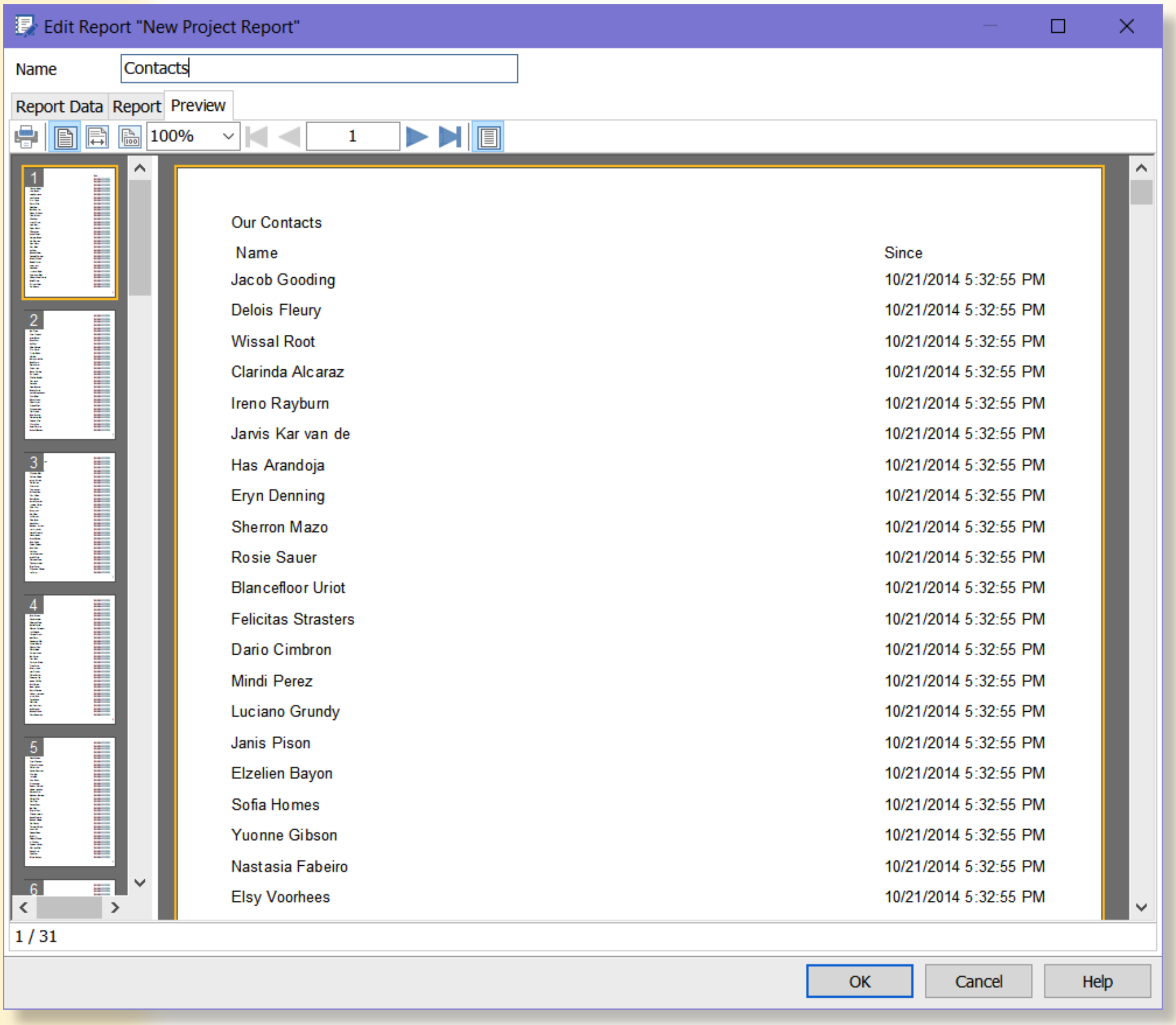


Edit Report "New Project Report"

Name: Contacts

IndexTag	0
KeepChild	□ False
KeepFooter	□ False
KeepHeader	□ False
KeepTogether	□ False
Left	**0**
Name	**Master**
OutlineText	
ParentFont	☑ True

Centimeters 0.00; 3.90 19.00; 0.70 MasterData1: MainQuery



Name: Contacts

Page: 1

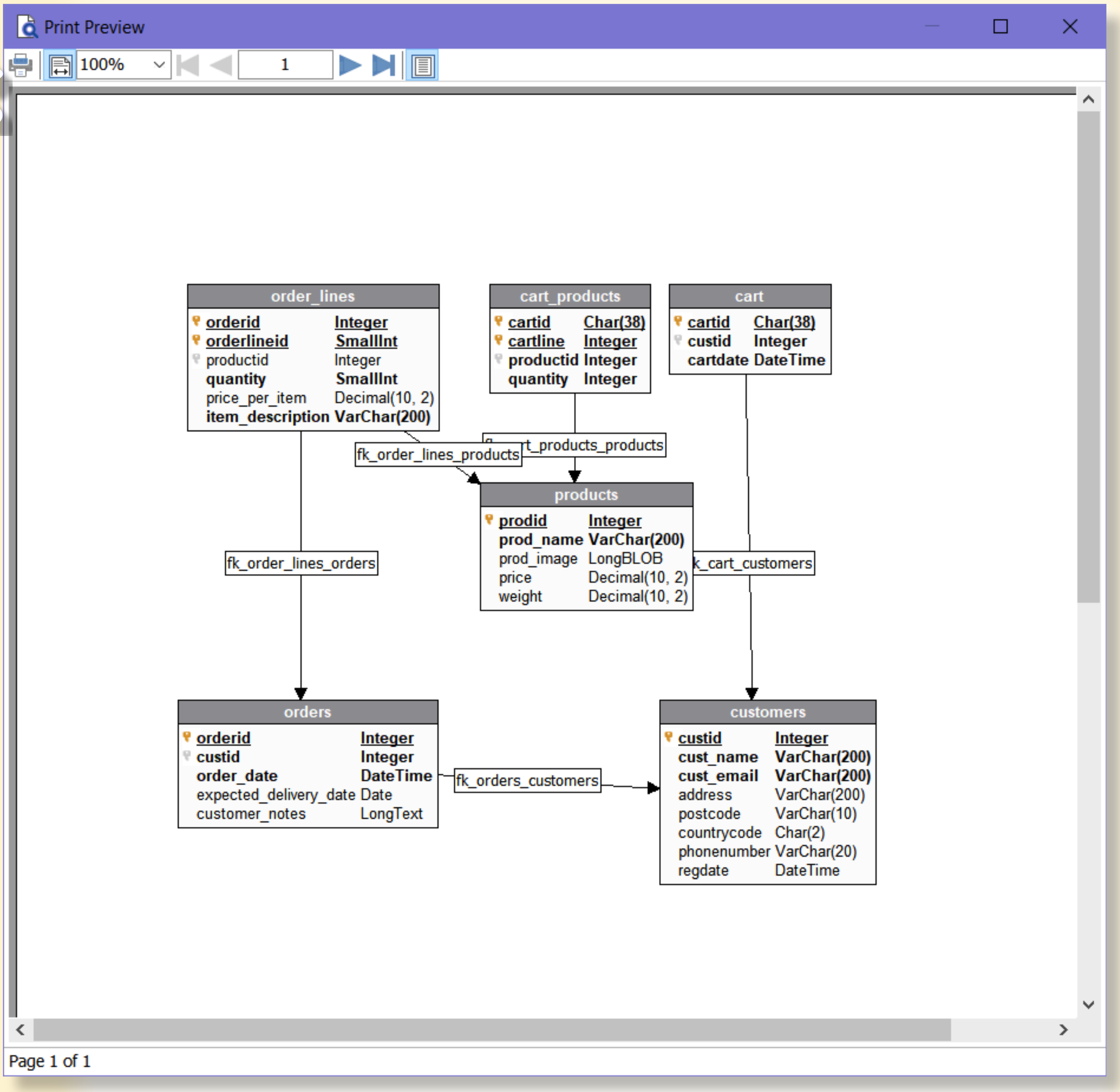
Our Contacts	
Name	Since
Jacob Gooding	10/21/2014 5:32:55 PM
Delois Fleury	10/21/2014 5:32:55 PM
Wissal Root	10/21/2014 5:32:55 PM
Clarinda Alcaraz	10/21/2014 5:32:55 PM
Ireno Rayburn	10/21/2014 5:32:55 PM
Jarvis Kar van de	10/21/2014 5:32:55 PM
Has Arandoja	10/21/2014 5:32:55 PM
Eryn Denning	10/21/2014 5:32:55 PM
Sherron Mazo	10/21/2014 5:32:55 PM
Rosie Sauer	10/21/2014 5:32:55 PM
Blancefloor Uriot	10/21/2014 5:32:55 PM
Felicitas Strasters	10/21/2014 5:32:55 PM
Dario Cimbron	10/21/2014 5:32:55 PM
Mindi Perez	10/21/2014 5:32:55 PM
Luciano Grundy	10/21/2014 5:32:55 PM
Janis Pison	10/21/2014 5:32:55 PM
Elzelien Bayon	10/21/2014 5:32:55 PM
Sofia Homes	10/21/2014 5:32:55 PM
Yuonne Gibson	10/21/2014 5:32:55 PM
Nastasia Fabeiro	10/21/2014 5:32:55 PM
Elsy Voorhees	10/21/2014 5:32:55 PM

1 / 31

Figure 39:
Report Preview

DATABASE DOCUMENTATION

And there's another printing feature. A visual overview of your database and relations between tables can be created using the modeling tools, either reverse-engineer an existing database or create the database from a model.



order_lines	
orderid	**Integer**
orderlineid	**SmallInt**
productid	Integer
quantity	**SmallInt**
price_per_item	Decimal(10, 2)
item_description	**VarChar(200)**

cart_products	
cartid	**Char(38)**
cartline	**Integer**
productid	**Integer**
quantity	**Integer**

cart	
cartid	**Char(38)**
custid	**Integer**
cartdate	**DateTime**

products	
prodid	**Integer**
prod_name	**VarChar(200)**
prod_image	LongBLOB
price	Decimal(10, 2)
weight	Decimal(10, 2)

orders	
orderid	**Integer**
custid	**Integer**
order_date	**DateTime**
expected_delivery_date	Date
customer_notes	LongText

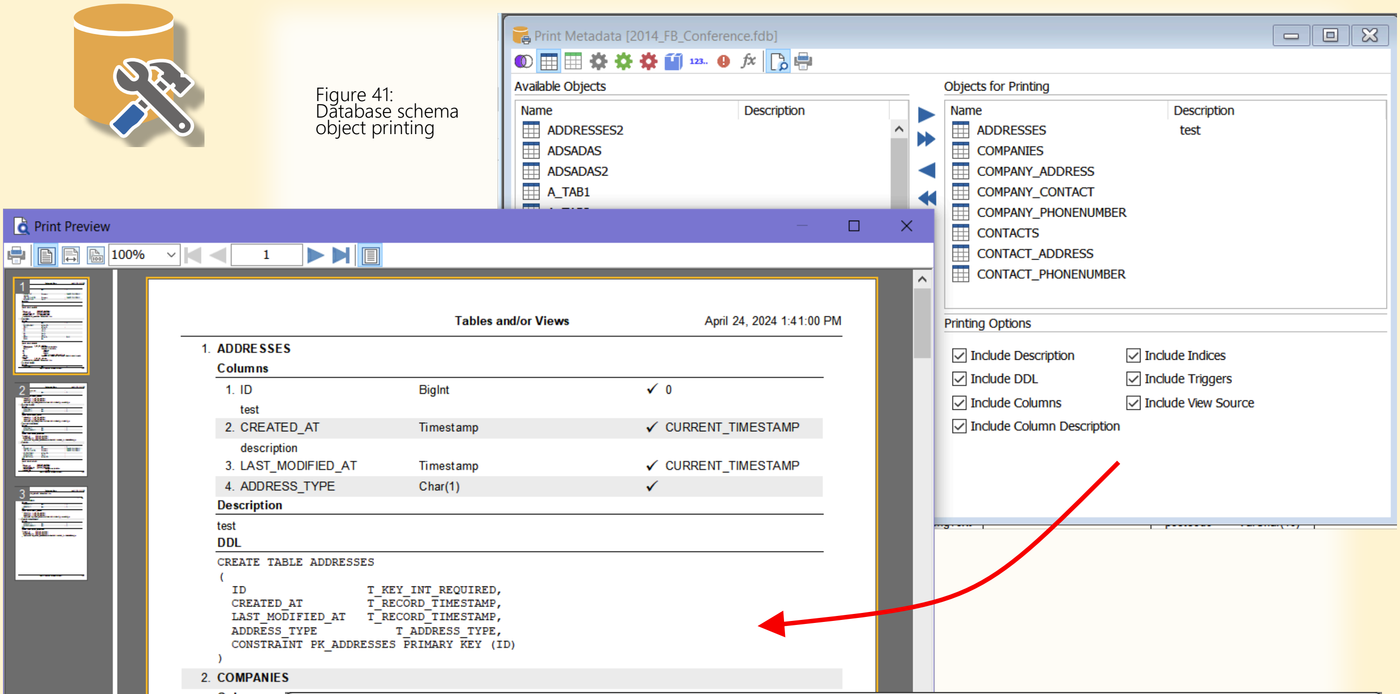
customers	
custid	**Integer**
cust_name	**VarChar(200)**
cust_email	**VarChar(200)**
address	VarChar(200)
postcode	VarChar(10)
countrycode	Char(2)
phonenumber	VarChar(20)
regdate	DateTime

Figure 40: Print preview of a reverse engineered database

But there's more: **indices, triggers, stored procedures**.
Detailed documentation for tables inside the database using table and column descriptions is supported by nearly all database systems. But you need a way to use it.

This is where meta data printing comes in.
With this tool you can print (or print-to-PDF) schema objects like tables, views, triggers, indices and so on.

Object descriptions, source code or **DDL** is optional.
You can print all, or a selection of items. Also useful when you want to give a third party documentation, for example, for a set of database View's that can be used for export data or custom reporting functionality.



Print Metadata [2014_FB_Conference.fdb]

Available Objects

Name	Description
ADDRESSES2	
ADSADAS	
ADSADAS2	
A_TAB1	

Objects for Printing

Name	Description
ADDRESSES	test
COMPANIES	
COMPANY_ADDRESS	
COMPANY_CONTACT	
COMPANY_PHONENUMBER	
CONTACTS	
CONTACT_ADDRESS	
CONTACT_PHONENUMBER	

Printing Options

☑ Include Description ☑ Include Indices
☑ Include DDL ☑ Include Triggers
☑ Include Columns ☑ Include View Source
☑ Include Column Description

Print Preview

100% 1

Tables and/or Views April 24, 2024 1:41:00 PM

1. ADDRESSES

Columns

1. ID	BigInt	✓	0
test			
2. CREATED_AT	Timestamp	✓	CURRENT_TIMESTAMP
description			
3. LAST_MODIFIED_AT	Timestamp	✓	CURRENT_TIMESTAMP
4. ADDRESS_TYPE	Char(1)	✓	

Description

test

DDL

```
CREATE TABLE ADDRESSES
(
  ID                T_KEY_INT_REQUIRED,
  CREATED_AT        T_RECORD_TIMESTAMP,
  LAST_MODIFIED_AT  T_RECORD_TIMESTAMP,
  ADDRESS_TYPE          T_ADDRESS_TYPE,
  CONSTRAINT PK_ADDRESSES PRIMARY KEY (ID)
)
```

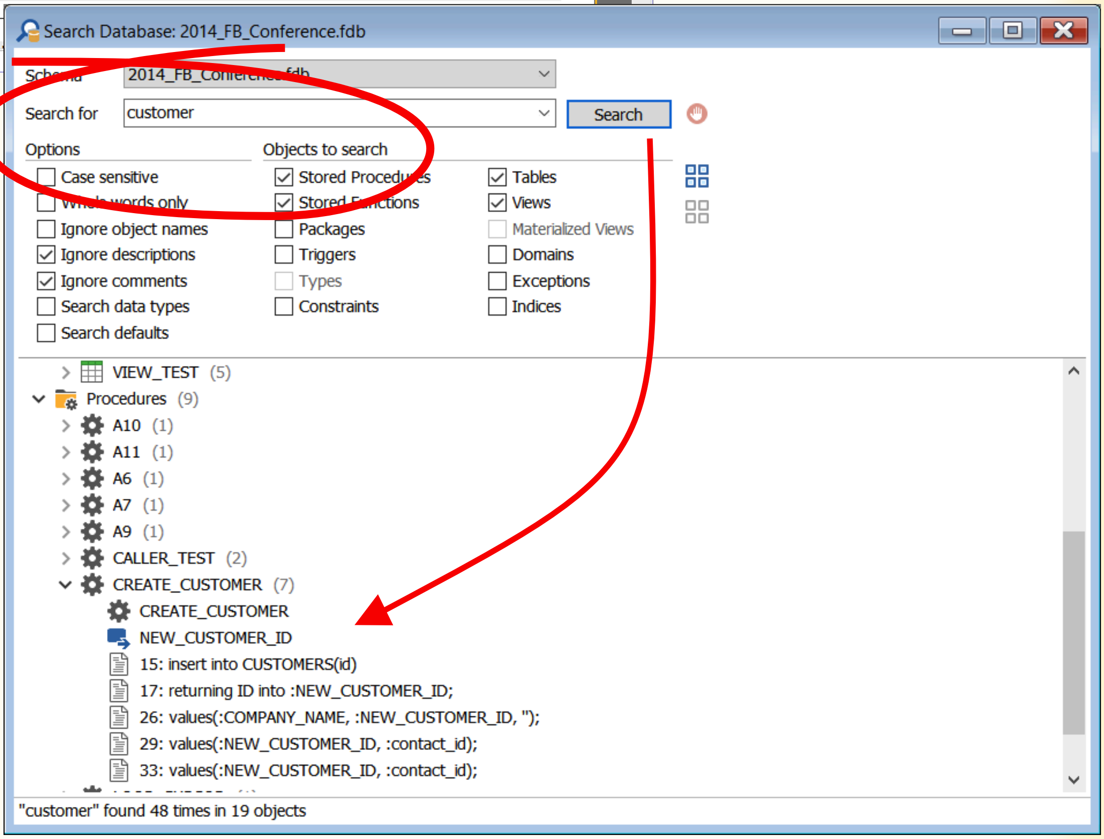
2. COMPANIES

Columns

1. ID

1 / 3

Figure 41:
Database schema object printing



Search Database: 2014_FB_Conference.fdb

Schema: 2014_FB_Conference.fdb

Search for: customer

Options
☐ Case sensitive
☐ Whole words only
☐ Ignore object names
☑ Ignore descriptions
☑ Ignore comments
☐ Search data types
☐ Search defaults

Objects to search
☑ Stored Procedures
☑ Stored Functions
☐ Packages
☐ Triggers
☐ Types
☐ Constraints
☑ Tables
☑ Views
☐ Materialized Views
☐ Domains
☐ Exceptions
☐ Indices

```
> VIEW_TEST (5)
v Procedures (9)
  > A10 (1)
  > A11 (1)
  > A6 (1)
  > A7 (1)
  > A9 (1)
  > CALLER_TEST (2)
  v CREATE_CUSTOMER (7)
      CREATE_CUSTOMER
      NEW_CUSTOMER_ID
      15: insert into CUSTOMERS(id)
      17: returning ID into :NEW_CUSTOMER_ID;
      26: values(:COMPANY_NAME, :NEW_CUSTOMER_ID, '');
      29: values(:NEW_CUSTOMER_ID, :contact_id);
      33: values(:NEW_CUSTOMER_ID, :contact_id);
```

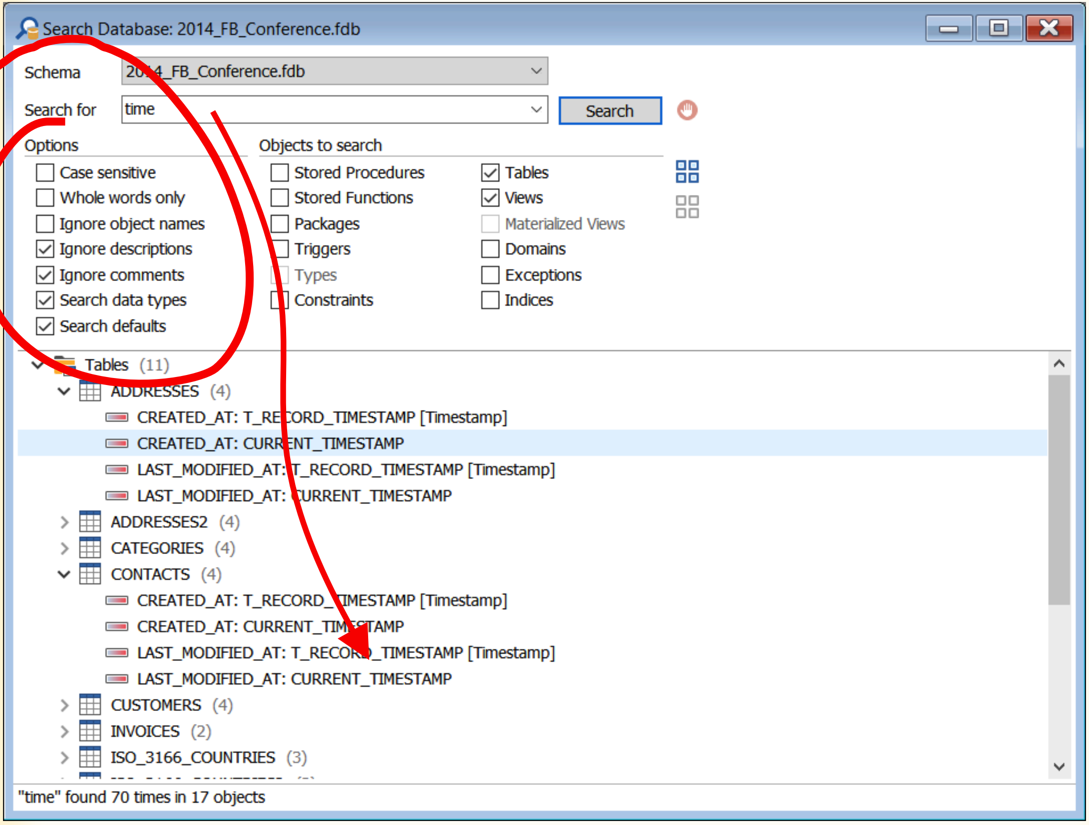
"customer" found 48 times in 19 objects

Figure 42:
Database search can search for objects and in source code

OTHER VALUABLE TOOLS

There's not enough room here to show you all the tools, but let's highlight two more. **The Database Meta Data Search tool** let's you search for objects inside your database. It can search for object names, but also **searches inside the source code of views, triggers and stored routines.**

For columns, it can even search for appearances in column defaults or data types (*either a raw data type or domain/user defined data type name*).
As not all database systems support dependency tracking or don't support it at the same detail level, using the *Database Search tool* can help you to keep track of objects, occurrences in different places or use in code or default definitions.



Search Database: 2014_FB_Conference.fdb

Schema: 2014_FB_Conference.fdb
Search for: time

Options
- ☐ Case sensitive
- ☐ Whole words only
- ☐ Ignore object names
- ☑ Ignore descriptions
- ☑ Ignore comments
- ☑ Search data types
- ☑ Search defaults

Objects to search
- ☐ Stored Procedures
- ☐ Stored Functions
- ☐ Packages
- ☐ Triggers
- ☐ Types
- ☐ Constraints
- ☑ Tables
- ☑ Views
- ☐ Materialized Views
- ☐ Domains
- ☐ Exceptions
- ☐ Indices

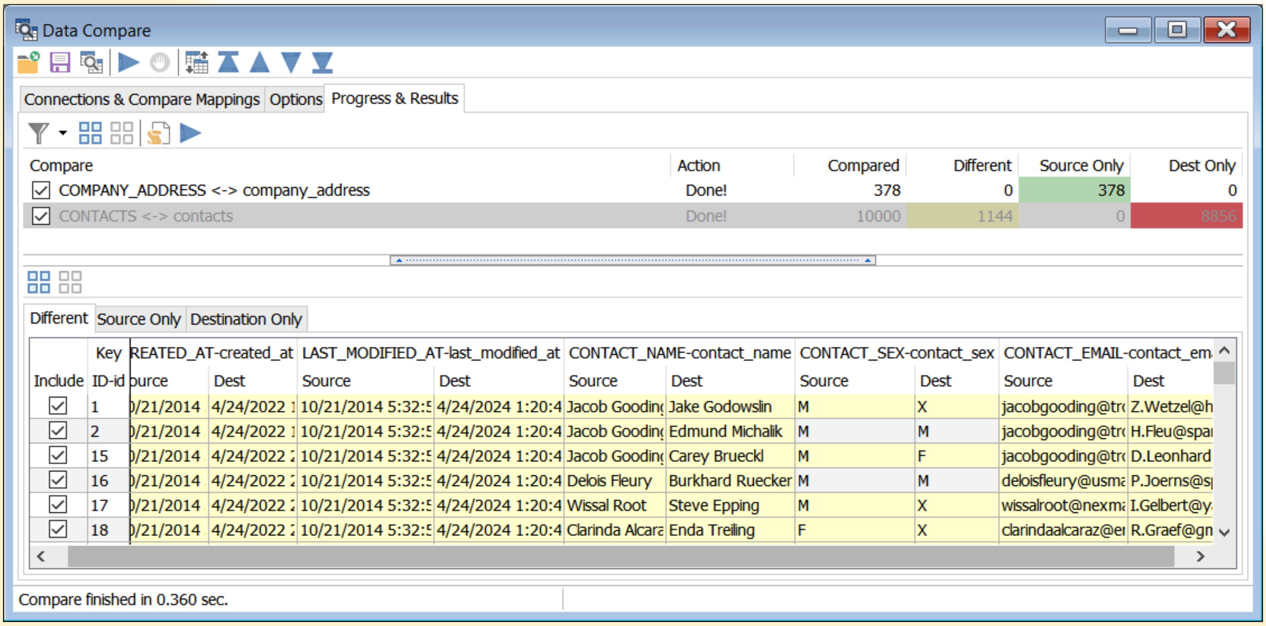
```
Tables (11)
  ADDRESSES (4)
    CREATED_AT: T_RECORD_TIMESTAMP [Timestamp]
    CREATED_AT: CURRENT_TIMESTAMP
    LAST_MODIFIED_AT: T_RECORD_TIMESTAMP [Timestamp]
    LAST_MODIFIED_AT: CURRENT_TIMESTAMP
  ADDRESSES2 (4)
  CATEGORIES (4)
  CONTACTS (4)
    CREATED_AT: T_RECORD_TIMESTAMP [Timestamp]
    CREATED_AT: CURRENT_TIMESTAMP
    LAST_MODIFIED_AT: T_RECORD_TIMESTAMP [Timestamp]
    LAST_MODIFIED_AT: CURRENT_TIMESTAMP
  CUSTOMERS (4)
  INVOICES (2)
  ISO_3166_COUNTRIES (3)
```

"time" found 70 times in 17 objects

Figure 43:
Database search can also search for data types and column defaults

Besides the fore mentioned *Meta Data Compare*, there's also a *Data Compare tool*. Especially useful for **data analysis, checking of logging functionality or simply to compare** one state of a database to another. Just as with the **DataPump** tool, you link tables in a source and destination database together. If the columns are different in name, you can link those manually as well.

Figure 44:
Data Compare with linked tables



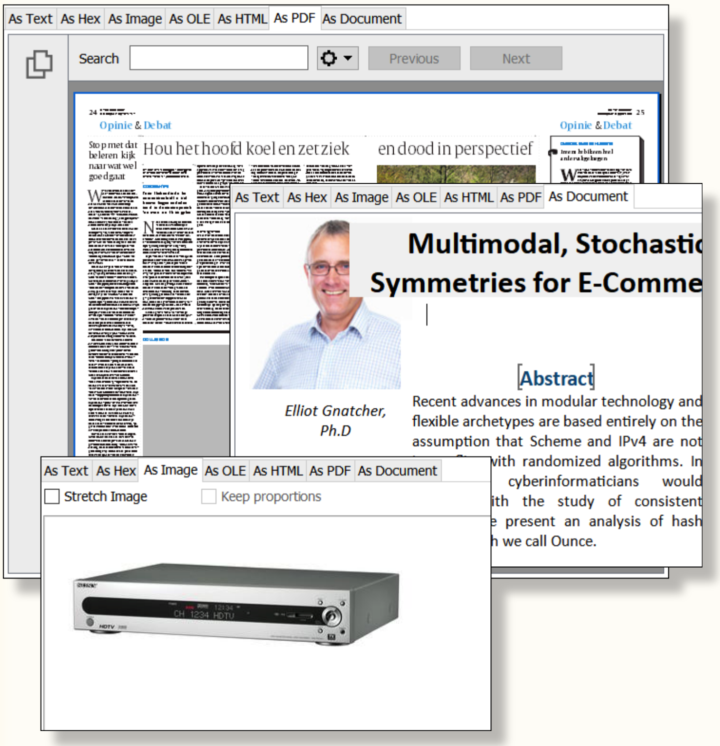
Data Compare

Compare	Action	Compared	Different	Source Only	Dest Only
☑ COMPANY_ADDRESS <-> company_address	Done!	378	0	378	0
☑ CONTACTS <-> contacts	Done!	10000	1144	0	8856

	Key	REATED_AT-created_at		LAST_MODIFIED_AT-last_modified_at		CONTACT_NAME-contact_name		CONTACT_SEX-contact_sex		CONTACT_EMAIL-contact_em	
Include	ID-id	ource	Dest	Source	Dest	Source	Dest	Source	Dest	Source	Dest
☑	1	)/21/2014	4/24/2022 1	10/21/2014 5:32:5	4/24/2024 1:20:4	Jacob Gooding	Jake Godowslin	M	X	jacobgooding@tr	Z.Wetzel@h
☑	2	)/21/2014	4/24/2022 1	10/21/2014 5:32:5	4/24/2024 1:20:4	Jacob Gooding	Edmund Michalik	M	M	jacobgooding@tr	H.Fleu@spa
☑	15	)/21/2014	4/24/2022 2	10/21/2014 5:32:5	4/24/2024 1:20:4	Jacob Gooding	Carey Brueckl	M	F	jacobgooding@tr	D.Leonhard
☑	16	)/21/2014	4/24/2022 2	10/21/2014 5:32:5	4/24/2024 1:20:4	Delois Fleury	Burkhard Ruecker	M	M	deloisfleury@usma	P.Joerns@s
☑	17	)/21/2014	4/24/2022 2	10/21/2014 5:32:5	4/24/2024 1:20:4	Wissal Root	Steve Epping	M	X	wissalroot@nexma	I.Gelbert@y
☑	18	)/21/2014	4/24/2022 2	10/21/2014 5:32:5	4/24/2024 1:20:4	Clarinda Alcara	Enda Treiling	F	X	clarindaalcaraz@e	R.Graef@gn

Compare finished in 0.360 sec.

A database contains more than just text or numbers, it can also contain images, **HTML or documents like PDFs**. You can view these with the **BLOB Editor**, either found under the Tools menu item, or directly embedded in the SQL Editor.



Multimodal, Stochastic
Symmetries for E-Comme

Elliot Gnatcher, Ph.D

Abstract

Recent advances in modular technology and flexible archetypes are based entirely on the assumption that Scheme and IPv4 are not with randomized algorithms. In cyberinformaticians would ith the study of consistent e present an analysis of hash h we call Ounce.

Figure 45:
BLOB Editor, shown with PDF, document and image

When you browse data, the **BLOB Editor** will attempt to automatically detect the type of data, if it's an image, it will display the image, if it's a PDF, it will display the PDF.

If it can't determine the type of the content, it will display a hex-editor. The **BLOB Editor** supports and detects the following formats: **DOC, DOCX, RTF, PDF, HTML, XML, ICO, SGI, PCX, PSD, BMP, GIF, TIFF, PSP, PNG, EPS, WMF** and **JPG**.

ENTERPRISE EDITION & TEAMSERVER

Database Workbench requires you to register servers and possibly databases so you can use those with the application. These are stored in configuration files in your Windows profile.

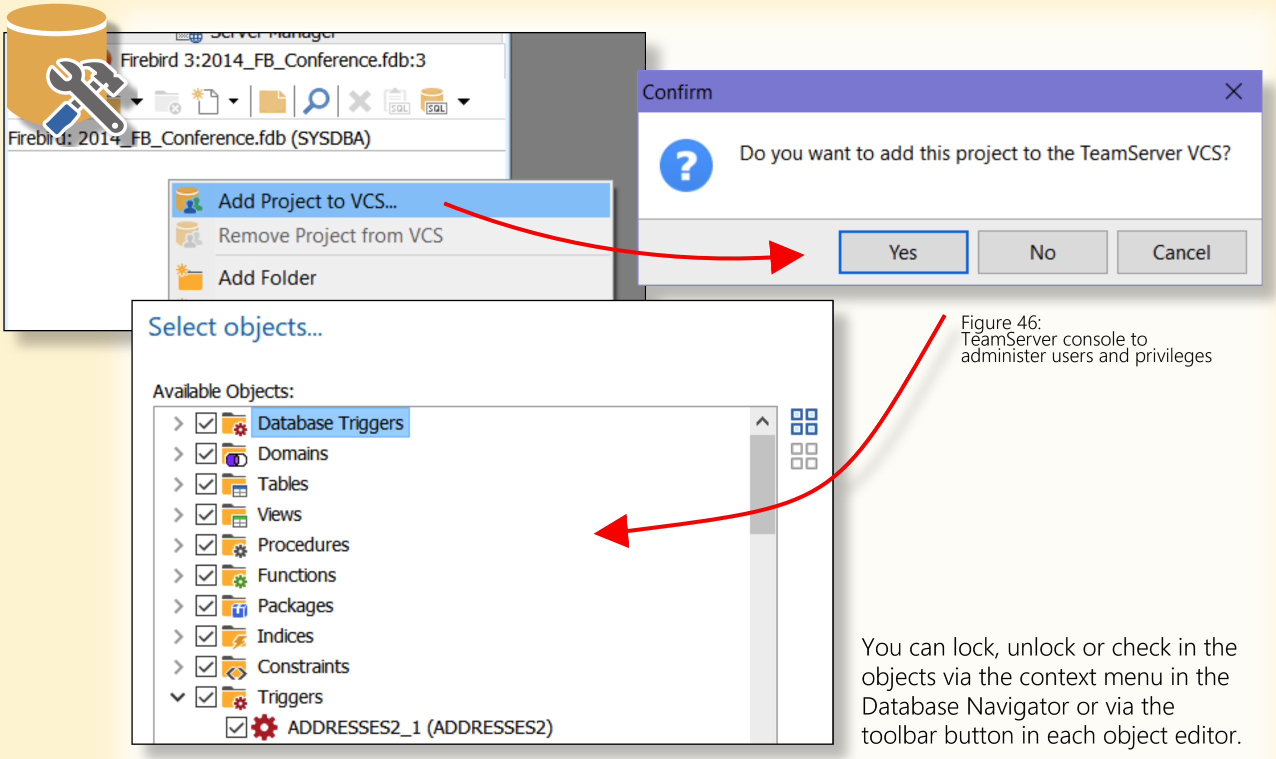
But if you're working with a team of developers, it might be easier to store these in a central repository. **Database Workbench TeamServer** offers this for the **Enterprise** edition in addition to your local repository.

When a **TeamServer** registered databases is opened, the Workspace can contain shared items as well: notes, folders, reports, TO-DO items.

The **TeamServer Console** allows you to create groups of privileges for users and administer the privileges for each user and group. For example, allow users to create public notes in the workspace, register a shared server or database.

You can also control which team members are allowed to create or drop databases for a server, create backups and so on.

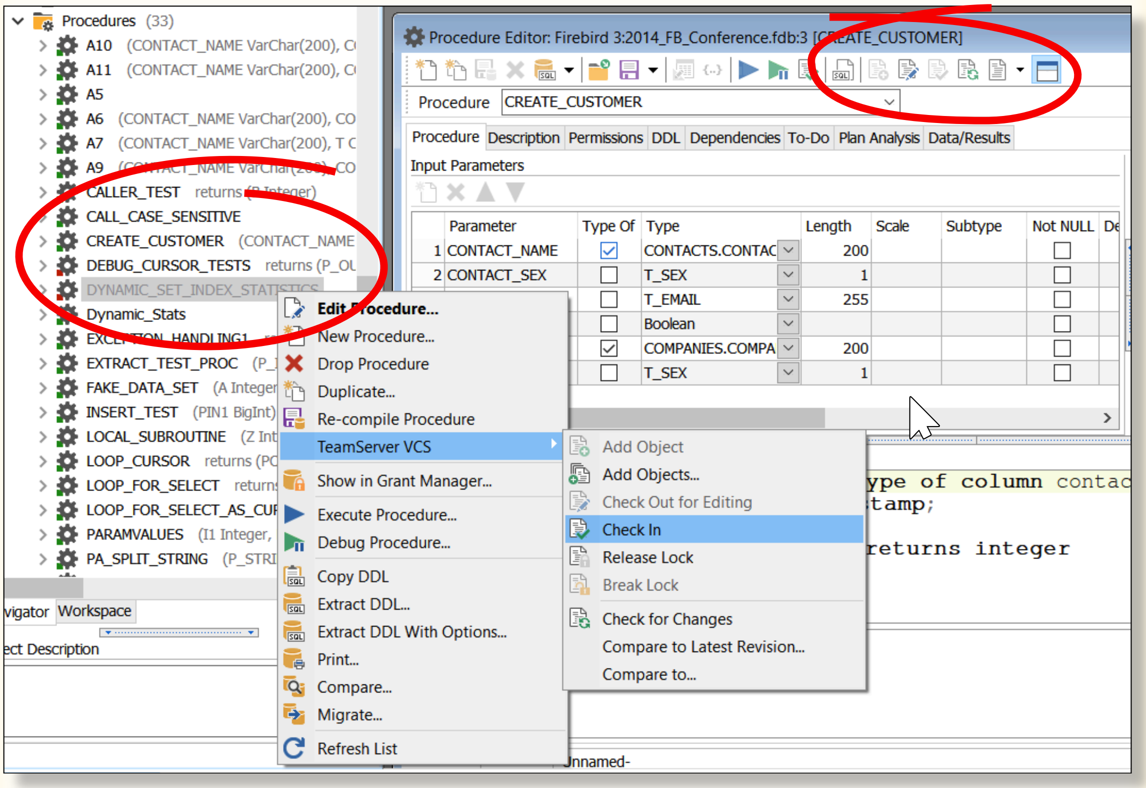
TeamServer also offers a **Version Control System (VCS)** for shared databases. After adding a database to the **VCS**, you can add individual objects. Once added, **Database Workbench** will no longer allow you to modify or drop the object, you first need to lock it for editing.



Do you want to add this project to the TeamServer VCS?

Figure 46:
TeamServer console to administer users and privileges

You can lock, unlock or check in the objects via the context menu in the Database Navigator or via the toolbar button in each object editor.



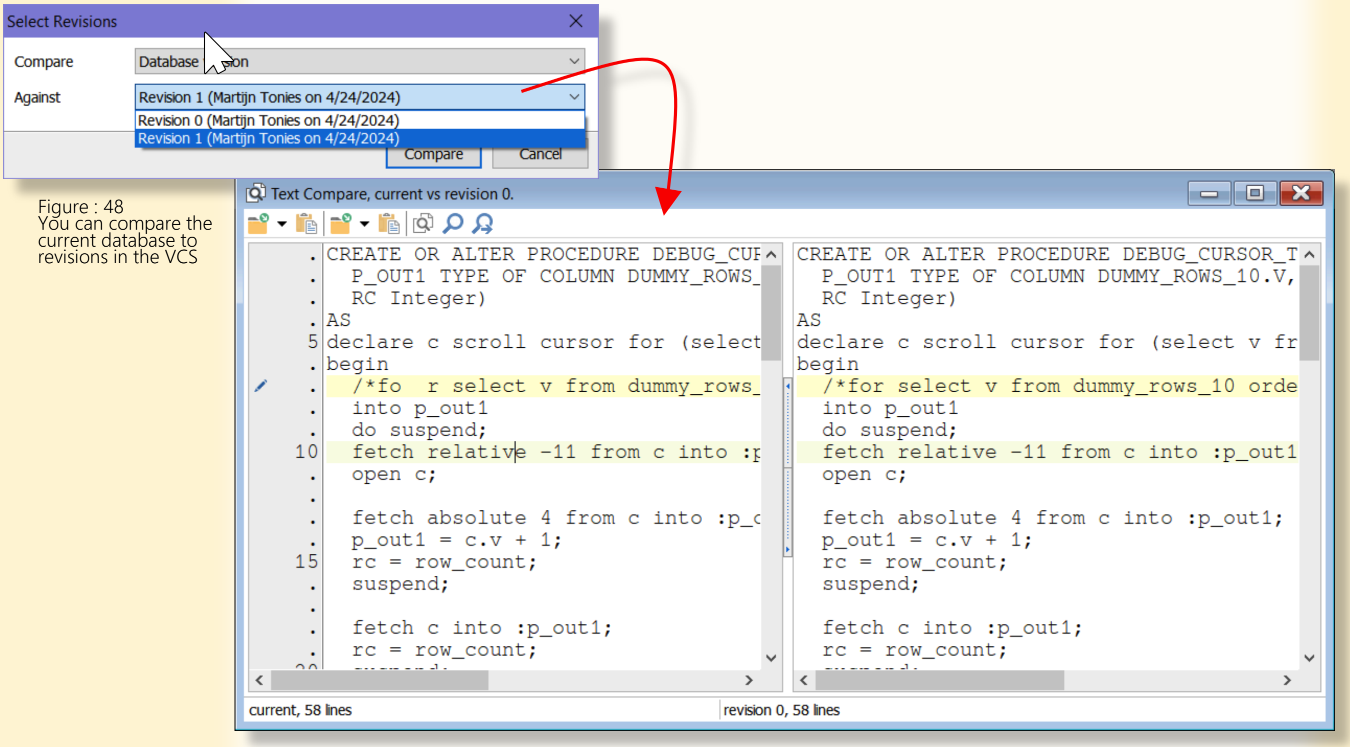
Procedure: CREATE_CUSTOMER

	Parameter	Type Of	Type	Length	Scale	Subtype	Not NULL	De
1	CONTACT_NAME	☑	CONTACTS.CONTAC	200			□	
2	CONTACT_SEX	□	T_SEX	1			□	
		□	T_EMAIL	255			□	
		□	Boolean				□	
		☑	COMPANIES.COMPA	200			□	
		□	T_SEX	1			□	

```
ype of column contac
tamp;

returns integer
```

Figure 47:
Objects in the VCS show a status indicator, editors have VCS related functionality

If you modify an object, **Database Workbenc**h automatically saves the generated **DDL** statements to the **VCS**, as to keep a record of all changes. You can check for changes against the **VCS** or compare between object revisions.



Select Revisions

Compare: Database on

Against: Revision 1 (Martijn Tonies on 4/24/2024)

Revision 0 (Martijn Tonies on 4/24/2024)
Revision 1 (Martijn Tonies on 4/24/2024)

Text Compare, current vs revision 0.

```
CREATE OR ALTER PROCEDURE DEBUG_CUF
  P_OUT1 TYPE OF COLUMN DUMMY_ROWS_
  RC Integer)
AS
declare c scroll cursor for (select
begin
  /*fo  r select v from dummy_rows_
  into p_out1
  do suspend;
  fetch relative -11 from c into :p
  open c;

  fetch absolute 4 from c into :p_c
  p_out1 = c.v + 1;
  rc = row_count;
  suspend;

  fetch c into :p_out1;
  rc = row_count;
```

```
CREATE OR ALTER PROCEDURE DEBUG_CURSOR_T
  P_OUT1 TYPE OF COLUMN DUMMY_ROWS_10.V,
  RC Integer)
AS
declare c scroll cursor for (select v fr
begin
  /*for select v from dummy_rows_10 orde
  into p_out1
  do suspend;
  fetch relative -11 from c into :p_out1
  open c;

  fetch absolute 4 from c into :p_out1;
  p_out1 = c.v + 1;
  rc = row_count;
  suspend;

  fetch c into :p_out1;
  rc = row_count;
```

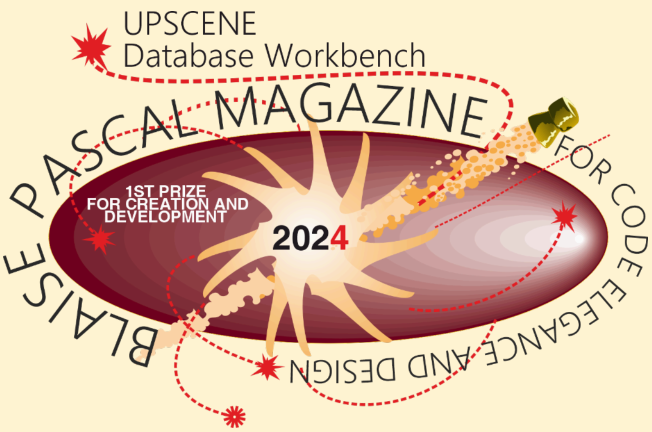
current, 58 lines | revision 0, 58 lines

Figure : 48
You can compare the current database to revisions in the VCS

Each **TeamServer** user or group can have different privileges for the **VCS**, allowing each user to lock objects or not, break an existing lock and so on.

CONCLUSION

this is an exceptional tool for database developers. Its incredible possibilities makes me very enthusiast. Even for simple small databases its is very helpful. I haven't seen for a long time a developer that makes such a good and incredible tool. I think he deserves an award.
It does what I like so much:
The user interface is logically structured and senses where the questions or expectations are for use..



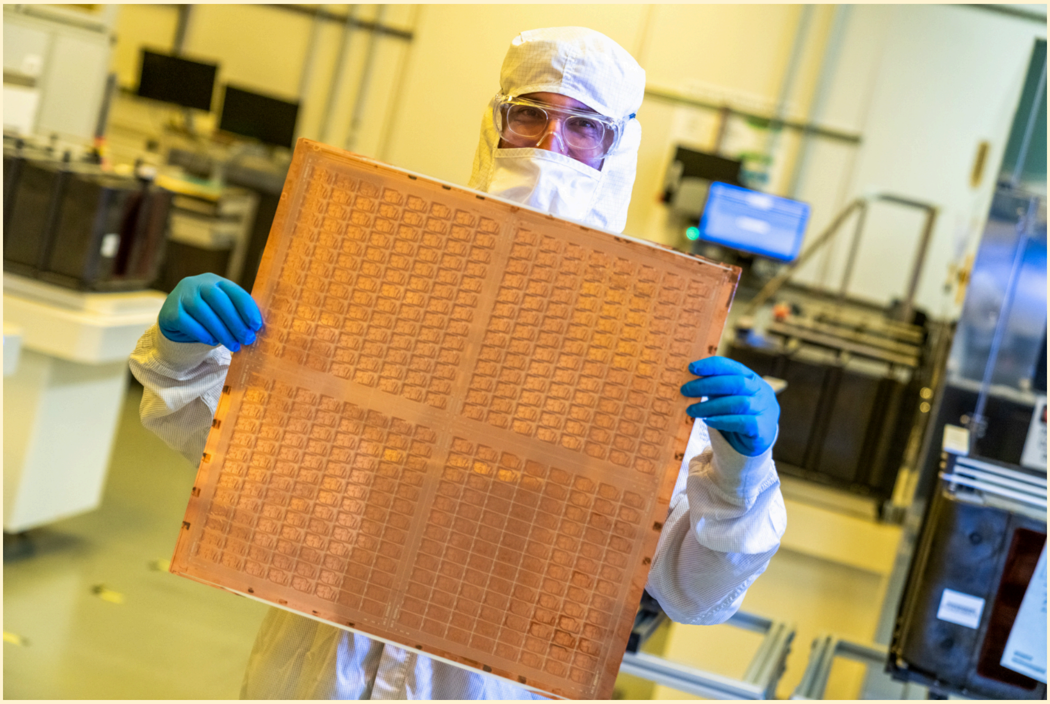
INTEL SEES A FUTURE AS CLEAR AS GLASS

The original article can be found here: https://www.cnet.com/pictures/take-an-early-look-at-intels-glass-packaging-tech-for-faster-chips/

Inside Intel's Chip Factory there is a possible view in to the future:
It looks and is simply ordinary glass.
Intel is transitioning its CPU's to a new architecture in order to meet the rapidly increasing demand for more powerful computing capabilities.
Computer processors are very intricate technological machines. Engineers selectively extract precise combinations of atoms from the periodic table to create materials capable of directing streams of electrons through intricately patterned circuits at extremely fast rates.
However, the next significant advancement in enhancing the efficiency of our laptops and increasing the strength of artificial intelligence may originate from conventional glass.

Intel has provided a comprehensive explanation of the glass technology during its Innovation event in San Jose, California. At a large, technologically advanced structure located in the hot desert landscape of the Phoenix area, Intel converts small tabletop-sized sheets of glass into rectangular sandwiches of circuitry, similar to the procedures used in building processors.
Intel has initiated a lengthy process of transitioning to a new technology that involves placing chips on a glass substrate instead of the current organic resin that resembles epoxy. The newly developed glass foundation, referred to as a substrate, provides the essential speed, power, and space required for the chip industry's transition to a new technology that involves assembling several "chiplets" into a single, larger processor.

Essentially, this implies a novel method to uphold Moore's Law, which measures advancements in packing additional transistor circuitry elements into a CPU.
The **A17 Pro CPU** in Apple's new **iPhone 15** Pro boasts a staggering 19 billion transistors.
The Ponte Vecchio supercomputing processor developed by Intel has a processing capacity exceeding 100 billion. Intel anticipates that by the end of the decade, computers will have a staggering one trillion transistors.
Intel adopted the chiplet strategy to narrow the gap with competitors who had better capabilities in CPU manufacture.
According to Creative Strategies analyst Ben Bajarin, Intel can now utilise this technology to exceed its competitors in a time when there is a high need for increased processing power that the industry is struggling to meet. Intel's glass substrate technology showcases their expertise in packaging.

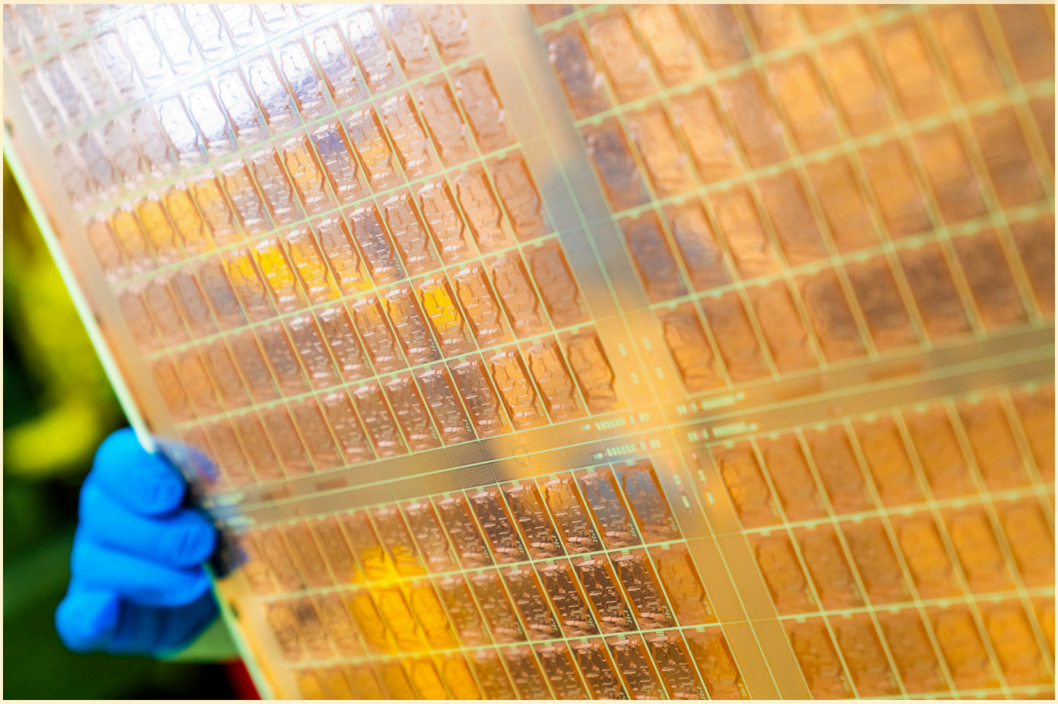


In the future, you can expect more advanced computers and AI technologies that are significantly more intelligent than the ones currently available. The semiconductor industry will transition to glass substrates due to many reasons. The whole chip industry, particularly high-end CPUs, will undergo the glass transition in order to address the problems of chipmaking, (with Intel leading?)

Through extensive collaboration with academics and rigorous testing of innovative techniques over a period of more than ten years, a team of 600 employees based in Chandler has successfully transformed research and development into an operational manufacturing process. "The innovation is complete," stated Ann Kelleher, the executive vice president overseeing technology development at Intel. The utilisation of glass substrate technology provides us with the capability to achieve superior performance for our products in the long run.
These are confident statements from a business that is currently in the middle of a four-year endeavour to regain the dominance it lost to Taiwan Semiconductor Manufacturing Co. and Samsung, who are chip "foundries" responsible for manufacturing processors for several electronics companies. Intel's production progress saw a significant slowdown for a period of several years, beginning almost ten years ago. As a result, it relinquished its previously dominant position to the two chipmakers from Asia.

The glass technology integrated beneath a processor is expected to be introduced in the latter half of the decade. Initially, it will be implemented in the largest and most energy-intensive chips, which are utilised in numerous servers housed in data centres operated by major hyperscale companies such as Google, Amazon, Microsoft, and Meta.

The new substrate has the capacity to handle tenfold the power and data connections compared to current organic substrates, enabling a higher volume of data transfer to and from a chip.
Minimising warping is crucial for ensuring that processors remain flat and effectively link to the external environment, allowing for the use of chip packages that are 50% larger.
It effectively transfers power, allowing semiconductors to operate at higher speeds or with greater efficiency.
Furthermore, it has the capability to operate at elevated temperatures, and when it undergoes thermal expansion, it maintains a consistent rate of expansion with silicon to prevent any potential mechanical malfunctions.



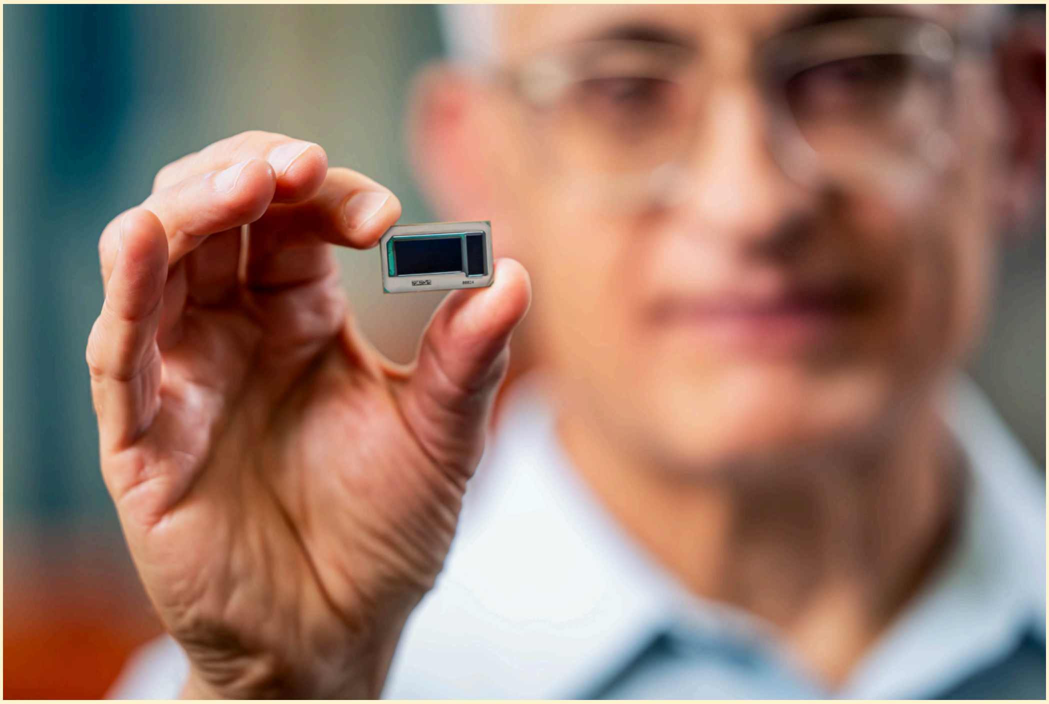
Glass will facilitate the development of next-generation server and data centre processors, which will replace large processors such as Intel Xeons. These processors will be capable of running cloud computing services, such as email and online banking, as well as Nvidia's highly sought-after artificial intelligence processors, which have gained immense popularity due to the widespread adoption of generative AI.

However, if glass substrates reach a more advanced stage of development and become more affordable, this technology will extend beyond data centres to personal computers. It is evident that Intel anticipates this technology being integrated into client applications. referring to personal computers.

The chipmaking industry is expected to experience a comeback with the development of five nodes within a span of four years. Intel, under the guidance of Chief Executive Pat Gelsinger, who returned to the company in 2021, is making efforts to regain its position at the forefront of the industry. During each press conference and quarterly earnings call, Intel executives repeatedly emphasise the goal of achieving "five nodes in four years."
This refers to the ambitious plan to rapidly progress through five significant chip manufacturing advancements in order to catch up with and ultimately surpass TSMC and Samsung by 2025. Kelleher, who is leading the project, states that two of the processes have been finished and the remaining steps are progressing according to the planned timeline.

AN ALL-INCLUSIVE BUNDLE

Even if Intel manages to regain its advantage in the lithography production process, which involves imprinting transistors into a silicon surface, the business and its competitors still confront a significant challenge: constructing the housing that connects these chips to a circuit board. Packaging and glass substrates play a crucial role in this context.
The Intel 8086 chip, which was developed in 1978, served as the foundation for all subsequent PC and server processors manufactured by Intel. It consisted of a flat square of silicon with 29,000 transistors. In order to safeguard and connect it to a circuit board, the device was enclosed in a packaging that resembled a flat caterpillar. The device was powered and received data by forty metal legs.



Subsequently, there has been a significant advancement in CPU packaging. The distinction between chipmaking and packaging, which used to be quite basic, is now becoming less clear. Currently, packaging methods employ lithography equipment to engrave their own circuitry, but with less precision compared to CPUs.
Over time, processors have evolved from having caterpillar-like legs to having hundreds of pins resembling a small bed of nails that cover the bottom of the processor. However, in the end, that method proved to be insufficient in providing an adequate number of electrical connections to the circuit board.
Today's packages are equipped with flat metal contact patches located on the bottom of the package. The chip is affixed to the circuit board with significant pressure, amounting to hundreds of pounds, during installation.
A metallic cap positioned on top of a processor effectively dissipates excess heat that would otherwise cause a computer to malfunction. Below the processor lies a substrate featuring a progressively intricate, three-dimensional network of power and data connections that serve to connect the chip to the external environment.

THE HIDDEN DEPTHS

Transitioning from current organic substrates to glass presents many problems. Glass is fragile, hence it necessitates cautious handling, for instance.
In order to facilitate the transition, Intel is modifying glass-handling equipment obtained from professionals in the display industry, who possess the knowledge and expertise to handle glass without causing any damage. The display sector is responsible for manufacturing a wide range of products, including small wristwatch screens and large flat-panel TVs. In addition, they are required to engrave circuitry onto glass and have successfully created numerous essential ultrapure materials and meticulous handling procedures.

CHALLENGES RELATED TO MICROCHIPS

❶ Intel's strategy to recover its chip manufacturing capabilities might potentially revive the United States' manufacturing strength.

❷ The practice of stacking chips in a layered manner, similar to pancakes, has the potential to reduce the cost of laptops.

❸ Intel gained valuable insights when one of its supercomputer chips was damaged by an elevator collision.

Intel is planning to build a 'Megafab' that may potentially become the largest chip manufacturing plant in the world. The project is estimated to cost around $100 billion.
However, there are distinctions.

Flat-panel displays use electronic components that are sensitive and located exclusively on one side, allowing glass to smoothly move through factories using rollers. Intel constructs a structure consisting of various materials and circuitry, known as redistribution layers, on both surfaces of the glass. Consequently, their computers are required to securely grip the glass solely at its edges.
Every panel is meticulously removed from the container, inserted into the machine, rotated into a vertical position, and then inserted further to allow for additional layers to be added to the sandwich.

Like all other products at Intel, it is specifically built for large-scale production rather than small-scale research and development initiatives.

INTEL FOUNDRY SERVICES IS ENHANCED BY PACKAGING.

The utilisation of Intel packaging technology is expected to be beneficial for its proprietary data centre processors. It is crucial for Intel's planned corporate restructuring to include becoming a chip foundry, similar to TSMC and Samsung, where it manufactures processors for other firms under its Intel Foundry Services subsidiary.

Intel can offer packaging services, regardless of whether it manufactures the chips and chiplets that are included in the box. However, this can subsequently result in a more extensive customer agreement, wherein Intel fabrication facilities, sometimes known as "fabs," are responsible for constructing both the silicon processor chips and chiplets.

"At IFS, Mark Gardner, the senior director in charge of the Foundry Advanced Packaging group, stated that we possess both competitiveness and capacity." According to him, it is relatively simpler to get a packaging customer compared to a chipmaking customer, as there are fewer technological complexities and shorter timeframes for completion.

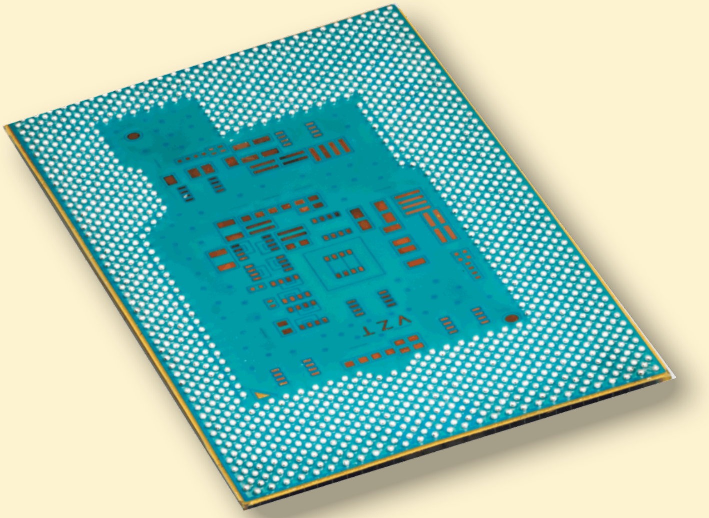
However, when it comes to customer agreements for packaging, it can result in a more profound partnership that extends beyond chipmaking. Specifically, Intel anticipates that its 18A chipmaking process will overtake TSMC and Samsung by 2024.

"It represents an opportunity to establish a connection or gain entry," Gardner stated. "The trajectory of packaging first and advanced packaging then 18A is working well for a specific customer."

The M2 Ultra, which marks the culmination of Apple's two-year shift away from Intel CPUs, is accompanied by two M2 Max chips that have a high-speed interconnect. AMD has increased its market share at the expense of Intel by utilising TSMC and Samsung to manufacture its designs, particularly server chips that incorporate several chiplets.

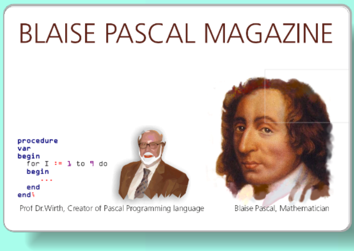
The extent to which the processor business will transition from "monolithic," single-die designs to chiplet designs remains uncertain. There are still benefits in terms of cost and simplicity when one chooses to eschew complex packaging. However, it is evident that the most significant processors, specifically the server and artificial intelligence (AI) processors located in data centres, will evolve into extensive networks of interconnected chiplets.

Glass substrates are useful in providing chip designers with ample space, communication linkages, and power delivery capabilities, allowing for future expansion.



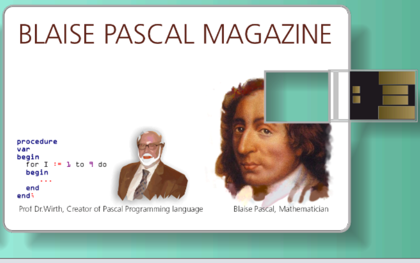
LIB-STICK ON USB CREDIT CARD BLAISE PASCAL MAGAZINE

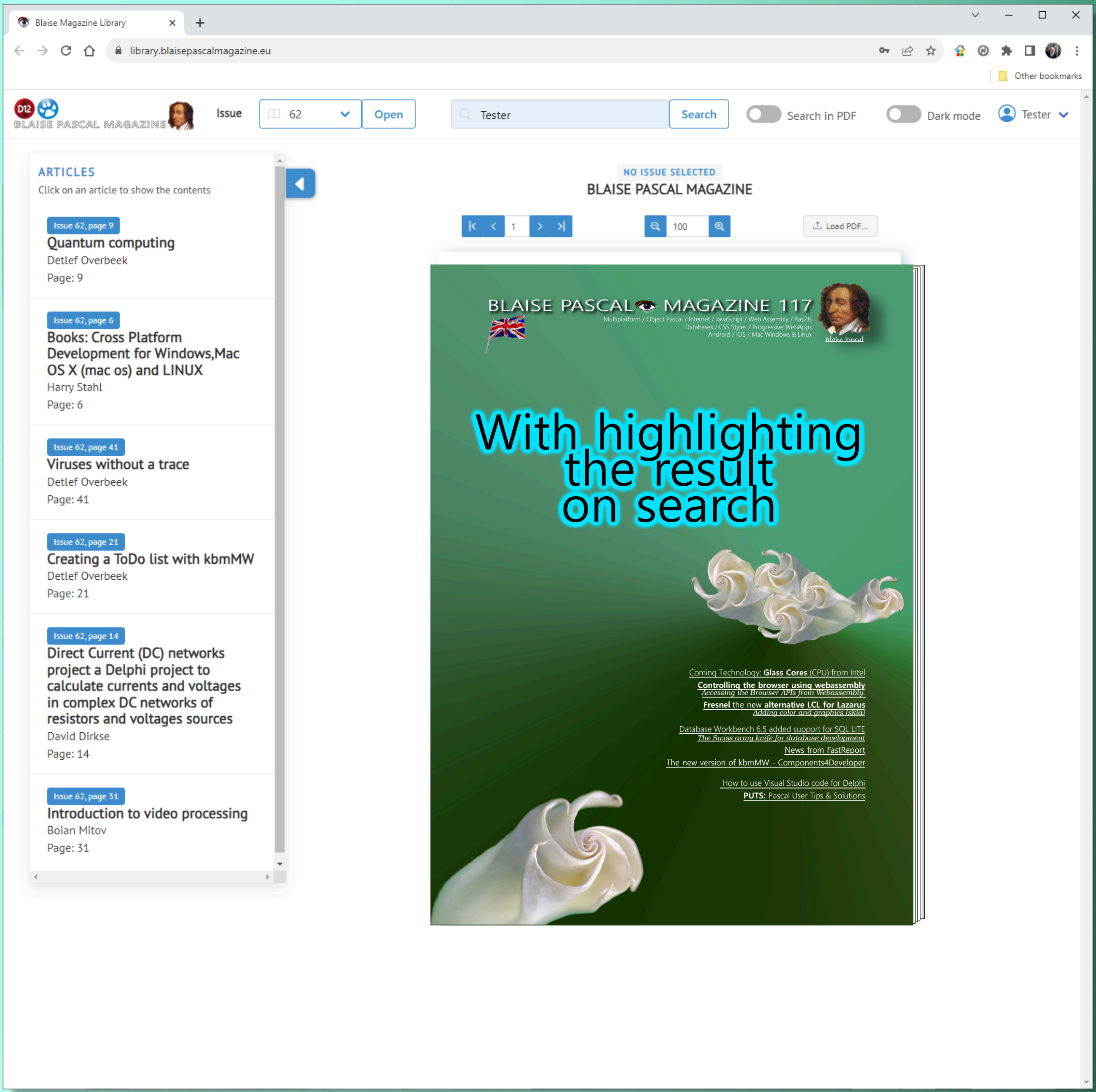
LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY € 100


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

Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
IJsselstein Netherlands

editor@blaisepascalmagazine.eu
https://www.blaisepascalmagazine.eu


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



ARTICLES

Click on an article to show the contents

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

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

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

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

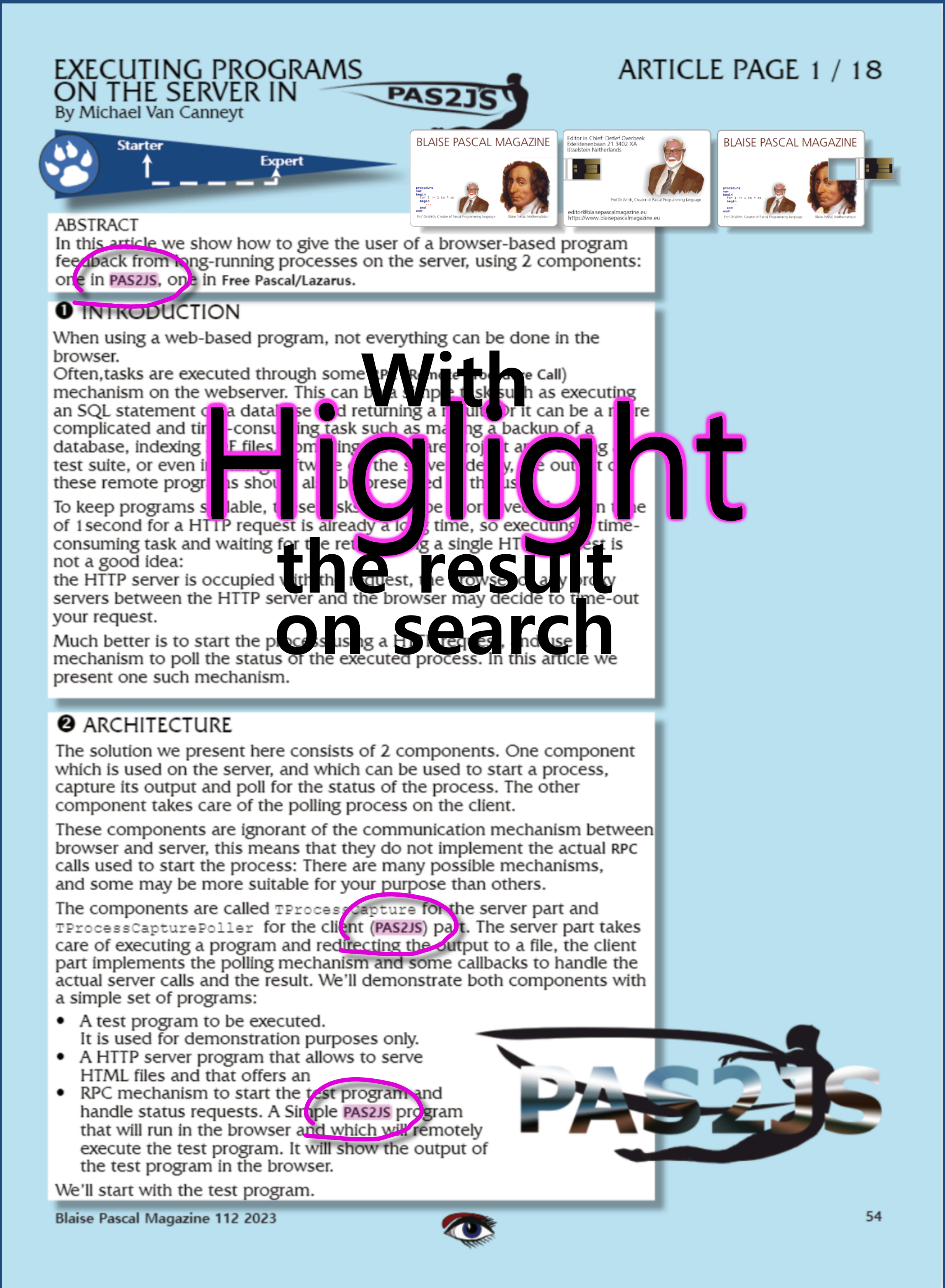
Issue 62, page 14
Direct Current (DC) networks project a Delphi project to calculate currents and voltages in complex DC networks of resistors and voltages sources
David Dirkse
Page: 14

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

NO ISSUE SELECTED
BLAISE PASCAL MAGAZINE

BLAISE PASCAL MAGAZINE 117
Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux
Blaise Pascal
With highlighting the result on search
Coming Technology: Glass Cores (CPU) from Intel
Controlling the browser using webassembly
Accessing the Browser APIs from Webassembly.
Fresnel the new alternative LCL for Lazarus
Adding color and graphics (skia)
Database Workbench 6.5 added support for SQL LITE
The Swiss army knife for database development
News from FastReport
The new version of kbmMW - Components4Developer
How to use Visual Studio code for Delphi
PUTS: Pascal User Tips & Solutions

LIB-STICK BLAISE PASCAL MAGAZINE



EXECUTING PROGRAMS ON THE SERVER IN PAS2JS

ARTICLE PAGE 1 / 18

By Michael Van Canneyt

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

❶ INTRODUCTION

When using a web-based program, not everything can be done in the browser.
Often,tasks are executed through some RPC (Remote Procedure Call) mechanism on the webserver. This can be a simple task such as executing an SQL statement on a database and returning a result. Or it can be a more complicated and time-consuming task such as making a backup of a database, indexing PDF files, compiling a software project and running a test suite, or even installing software on the server. [illegible] the output of these remote programs should also be presented to the user.

To keep programs scalable, these tasks [illegible] the [illegible] of 1second for a HTTP request is already a long time, so executing a time-consuming task and waiting for the result during a single HTTP request is not a good idea:
the HTTP server is occupied with the request, the browser or any proxy servers between the HTTP server and the browser may decide to time-out your request.

Much better is to start the process using a HTTP request, and use a mechanism to poll the status of the executed process. In this article we present one such mechanism.

❷ ARCHITECTURE

The solution we present here consists of 2 components. One component which is used on the server, and which can be used to start a process, capture its output and poll for the status of the process. The other component takes care of the polling process on the client.

These components are ignorant of the communication mechanism between browser and server, this means that they do not implement the actual RPC calls used to start the process: There are many possible mechanisms, and some may be more suitable for your purpose than others.

The components are called `TProcessCapture` for the server part and `TProcessCapturePoller` for the client (PAS2JS) part. The server part takes care of executing a program and redirecting the output to a file, the client part implements the polling mechanism and some callbacks to handle the actual server calls and the result. We'll demonstrate both components with a simple set of programs:

- A test program to be executed. It is used for demonstration purposes only.
- A HTTP server program that allows to serve HTML files and that offers an
- RPC mechanism to start the test program and handle status requests. A Simple PAS2JS program that will run in the browser and which will remotely execute the test program. It will show the output of the test program in the browser.

We'll start with the test program.

LAZARUS HANDBOOK 2
LAZARUS HANDBOOK 1
LAZARUS HANDBOOK
FOR PROGRAMMING WITH FREE PASCAL
1

BLAISE PASCAL MAGAZINE 117
IT FINALLY HAPPENS:
WITH FRESNEL YOU CAN CREATE
A UNIVERSAL GRAPHICAL APPLICATION
RUNNING ON ALL NATIVE PLATFORMS AND IN THE BROWSER

ADVERTISEMENT

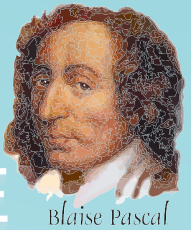
LAZARUS HANDBOOK PRINTED +SUBSCRIPTION 1 YEAR

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

SPECIAL OFFER € 75 +SHIPPING

LAZARUS HANDBOOK 2
LAZARUS HANDBOOK 1
LAZARUS HANDBOOK
FOR PROGRAMMING WITH FREE PASCAL
1
+
BLAISE PASCAL MAGAZINE 117
Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux
Blaise Pascal
IT FINALLY HAPPENS:
WITH FRESNEL YOU CAN CREATE
A UNIVERSAL GRAPHICAL APPLICATION
RUNNING ON ALL NATIVE PLATFORMS AND IN THE BROWSER

FRESNEL


Blaise Pascal

barnsten

GOLD
SPONSOR
AT

Delphi Summit
JUN 13-14 2024 | AMSTERDAM

Barnsten proudly announces its gold sponsorship at this year's Delphi Summit, where the world's top Delphi developers gather for inspiration, knowledge exchange, and networking.

We're thrilled to connect with Delphi enthusiasts from around the globe, eager to share their stories of success and overcome challenges.
As Embarcadero software vendors for over 25 years, our expertise is at your service. Whether you seek advice or solutions for your development needs, our dedicated Barnsten team is here to empower your projects.

Visit our booth to explore exclusive offers and discounts from renowned component vendors like Fast Reports, Steema TeeChart, Gnostice, TMS, Devart, Woll2Woll, and more!

Let's make this summit an unforgettable experience together!

Warm regards,
The Barnsten Team

Delphi Summit

JUN 13-14 2024 | AMSTERDAM

embarcadero®
Official Technology Partner

Delphi Summit 2024

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

300 **Attendees**
meeting fellow developers

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

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

THE DELPHI SUMMIT IN A NUTSHELL

About the summit.

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

LOCATION AND SPEAKERS

Meet and great

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

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

Delphi Summit 2024

Agenda

Amsterdam. June 13-14, 2024

Day 1: Thursday 13th

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

Day 2: Friday 14th

09:00	**Registration**		
	Mainstage		
10:00	**Welcome and opening – Kees de Kraker and Marco Geuze**		
10:15	**Panel discussion with Jim McKeeth, Marco Cantu, Ian Barker and MVPs**		
11:00	Coffee Break & Network		
	Stage Sydney	**Stage Alexandria**	**Stage Rio**
11:30	**Ray Konopka** - Component Building: Fundamentals This session focuses on the fundamental techniques required for building robust Delphi components. We build a custom component, showing the key classes from which all components descend, followed by an analysis of the anatomy of a component. We conclude with a discussion on the proper way to distribute custom components through runtime and design packages.	**Patrick Quist** – Linux Delphi Services A journey through the Cloud(s)	**Stefan Glienke** – Spring4D Some goodies from the Spring4D collections.
12:30	Lunch, Network & Gaming		
13:30	**Olaf Monien - REST Easy** Connecting to REST APIs and visualizing data on desktop and mobile devices.	**Christoph Schneider** – Firestone Cloud For the Firestore Cloud database, the FB4D open-source library contains everything you need to access it from VCL/FMX applications. In this session, the author will show you how easy it is to write and read a document and to be notified of changes in the database with the new object-to-document wrapper.	**Patrick Prémartin** – Synchronize your databases Our users want to access their data from anywhere, on any type of device, with or without an Internet connection. Some also want to work together offline or online, remotely or on-site, on desktops, laptops, smartphones or tablets. Here's an easy-to-implement solution in Delphi to transform any local database into a synchronized one
14:30	**Carlos Agnes** - The Best of Delphi Underground A set of small Delphi secrets and how they work under the hood. IDE and debugging tips, historical issues like why the base date for TDateTime is 12/30/1899, Exceptions stacks, interface tricks, and the dictionary of secrets.	**Andrea Raimondi** - Algorithmic password hardening From the forgotten lessons of Enigma to generating salts and scrambling passwords, Andrea will guide you through the best ways to keep everything safe.	**Bruno Fierens** – Build a full-stack application within an hour In this session, you'll discover how to leverage a new and innovative approach to build web client applications using TMS WEB Core as well as native Delphi applications on desktop or mobile platforms that work with backend data.
15:30	Break & Network		
15:45	**Barnsten** - License Management, support and subscription Barnsten will inform you about the different Embarcadero licence types that are available. The subscription is also discussed. What is covered by the contract and how can Barnsten help you with your licensing questions. Such as: licence transfer to another user, what about previous versions, how to log a feature request., bug or support case etc. And there will be room for questions after the presentation.	**NexusDB** - Implementing NP-C and NP-Hard Algorithms In this session, we'll delve into the complexities of designing algorithms for NP-Complete and NP-Hard problems. Using the Eternity II puzzle and commercial scheduling software as case studies, we'll discuss why these problems remain unsolved, explore practical algorithmic solutions, and highlight the role of modern hardware and Delphi as the IDE. Gain insights into the impact and practical handling of these problems in real-world	

		programming.	
	Mainstage		
16:15	**Ian Barker -** What to do if you're old, ugly, and everything is annoying Join Ian for this session where he applies his uniquely lively style of presentation to the subject of software development in an age where everyone wants your apps to be free, have a name like ZZxQFImbl, and be 'monetized' by a YouTube influencer with green hair, a pierced fingernail, and their own brand of hair removal creme.		
16:45	**Door prize giveaway**		
17:00	**Closing talk with Jim McKeeth, Kees de Kraker and Marco Geuze**		
17:15	Network & Gaming		
18:00	Conference ends		

FPC/LAZARUS FRESNEL

Now fpc/lazarus using fresnel
Has three working backends,
A css-driven layout,
Multiple platforms,
A powerful event mechanism.

We now can:

CREATE A UNIVERSAL GRAPHICAL APPLICATION RUNNING ON ALL NATIVE PLATFORMS AND IN THE BROWSER.

All this using a single codebase,
and running at native speed.
And obviously,
all this using your favourite Programming language:

OBJECT PASCAL.

THE LAZARUS DEBUGGER

BY MARTIN FRIEBE

PART 7: Change happens – Waiting for it

Starter Expert

FOLLOW THE DATA, RATHER THAN THE CODE

So far all our debugging methods had one thing in common. We would pause the app and single step or run to the next breakpoint. And on each pause we would check if our data matched our expectations. The more code our application has, the more stepping we may have to do. In some cases, it can even be hard to tell where to pause for the single stepping.

Now we will have a look, how the debugger can help us finding the right spot. Of course in the scope of this article we can't have a sample project so big that we actually couldn't solve it by stepping. We'll just have to pretend, and also the feature in question works well with any size of app.

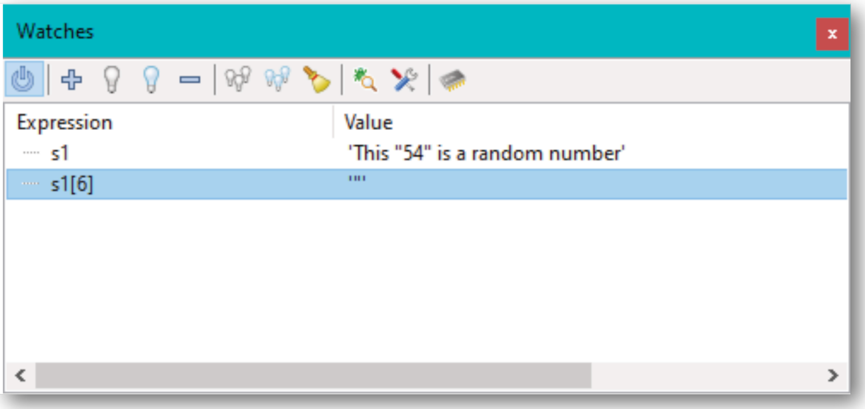
```
program project1;
{$Mode objfpc}{$H+}

uses SysUtils;

function ChangeQuotes(const ASource: String; var ADest: String): Boolean;
var
 Len: SizeInt;
 Src, Dst: PChar;
begin
  Result := False;
  Len := Length(ASource);
  SetLength(ADest, Len);
  Src := PChar(ASource);
  Dst := PChar(ADest);
  while Len > 0 do begin
    Dst^ := Src^;
    if Dst^ ='"' then begin
      Src^ := '''';
      Result := True;
    end;
    inc(Src);
    inc(Dst);
    dec(Len);
  end;
end;


var
  s1, s2: String;
begin
  s1 := 'This "'+IntToStr(Random(99))+'" is a random number';
  writeln('Initial value: ', s1);
  if ChangeQuotes(s1,s2) then begin
    writeln('With double quotes: ', s1);
    writeln('With single quotes: ', s2);
  end
  else
    WriteLn('Nothing changed');
end.
```

The code is simple enough. We have a text containing double quotes, the function replaces them with single quotes, and returns true. The 2 versions of the text will be printed.

And as in previous articles the output does not match what we expect:

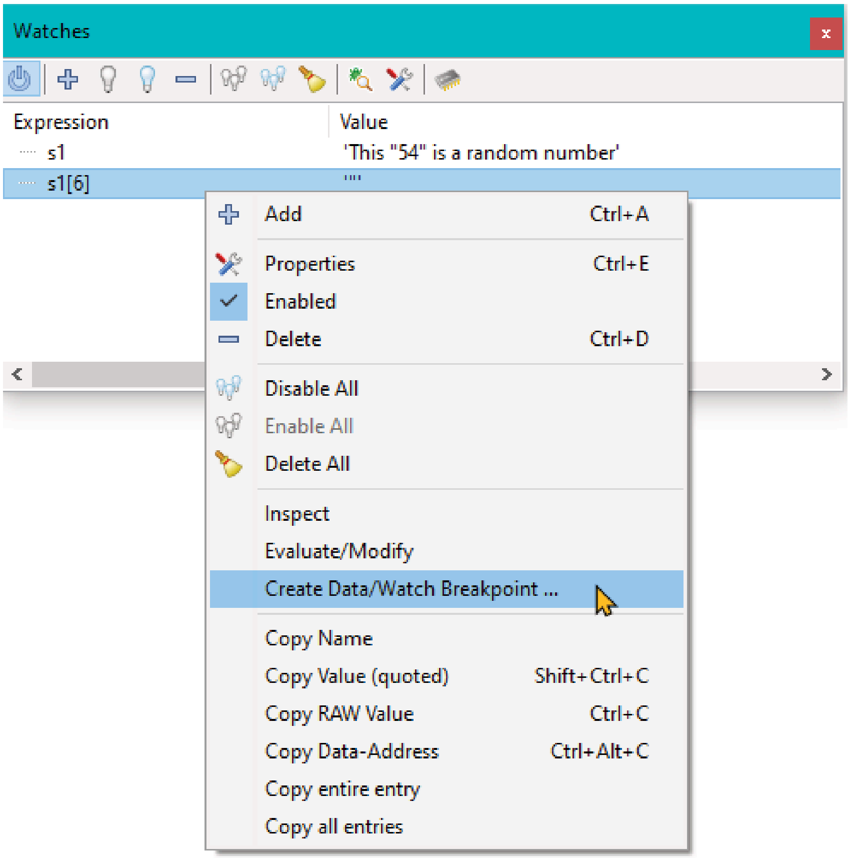
```
Initial value: This "54" is a random number
With double quotes: This '54' is a random number
With single quotes: This "54" is a random number
```

Of the 2 last lines, each line has the quotes that should be in the other line.
Well, we need to start the app, and so we need to run to a breakpoint. Let’s make that on line 33.
We know from the writeln that the value of s1 is ok on that line.



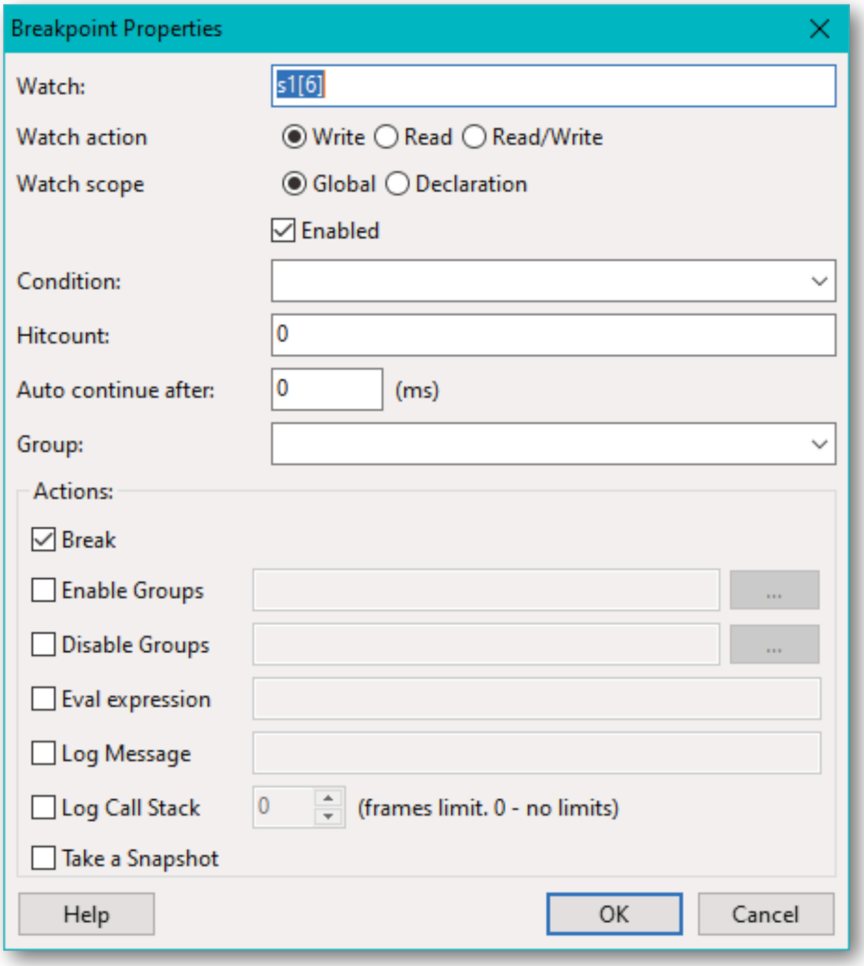
Expression	Value
s1	'This "54" is a random number'
s1[6]	'"'

As you can see, I also added `"s1[6]"` to the list of watches. This is one of the double quotes in the initial string. That double quote is not supposed to change, yet from the 2nd line of output we already know it does get changed.
From here we will ask the debugger to do the work, and find the code that makes this change.
The command that does this can be found in the **context menu** on the **watch “s1[6]”**.



Expression	Value
s1	'This "54" is a random number'
s1[6]	'"'

“**Create Data/Watch Breakpoint**” will open a new dialog. That dialog is also available from the “**Run**” menu **“Add Breakpoint”** → **“Data/Watch Breakpoint”** and from the breakpoint dialogs **“Add”** button’s dropdown.



Breakpoint Properties

Watch: s1[6]

Watch action ● Write ○ Read ○ Read/Write

Watch scope ● Global ○ Declaration

☑ Enabled

Condition:

Hitcount: 0

Auto continue after: 0 (ms)

Group:

Actions:

☑ Break

□ Enable Groups

□ Disable Groups

□ Eval expression

□ Log Message

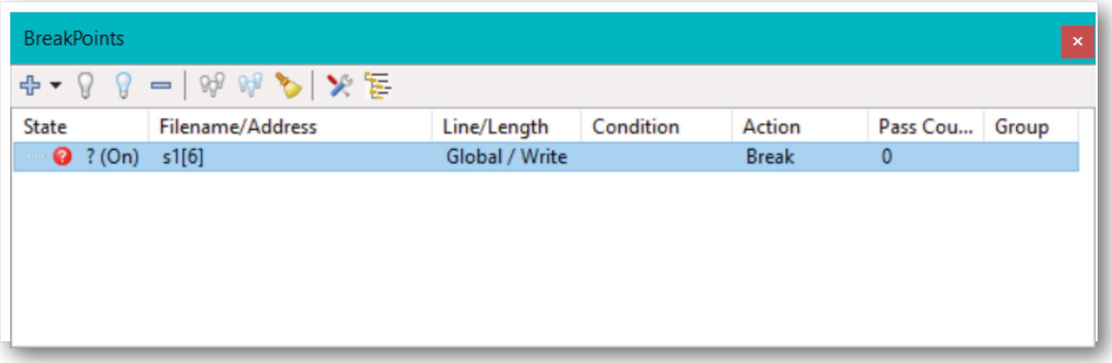
□ Log Call Stack 0 (frames limit. 0 - no limits)

□ Take a Snapshot

In this case all the options have been pre-filled, and we can keep them as they are.
We want the debugger to keep track of changes *(“write” action)* to the value `"s1[6]"`.
And when the value changes the debugger should pause the app and tell us which line it happened.

We will go through the other details, once we have seen the feature in action.

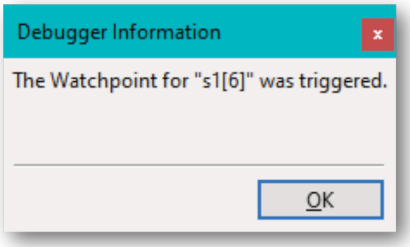
Pressing **“OK”** will create a **“Watchpoint”** or **“Data-breakpoint”**. Watchpoints are listed with other breakpoints in the breakpoint window.



BreakPoints

State	Filename/Address	Line/Length	Condition	Action	Pass Cou...	Group
? (On)	s1[6]	Global / Write		Break	0	

All that remains for us to do is to run (**F9**) the application. And as we do, the debugger will right away inform us:



Debugger Information

The Watchpoint for "s1[6]" was triggered.

The debugger will show us the project paused on line 20

```
      Result := True;
```

We have to keep in mind, that in order to know the statement that changed the value, the debugger must have executed it already. So it will show us the next statement below the one that made the change.
The offending statement therefore is at line **19**

```
      Src^ := '''';
```

And indeed that clearly is a mistake. We shouldn't assign a value to the source.

Instead of

```
    if Dst^ ='"' then begin
      Src^ := '''';
```

it should be

```
    if Src^ ='"' then begin
      Dst^ := '''';
```

As promised the debugger has found the offending line for us. We can fix the bug, and our project correctly outputs:
Initial value: This **"54"** is a random number
With double quotes: This **"54"** is a random number
With single quotes: This **'54'** is a random number

FINE TUNING – THE SETTINGS

For the above debug session we have just filled in the watched expression and left all settings at their defaults. Most of the settings, are the same as for breakpoints and those have been described in the last article.

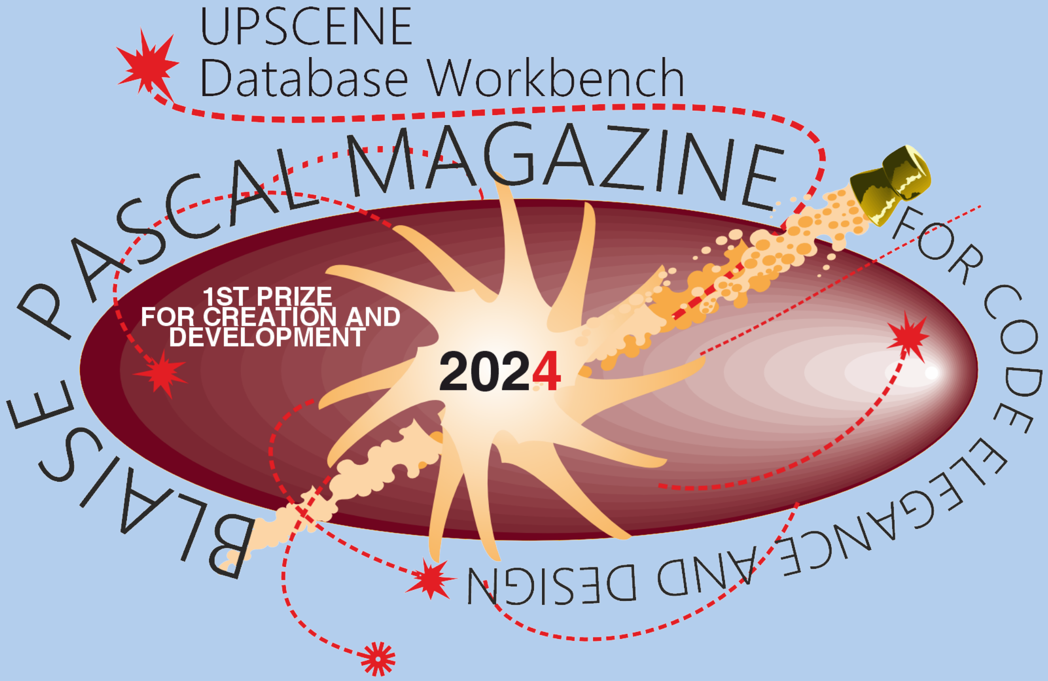
However we do have **"Watch action"** with the options **"Read", "Write"** and **"Read/Write"**. **"Write"** is by far the most common action. It tells the debugger to react only if the application writes data to the memory of the variable. That is any data written, even if the new data is identical to the existing data, which means the value does not change.
"Read" on the other hand tells us when the application reads the memory. That may be useful if we have a variable that the app is not supposed to use, but maybe does access via a pointer.
Then the app may not write, but just use the value. And **"Read"** will catch that. In most cases like this it would be better to use the **"Read/Write"** option. Since if the app shouldn't access the value, then it should do neither **read**, nor **write**.
The other option that we need to look at is **"Watch Scope"**. This is currently not supported by all debuggers. The **GDB** backend does support this, but the **FpDebug** backend does not.
This can be of use, if a **watchpoint** is set on a local variable. Once the procedure owning the variable returns, the memory is no longer reserved for the variable, and another procedure will eventually put a different variable there. Since **watchpoints** work on the memory where the variable is/was, they will then be triggered by changes or access to the new variable using the same memory. Setting **"Scope"** to **"Local"** means the **GDB** based debugger will notice when the procedure is exited, tell you about it and clear the **watchpoint**.

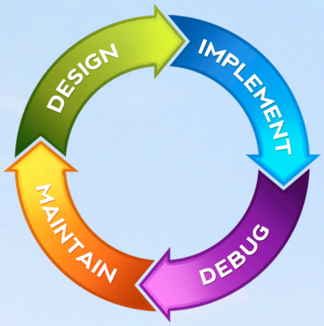
FINE-PRINT – ABOUT "THE DATA"

So here we go, there are a few caveats. To start with, what **`watchpoints`** can and can't do is actually determined by your hardware. The following applies to what you get, if you have a modern Intel or AMD CPU. With **FpDebug** you are currently limited to those anyway, with **GDB** you can debug code for **other target CPUs,** but then may get different availability of **`watchpoints`**.
There are a maximum of 4 **`watchpoints`** available, and each of them can cover up to a pointer-size area of memory. Actually, that is 1, 2 or 4 bytes on 32bit machines, and 1, 2, 4 or 8 on 64bit machines.
For that reason in the above example we could not have watched the entire text of the string. Its length exceeded the 8 bytes by far. Instead we picked a single character. However, there is a 2nd caveat hidden here. It is possible to add a string (**`Ansistring`**) as **`watchpoint`**. **`Ansistrings`** internally have a pointer to the text. And a **`watchpoint`** would then act on that pointer. Predicting what a **`watchpoint`** would react to in this case, requires some knowledge of the internal workings of **`Ansistrings`**.
Similar traps lay ahead with objects. Here too, variables contain a pointer to the instance. And so the same rules apply. However, in some cases this may be useful, namely if we want to know when a variable is changed to point to a different instance: **`"MyVar := SomeOtherObj;".`** This assignment changes the pointer in **`"MyVar"`**.
Watchpoints can be used for **`ordinal`** types such as **`numbers, booleans, enums`**. And they can be used for small sets too. And while it is often not possible to monitor an entire object or **`record`**, **individual fields can be monitored, just like individual chars in a string can be**.

FINE LINE – WHERE EXACTLY DOES THE APP PAUSE

One small note to end the article. **Watchpoints** aren't always exact. Well they are in terms of **`assembler`**. They will always stop at the **`asm`** instruction right after the one that makes the change. That is to say, when they stop, the **change has just been made.**
In the **Pascal** source that does not always mean the next line of Pascal code. The change could happen in the middle of the **`asm`** instructions belonging to a line of **Pascal** code, and then the debugger shows you the line that makes the change. If however it happens at the very end of the **`asm`** belonging to that line, then the debugger will show you the next line of **Pascal** code. So you may have to look one line further up.
`Watchpoints` are only valid during the debug session in which they were created. **When you start the debugger the next time, addresses may have changed,** and the debugger will not activate any existing **`watchpoints`**. You will need to delete them and create them again.


UPSCENE
Database Workbench
BLAISE PASCAL MAGAZINE
FOR CODE ELEGANCE AND DESIGN
1ST PRIZE
FOR CREATION AND
DEVELOPMENT
2024


DESIGN
IMPLEMENT
DEBUG
MAINTAIN

CONTROLLING THE BROWSER USING WEBASSEMBLY

ACCESSING THE BROWSER APIS FROM WEBASSEMBLY.

BY MICHAEL VAN CANNEYT

Starter Expert

ABSTRACT
WebAssembly modules have no access to the world outside the webassembly virtual machine, except through the **API**s that are made available from the host environment. The Browser has lots of **API**s, and in this article we show how to make use of all possible Browser **API**s in **WebAssembly**. Moreover we will show that you can use these **API**s as if you were programming Javascript directly.

❶ INTRODUCTION

The **WebAssembly** support of **FreePascal** has been introduced in some previous articles:
FreePascal can compile your pascal code to **WebAssembly**, and the resulting webassembly file can be run in any hosting environment.
The most used hosting environment is still the browser. Still, many efforts are underway to make **webasssembly** usable in dedicated containers: this offers the possibility to create safe sandboxed environments for your programs.
Your programs will be safe and **sandboxed**, because a **webassembly** can only communicate with the world outside the **webassembly** through the APIs that are made available by the hosting environment.
The **webassembly** standard does not specify what **APIs** a hosting environment needs to expose, it only describes how these **APIs** can be exposed.
In order for a **FreePascal** program to run, it requires the host environment to expose the **WASI API** to the webassembly. This **API** is managed separately by the **WebAssembly** committee, and offers some limited services: file access, getting the time and so on.
It provides just the calls that allow the **FPC** team to implement the **SysUtils** unit, which provides these basic services to your pascal program.
Inversely, a **webassembly** module can export some functions, which can be called from the hosting environment.
This situation is shown for the browser in *figure 1 on page 2 of this article:* the **Javascript** in a web page can load a **webassembly** module. The webassembly module imports some routines made available by the **Javascript** (*the blue arrow*), and exports some functions which can be called from Javascript (*the green arrow*).
In the browser, calling a **webassembly** function suspends the javascript execution flow: the Javascript waits for the called **webassembly** function to finish, before it resumes execution.
It also means no event handlers will be executed while the **Webassembly** is executing.

Running a complete program in **WebAssembly** simply means calling the main function of the application, which must of course be exported from the **webassembly**;
In pascal this means the program begin..end block will be executed.

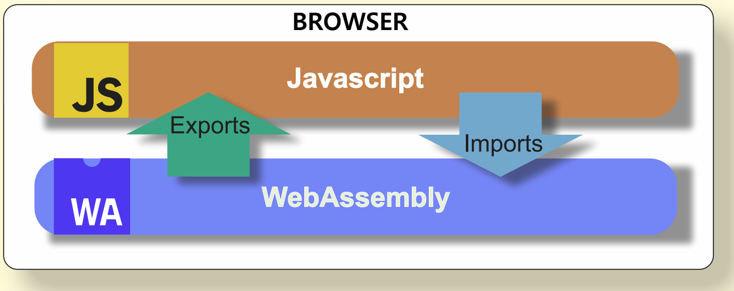


Figure 1: Import and export of functions from and to a webassembly module

❷ THE JOB FRAMEWORK

The browser has hundreds of **APIs** available in **Javascript**, these **APIs** are standardized and described in the form of interfaces. For a **webassembly** program running in the browser, it would be interesting to have access to the full browser **API**: This would allow the **Webassembly** program to do everything that can be done in **Javascript**, with the additional advantages that no-one can read the code, and that for computationally intensive tasks, the **webassembly** executes faster than **Javascript**.
Free Pascal now offers a way to access the **APIs** of the browser: The **Javascript Object Bridge** or **JOB** for short. This development was sponsored by Tixeo, a company interested in porting their software to the browser.
The **JOB** mechanism (*or* **API**) offers a way to create a proxy interface or class in **WebAssembly**, for any **Javascript API.** This means that for every class available in **Javascript**, you can create a class in **WebAssembly** that will have the same declaration as its counterpart in **Javascript**.
Whenever you create an instance of a proxy class in **WebAssembly**, this will automatically create its counterpart in **Javascript**. When you call a method or set a property on the proxy class, this will call the method or set the property on the **Javascript** counterpart of the proxy class. All this is transparent for the webassembly programmer: to the **webassembly**, it is as if he is creating and using browser **Javascript** classes in **WebAssembly**. Schematically, this looks like *figure 2 on page 3 of this article*. **JOB** does the following things to make this possible:

- You can create a Javascript object, and get a reference to this new object.
- You can get a reference to an existing Javascript object.
- Using this reference, you can call the methods of the object or set its properties as if you were manipulating a native **Pascal** object. To illustrate this, in Javascript you can set the caption of a button as follows:

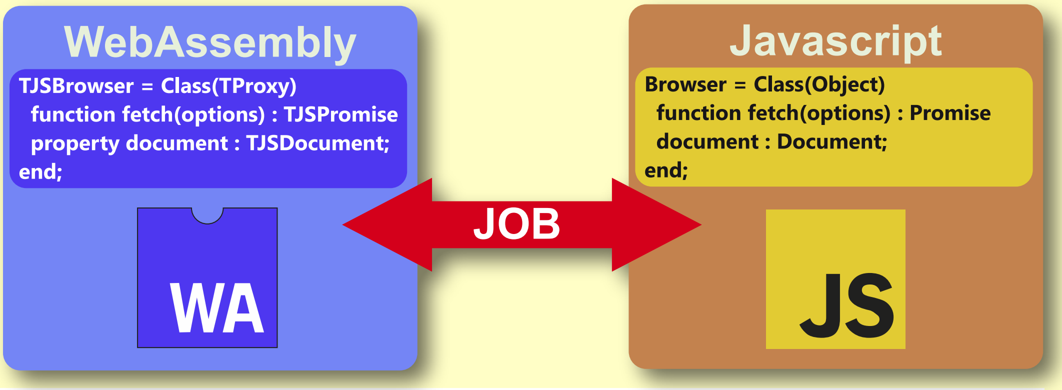

WebAssembly
TJSBrowser = Class(TProxy)
function fetch(options) : TJSPromise
property document : TJSDocument;
end;
WA
JOB
Javascript
Browser = Class(Object)
function fetch(options) : Promise
document : Document;
end;
JS

Figure 2: Webassembly proxy classes for Javascript classes

```
document.getElementById("mybutton").innerText="Press me";
```

when using **JOB**, in your **webassembly Pascal** program you can write

```
document.getElementById('mybutton').innerText:='Press me';
```

Which is of course a **one-to-one** translation of the **Javascript** code. To understand what happens, let us analyse this code. First of all, the 'document' variable is used. The document is exposed in the browser. using **JOB**, we can define an interface and instantiate a variable:

```
Type
// The API we want to use.
IJSDocument = interface(IJSNode)
  function getElementById(const aElementId: UnicodeString): IJSElement;
end;

// A class that implements this API.
TJSDocument = class(TJSNode)
  function getElementById(const aElementId: UnicodeString): IJSElement;
end;

var
  JSDocument : IJSDocument;
initialization
  JSDocument:=TJSDocument.JOBCreateGlobal('document');
end.
```

The `JOBCreateGlobal` call will retrieve a reference to the document instance in **Javascript**, and uses it to create an instance of the `TJSDocument` proxy for the Document class.
The `getElementById` method is implemented as follows:

```
function TJSDocument.getElementById(const aElementId: UnicodeString): IJSElement;
begin
  Result:=InvokeJSObjectResult('getElementById',
                                     [aElementId],
                                      TJSElement) as IJSElement;
end;
```

The `InvokeJSObjectResult` method call is part of the **JOB API**, and it executes a method in **Javascript**: the name of the method to call must be specified, as well as any arguments that the method needs.

Since the result will be an object, the Javascript side of **JOB** will return simply a reference to the resulting object in **Javascript** (*internally, this is an integer*). To convert this reference to an actual class instance, the class of the object is specified (`TJSElement`):
An instance of this class will be created, passing it the reference returned by the **Javascript** side of **JOB**.
When the `getElementById` call returns, the result is a **IJSElement** interface.
On this result, the `innerHTML` property can be set. This is also handled by **JOB**:
All properties of a **Javascript** object can be represented by **JOB** as native pascal properties:

```
Type
  IJSElement = interface(IJSNode)
    function _GetinnerHTML: UnicodeString;
    procedure _SetinnerHTML(const aValue: UnicodeString);
    property innerHTML: UnicodeString read _GetinnerHTML  write _SetinnerHTML;
  end;

  TJSElement = class(TJSNode)
    function _GetinnerHTML: UnicodeString;
    procedure _SetinnerHTML(const aValue: UnicodeString);
    property innerHTML: UnicodeString read _GetinnerHTML  write _SetinnerHTML;
  end;
```

The implementation of the **`Read/Write`** accessors is quite simple:

```
function TJSElement._GetinnerHTML: UnicodeString;
begin
  Result:=ReadJSPropertyUnicodeString('innerHTML');
end;

procedure TJSElement._SetinnerHTML(const aValue : UnicodeString);
begin
  WriteJSPropertyUnicodeString('innerHTML',aValue);
end;
```

The use of interfaces make sure that when an (*intermediate*) object is no longer needed, the object also released on the **Javascript** side. To make all this possible, on the Javascript side, the **JOB API** consists of (*currently*) 11 API methods. When these 11 methods are implemented, the webassembly can use proxy classes to execute any method on any object in the browser.
A default **Javascript** implementation for **JOB** has been developed using **PAS2JS** (*naturally*), but one could write this **API** in plain **Javascript** as well. The **JOB** technology is implemented in 2 units:

Job.js for the webassembly program: it implements the various **JOB** calls that handle encoding a call to the **Javascript** side of things, sends the call description to the **Javascript** environment and when the call returns, it retrieves the result and converts it, if needed, to an object instance.

Job Browser for the **PAS2** program. This implements the decoding of a call, executes the call on the **Javascript** object, and when the call returns, it encodes the result and sends it back to the **WebAssembly**.

There is a third (*shared*) unit which contains some common constants and types that make up the **JOB API.**

❸ WEBIDL2PAS REVISITED

In the above code examples, we showed how to access arbitrary methods and properties of some Javascript objects. The examples made use of an interface and a class that implements this interface. It makes clear that for every method you wish to call and for every property you wish to get or set, a small piece of 'glue' code needs to be created: a proxy object for every **Javascript** object. If all classes and **APIs** of the browser must be encoded like this, this is a lot of work.
You could call the **JOB** methods directly, in that case no classes and no glue code needs to be produced. The disadvantage of that approach is that there is no type safety, and no code completion if you want to code in the **IDE**. You also will need to manage the lifetime of the objects explicitly.
Luckily, there is no need to code all these proxy classes. This task can be automated. All browser **APIs** are standardized by the **W3C** committees using a **IDL (Interface Definition Language)** called **WebIDL**. All browser creators use these **IDL** files to implement their **Javascript APIs**.
The **Mozilla** foundation maintains these files, they are available at:

`https://hg.mozilla.org/mozilla-central/file/tip/dom/webidl`

or on:

`https://github.com/mozilla/gecko-dev.git`

There are some really minor differences between these archives, most likely due to the time it takes to synchronize. As you can see in these archives, there are more than 700 files, representing all the APIs made available by the browser.
In a previous article on Pas2js, the **`webidl2pas tool`** that comes with **Free Pascal** and **PAS2JS** was discussed. This tool can transform a .webidl file to a **Pascal** external class definition that can be used by **Pas2JS**. The tool has been adapted so it can now also create the proxy classes to access all the browser **APIs** from **`webassembly`**.
By downloading and concatenating all **`.webidl`** files from the above sources and applying some small patches (*the files are not perfect, and one or two constructs are not possible in Pascal*), a pascal unit can be produced that describes all these **APIs**.
Such a file has been committed to the **FPC** git repository: job web (*the file is located in packages/wasm-job/examples*). The file is huge. The interface section is about 80.000 lines long and contains roughly 1600 interface declarations, and a similar amount of classes.
This represents all the available browser **APIs**:

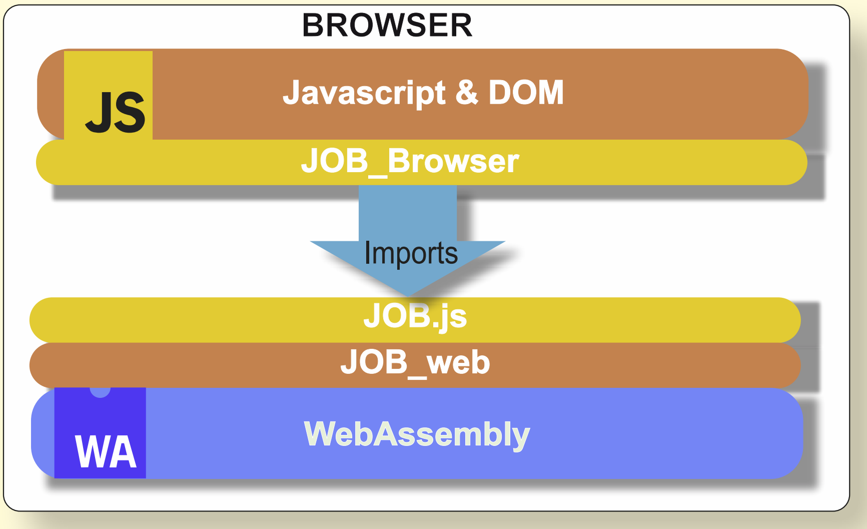


Figure 3: The various layers used in webassembly to use the browser APIS

by using this file in your webassembly program, you have direct access to all possible browser APIs. Diagrammatic, the architecture of a web application wishing to use the Javascript and DOM APIs using JOB looks like figure 3 on page 6.

❹ A JAVASCRIPT CAMERA APPLICATION

To make all this a little more understandable, we'll create an example: a web page where we have a video element, connected to the camera, and a canvas where we can create a picture (*a still*) of what the camera is showing. Basically, a camera application as you would have it on your smartphone.
We will make this application first in **Javascript**, then in **PAS2JS** and lastly we'll make it using a **webassembly program**. We'll show how the code is similar at each stage.
The HTML for this webpage will be the same in all 3 cases.
The actual program will be in the **camera.js javascript** file:

```
<!doctype html>
   <head>
      <meta http-equiv="Content-type" content="text/html; charset=utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <title>Video capture still - Javascript</title>
      <link rel="stylesheet" href="css/bulma.min.css">
      <link rel="stylesheet" href="css/camera.css">
      <script src="camera.js" type="application/javascript"></script>
   </head>
   <body>
      <div class="container">
        <h1 class="title is-1">Capture still from video</h1>
        <div class="columns">
          <div class="camera column">
            <video id="video">Video stream not available.</video>
          </div>
          <div class=" column">
            <canvas id="canvas" ></canvas>
          </div>a
        </div>
        <div class="box columns is-centered">
          <div class="column is-3">
            <button id="start" class="button is-info"></button>
            <button id="still" class="button is-link"></button>
          </div>
        </div>
      </div>
   </body>
</html>
```

As usual we use some **Bulma CSS** classes to format the page.
There are 4 elements which are important, so they have an id attribute:

video a video element, which will show the camera feed.
canvas a canvas element, which will show the still.
start a button to start the camera feed. When pressed, this will ask for permission to use the camera. Note that this button does not show a caption, it will be set in code.
stil a button to create a still (*photo*) from the camera feed.
Similarly, the caption for the button will be set in code.

The id attribute is used to get a reference to the elements when the page is loaded, in the **camera.js javascript** program:

JAVASCRIPT CODE

```
var video = null;
var canvas = null;
var context = null;
var photo = null;
var startbutton = null;
var stillbutton = null;

function startup() {
   video = document.getElementById(’video’);
   canvas = document.getElementById(’canvas’);
   context = canvas.getContext(’2d’);

   startbutton = document.getElementById(’start’);
   startbutton.innerText = ’Start video’;
   startbutton.addEventListener(’click’, start video);

   stillbutton = document.getElementById(’still’);
   stillbutton.innerText = ’Create still’;
   stillbutton.addEventListener(’click’, createstill);
}

window.addEventListener(’load’, startup);
```

NOTE how a reference to each of the 4 elements is stored in a variable.
We also store context for the canvas, this context is used later to draw on the canvas.
The startvideo event handler is called when the user clicks the ’start’ button:

JAVASCRIPT CODE

```
function startvideo(ev) {
  navigator.mediaDevices.getUserMedia({
        video: true,
        audio: false
      })
    .then(function(stream) {
        video.srcObject = stream;
        video.play();
      })
    .catch(function(err) {
        console.log("An error occurred: " + err);
      });
}
```

The `getUserMedia` call will ask for permission to use the camera. This function returns a promise, and when the promise resolves, the stream is coupled to the video element. Lastly, the ’click’ handler for the ’**still**’ button draws the current video frame on the canvas:

JAVASCRIPT CODE

```
function createstill(ev) {
canvas.width = video.clientWidth;
canvas.height = video.clientHeight;
context.drawImage(video, 0, 0, video.clientWidth, video.
clientHeight);
}
```

And that’s all there is to creating a camera program using the browser. You can load this page from a webserver using the browser, or you can open it by double clicking the file in the file explorer: your default browser will open and show the application. In both cases, the program will function.

❺ THE CAMERA APPLICATION IN PAS2JS

In a first step, we will code the camera application in **PAS2JS**.
This will allow us to transform the **Javascript** to **PASCAL**, without concerning ourselves with the details of using **WEBASSEMBLY**.
The first thing to do is to add the mandatory **script** tag for running a **PAS2JS** application to the HTML:

```
<script>
  window.addEventListener(’load’ , rtl.run);
</script>
```

Then we translate our program piece by piece. We will put all code in a class,
it will become apparent in the next example why this is necessary.

Pascal Code

```
TCameraApp = class
  video        : TJSHTMLVideoElement;
  canvas       : TJSHTMLCanvasElement;
  context      : TJSCanvasRenderingContext2D;
  startbutton : TJSHTMLElement;
  stillbutton : TJSHTMLElement;
  function StartStream       (JS : JSValue) : JSValue;
  function DoError           (JS : JSValue) : JSValue;
  Procedure StartVideo       (Event : TJSEvent);
  Procedure CreateStill      (Event : TJSEvent);
  procedure Run;
end;
```

This class declares the same variables and functions as our Javascript code.
The main difference is of course that **Pascal** is a strongly typed language, and we must specify the types of all variables, method arguments and function results.
The main program simply creates an instance of this class and calls the **Run** method:

Pascal Code

```
With TCameraApp.Create do
    Run;
```

The run method looks suspiciously familiar:

Pascal Code

```
Procedure TCameraApp.Run;
begin
  video   :=TJSHTMLVideoElement(document.getElementById(’video’));
  canvas  :=TJSHTMLCanvasElement(document.getElementById(’canvas’));
  context :=TJSCanvasRenderingContext2D(canvas.getContext(’2d’));

  startbutton:=TJSHTMLElement(document.getElementById(’start’));
  startbutton.innerText:=’Start video’;
  startbutton.addEventListener(’click’, @startvideo);

  stillbutton:=TJSHTMLElement(document.getElementById(’still’));
  stillbutton.innerText:=’Create still’;
  stillbutton.addEventListener(’click’, @createstill);
end;
```

As you can see, this method is an almost copy-and-paste of the main javascript method.
The biggest difference is the typecasts, which are of course needed to keep
the **Pascal** compiler happy.

The `StartVideo` callback is slightly different. **PAS2JS**’ Web unit contains a typed defintion of the constraints argument to the `getUserMedia` call.
Using an instance of this class allows us to make sure that the correct elements are specified.
We also don’t use anonymous methods (*although this would be possible*),
but use named functions to handle the various possible outcomes of the promise:

Pascal Code

```
Procedure TCameraApp.StartVideo(Event: TJSEvent);
var
  constraints : TJSMediaConstraints;
begin
  constraints:=TJSMediaConstraints.new;
  constraints.video:=True;
  constraints.audio:=False;
  Window.navigator.mediaDevices.getUserMedia(constraints)
      ._then(@StartStream)
      .catch(@DoError)
end;
```

The `StartStream` method is executed when the promise resolves correctly. The promise resolved result (**JS**) must be typecast to the correct class before we can assign it to the srcObject property of the video element:

```
Function TCameraApp.StartStream(JS : JSValue) : JSValue;
begin
   Result:=Undefined;
   video.srcObject:=TJSHTMLMediaStream(JS);
   video.play();
end;
```

Other than that, the code is identical to the Javascript implementation. The same is true for the `DoError` method:

```
Function TCameraApp.DoError(JS : JSValue) : JSValue;
begin
   Result:=Undefined;
   console.log('An error occurred: ' + String(JS));
end;
```

Lastly, the 'click' event handler of the still button is again almost a copy and paste of the corresponding Javascript code.

```
Procedure TCameraApp.CreateStill(Event: TJSEvent);
begin
   canvas.width:=video.clientWidth;
   canvas.height:=video.clientHeight;
   context.drawImage(video, 0, 0, video.clientWidth, video.clientHeight);
end;
```

And with this the demo application is translated to **pascal**.
The workings of this application are no different from the pure **Javascript** version, and the **Pascal** code is - disregarding its **Pascal** nature - the same as the Javascript code.

6 THE CAMERA APPLICATION IN WEBASSEMBLY

Lastly, we come to the part that is the focus of this article: the webassembly program. To make this application using **webassembly**, we need to create actually 2 applications: the **webassembly loader program**, and the **webassembly program** itself.

The former is a small boilerplate application, created with **PAS2JS**. It is a generic program that **can be used to load any webassembly program that uses JOB to communicate with the browser APIs.**

The webassembly program is actually a library:
in the initialization, the necessary callbacks are set up and then it needs to return control to the browser in order for the **Javascript** event loop to be run. The program logic is implemented in `TMyApplication`. This class is a descendant of `TBrowserWASIHostApplication`.
The `TBrowserWASIHostApplication` class, in turn, is a TCustomApplication descendant which allows you to start a WebAssembly module written in Free Pascal: it has been introduced in an earlier article on FPC Webassembly support. The class needs very little methods: a constructor, the `DoRun` method, and an `OnBeforeStart` method. **Note** the **`JOB Browser`** unit in the uses clause: this unit contains the `TJSObjectBridge` class, which is the implementation of the JOB mechanism:

Pascal Code

```
Program camera;
{$mode objfpc}

uses
 JS, Classes, SysUtils, Web, WasiEnv, WasiHostApp, JOB_Browser, JOB_Shared;

Type

 TMyApplication = class(TBrowserWASIHostApplication)
 Private
   FJOB: TJSObjectBridge;
   function OnBeforeStart(Sender: TObject;
     aDescriptor: TWebAssemblyStartDescriptor): Boolean;
 Public
   constructor Create(aOwner : TComponent); override;
   procedure DoRun; override;
end;
```

The `TJSObjectBridge` is the class that registers the needed **JOB** functions in the webassembly. Under normal circumstances, only 1 property of this class needs to be set in order for it to do its work: the `WasiExports` property.
Other than that it performs its work completely in the background.
So, we create an instance of `TJSObjectBridge`, pass it the WasiEnvironment so it can register itself with the **Webassembly** modules that are loaded later on, and store a reference to it in **FJOB**:

```
Constructor TMyApplication.Create(aOwner: TComponent);
begin
   inherited Create(aOwner);
   FJOB:=TJSObjectBridge.Create(WasiEnvironment);
   RunEntryFunction:='_initialize';
end;
```

The last line in this function sets `RunEntryFunction` to `_initialize`.
This must be done because our **`webassembly`** module is a library:
The default run entry point (*used for programs*) is `_start.`
For a library, only the initialization of the library must be performed,
and the exported function that handles this initialization is called `_initialize`.
In the `DoRun` method, we simply call `StartWebAssembly`, passing it the name of the created **webassembly** function

```
Procedure TMyApplication.DoRun;

var
  wasm : String;

begin
  Terminate;
  // Allow to load file specified in hash: index.html#mywasmfile.wasm
  Wasm:=ParamStr(1);
  if Wasm='' then
  Wasm:='wasmcamera.wasm';
  StartWebAssembly(Wasm,true,@OnBeforeStart);
end;
```

The `ParamStr(1)` retrieves the first name after the hash sign in the URL.
If set, then it is interpreted as the name of the webassembly file to load. If not set, we use **'wasmcamera.wasm'** as the name.
The `StartWebAssembly` function will load the requested webassembly and executes the run entry function. (*in our case*, `_initialize`). The last parameter is an event which is executed right before calling the run entry function:
this allows the caller to do extra initialization after the webassembly module was loaded, but **before** the `start` function is called.
The event handler sets the `WasiExports` property of the `TJSObjectBridge` instance to the list of exported functions from the webassembly:

Pascal Code

```
Function TMyApplication.OnBeforeStart(Sender: TObject;
      aDescriptor: TWebAssemblyStartDescriptor): Boolean;
begin
  FJOB.WasiExports:=aDescriptor.Exported;
  Result:=true;
end;
```

The JOB framework needs a single exported function which it uses to call callback functions (event handlers) in webassembly. It searches this function in the list in WasiExports.
All that is left to do is to create and initialize an instance of our application class and call the Run method, the usual code needed when using the application class:

```
var
  Application : TMyApplication;
begin
  Application:=TMyApplication.Create(nil);
  Application.Initialize;
  Application.Run;
end.
```

With this, the loader for our **webassembly module** is finished. **NOTE** that there is no code specific to our camera application: this is completely generic code that can be used to load any **webassembly** module which needs the **JOB** framework. So now we turn to the code for our **webassembly** module, which is implemented as a library. It starts out in the usual way:

```
library wasmcamera;
{$mode objfpc}
{$h+}
{$codepage UTF8}

uses SysUtils, Variants, Job.Js, JOB_Web;
```

Note that it uses the **`Job.Js`** unit with the implementation of the webassembly side of the **JOB** mechanism, and the **`JOB WEB`** unit, which was generated by the webidl2pas tool. Then it defines the `TCameraApp` class:

```
type
 TCameraApp = class
   Video: IJSHTMLVideoElement;
   Canvas: IJSHTMLCanvasElement;
   StartButton: IJSHTMLButtonElement;
   StillButton: IJSHTMLButtonElement;
   Context: IJSCanvasRenderingContext2D;
   function StartStream(const Res : Variant) : Variant;
   function DoError(const Res : Variant) : Variant;
   procedure StartVideo(Event: IJSEvent);
   procedure CreateStill(Event: IJSEvent);
   procedure Run;
end;
```

As you can see, this class is virtually identical to the class for the **PAS2JS** program. The only thing that changes are some types: Instead of classes (*using prefix* **`TJS`**) we use interfaces (*using prefix* **`IJS`**). The **PAS2JS JSValue** is replaced with **`Variant`**:
both correspond to the any type in the **`IDL`** descriptions of the **`APIs`**. The **`Run`** method, which actually will initialize our application, is almost a copy of the pas2js code:

```
Procedure TCameraApp.Run;
begin
  Video:=TJSHTMLVideoElement.Cast(JSDocument.getElementById('video'));
  Canvas:=TJSHTMLCanvasElement.Cast(JSDocument.getElementById('canvas'));
  Context:=TJSCanvasRenderingContext2D.Cast(Canvas.getContext('2d'));

  StartButton:=TJSHTMLButtonElement.Cast(JSDocument.getElementById('start'));
  StartButton.InnerHTML:='Start video';
  StartButton.addEventListener('click', @StartVideo);

  StillButton:=TJSHTMLButtonElement.Cast(JSDocument.getElementById('still'));
  StillButton.InnerHTML:='Create still';
  StillButton.addEventListener('click', @CreateStill);
end;
```

NOTE the calls to the `Cast` class method in order to do a typecast from one interface type (*in this case* `IJSElement`) to another interface type.
This is needed in order to be able to do some reference count housekeeping. A regular typecast would result in wrong reference counts and could lead to objects being destroyed in **Javascript** when they're still used in the webassembly.
Other than that, the code is identical to the **PAS2JS** code or the **Javascript** code: no trickery is needed to set the callbacks, a real **pascal** event handler can be used.
It should be noted that all event handlers are declared with `'of object'`, meaning that only methods of classes can be used as callback handlers, plain routines cannot be used.
The `StartVideo` callback handler also looks surpisingly familiar:

Pascal Code

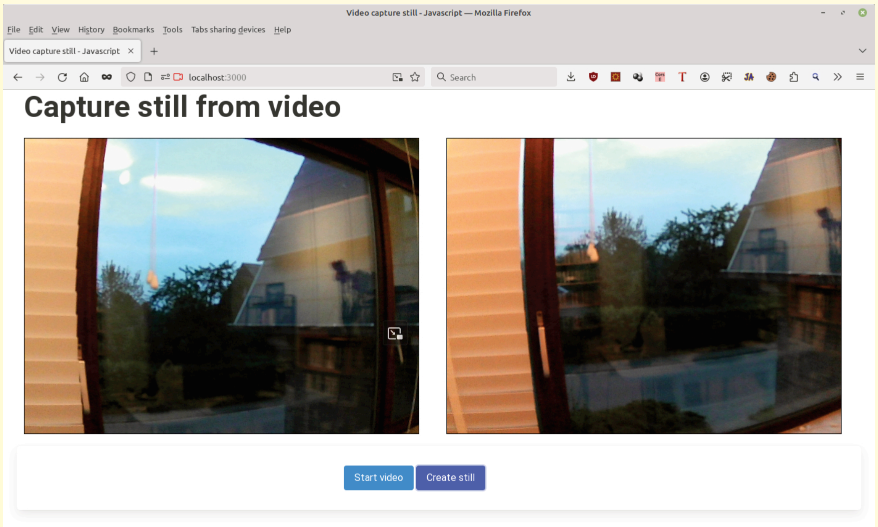
```
Procedure TCameraApp.StartVideo(Event: IJSEvent);
var
  constraints : TJSMediaStreamConstraints;
begin
  constraints:=TJSMediaStreamConstraints.Create;
  constraints.video:=True;
  constraints.audio:=False;
  JSWindow.navigator.mediaDevices.getUserMedia(Constraints)
        ._then(@StartStream)
        .catch(@DoError)
end;
```

Except for a constructor that is named `Create`, as opposed to the customary New in **PAS2JS**, the code is identical.
The `getUserMedia` returns a promise, and when this is resolved `StartStream` is called, which is again a copy of the **PAS2JS** method:

```
function TCameraApp.StartStream(const Res : Variant) : Variant;
var
  Stream : IJSMediaStream;
begin
  Stream:=IJSMediaStream(Res);
  Video.srcObject := Stream;
  Video.play();
end;
```

In case of an error, `doError` is called. Again, no change in code:

```
Function TCameraApp.DoError(const Res : Variant) : Variant;
begin
   writeln('Error accessing the webcam: '+string(Res));
end;
```


Video capture still - Javascript — Mozilla Firefox
File Edit View History Bookmarks Tools Tabs sharing devices Help
Video capture still - Javascript
localhost:3000
Search

Capture still from video

Figure 4: The webassembly camera program at work

The code for the `CreateStill` method, is also unchanged:

Pascal Code

```
Procedure TCameraApp.CreateStill(Event: IJSEvent);
begin
  Canvas.width:=Video.clientWidth;
  Canvas.height:=Video.clientHeight;
  Context.drawImage(Video,0,0,Video.ClientWidth,Video.ClientHeight);
end;
```

All that is left to do is to export a callback function which the **JOB** framework needs (`JOBCallBack`, *implemented in the* `Job.Js` *unit*), and to create an instance of our camera application class:

```
Exports
  JOBCallback;

begin
  With TCameraApp.Create do
    Run;
end.
```

For all practical purposes, the webassembly program can be coded as the **javascript** version or **PAS2JS** version would be coded. The result of all this work is shown in *figure 4 on page 11 of this article*.

❼ USING CUSTOM OBJECTS

JOB is used to give access to all the browser **APIs**. However, is it also possible to use custom objects created in **PAS2JS** or any other **Javascript API** from any **Javascript** framework?
The answer is 'Yes, of course'. You can perfectly code a webassembly proxy for a **PAS2JS** pascal class or a **Javascript** class. If a `.webidl` exists for the **Javascript** class, the proxy code could be generated by the **webidl2pas** tool.

Javascript classes have a function that serves as the constructor.
If this is a globally registered function, the **JOB** framework will find the function:
it looks for the constructor function in the global (`window`) scope.
All that is needed is to declare the name of this function in the **webassembly proxy** class.

For **PAS2JS** classes, you can specify a constructing function in the host environment.
Given the following class implemented in **PAS2JS**:

```
TMyObject = Class(TObject)
private
  fa: String; external name 'a';
public
  Constructor Create(aValue : string);
  Property a : String Read fa write fa;
end;

constructor TMyObject.Create(aValue: string);
begin
  fa:=aValue;
end;
```

You can create a constructor function to create a **Javascript** instance of this function.
The constructor function accepts the name of the requested object, and the parameters for the constructor which are provided in an array of `JSValue` (`variants`, *for all practical purposes*).

In the host application presented earlier, this would mean adding a method as follows:

```
Function TMyApplication.CreateMyObject(const aName: String;
                                        aArgs: TJSValueDynArray): TObject;
begin
  Result:=TMyObject.Create(String(aArgs[0]));
end;
```

NOTE that because the aName parameter contains the requested class name, you can use a single constructor function to construct many classes.

Registering the constructor function with the **JOB** framework is done using the `RegisterObjectFactory` call of the `TJSObjectBridge` class:

```
FJOB.RegisterObjectFactory('MyObject',@CreateMyObject);
```

You can do this call right after creating the `TJSObjectBridge` class.
If some **Javascript** class does not register itself in the global scope, then the **JOB** implementation will not find it without help. You can register a function that creates a regular **Javascript** object in a similar manner as for a **Pascal** class:

Pascal Code

```
function TMyApplication.CreateBrowserObject(const aName: String;
                                            aArgs: TJSValueDynArray): TJSObject;
begin
  Result:=TJSObject.New;
  Result['Aloha']:=String(aArgs[0]);
end;
```

In the above example, a plain **Javascript** '`Object`' instance is created,
but in fact any **Javascript** object can be returned.
This constructor function must also be registered with the `RegisterJSObjectFactory` call:

```
FJOB.RegisterJSObjectFactory('MyBrowserObject',@CreateBrowserObject);
```

The reason that two different calls are needed is that, from an **Object Pascal** point of view,
the **Javascript** `TJSObject` inheritance tree is distinct from the **Object Pascal** `TObject` inheritance tree.

After these calls, when the webassembly part of **JOB** needs to create an instance of **MyObject** or a **MyBrowserObject**, the correct registered function will be called to create an instance.
The necessary housekeeping will be done as it is done for Browser-provided objects: associate an **ID** with the object, and return that **ID** to the **webassembly**.

The **webassembly** proxy interface and class for the `TMyObject` class look as follows:

```
IJSTestObj = Interface (IJSObject)
    ['{DE03E9A4-3960-4090-A3FA-387B61E8AEA9}']
  function GetStringAttr : UnicodeString;
  procedure SetStringAttr(const aValue : UnicodeString);
  property StringAttr : Unicodestring  Read GetStringAttr
                                       Write SetStringAttr;
end;

TMyTestObj = Class(TJSObject,IJSTestObj)
  constructor Create(a: String);
  class function JSClassName: UnicodeString; override;
  function GetStringAttr : UnicodeString;
  procedure SetStringAttr(const aValue : UnicodeString);
  property StringAttr : Unicodestring Read GetStringAttr
                                      Write SetStringAttr;
end;
```

The implementation of the proxy class is simple. The `JOBCreate` method of `TJSObject` can be used to construct a new object. In order to do its work, it needs to know the class name of the **Javascript** class.
It expects the `JSClassName` class function to return the correct class name,
so we override that function and let it return the name we used to register our constructor function:

Pascal Code

```
class function TMyTestObj.JSClassName: UnicodeString;
begin
  Result:=’MyObject’;
end;

constructor TMyTestObj.Create(a: String);
begin
  Inherited JobCreate([a]);
end;
```

The `JOBCreate` method accepts parameters as an array of const, which are encoded and sent to the browser side. The implementation of the property getters and setters are simple:

```
function TMyTestObj.GetStringAttr: UnicodeString;
begin
  Result:=ReadJSPropertyUnicodeString(’a’);
end;

procedure TMyTestObj.SetStringAttr(const aValue: UnicodeString);
begin
  WriteJSPropertyUnicodeString(’a’,aValue);
end;
```

Similarly named **`ReadJSProperty`*** and **`WriteJSProperty`*** calls exist for all simple **Pascal** types, you must choose the function that corresponds to the type of the property in your Javascript class.

NOTE that the name of the field is given as ’**a**’: this is the **Javascript** name of the field in the **Pascal** class: it was forced to ’a’ using the external name ’**a**’ modifier in the class declaration. Without this modifier, ’**fa**’ would need to be used.

If a property must be set using a `setter/getter,` then you must adapt the proxy code accordingly, of course: you must then code a call to the `getter` and `setter`.

The class is now ready for use in your webassembly program:

```
var
  T : IJSTestObj;
begin
  Writeln(’Creating TMyTestObj object’);
  T:=TMyTestObj.Create(’solo’);
  Writeln(’Property : ’,T.StringAttr);
end;
```

SOURCE CODE

The expected output is of course ’solo’ for the property value. The **SOURCE CODE** that demonstrates this is included in the **PAS2JS** suite of demos, under the `demo/wasienv/job/simple` directory. or **you can download it from your Blaise Pascal Magazine code page:**
https://www.blaisepascalmagazine.eu/en/your-downloads/

❽ CONCLUSION

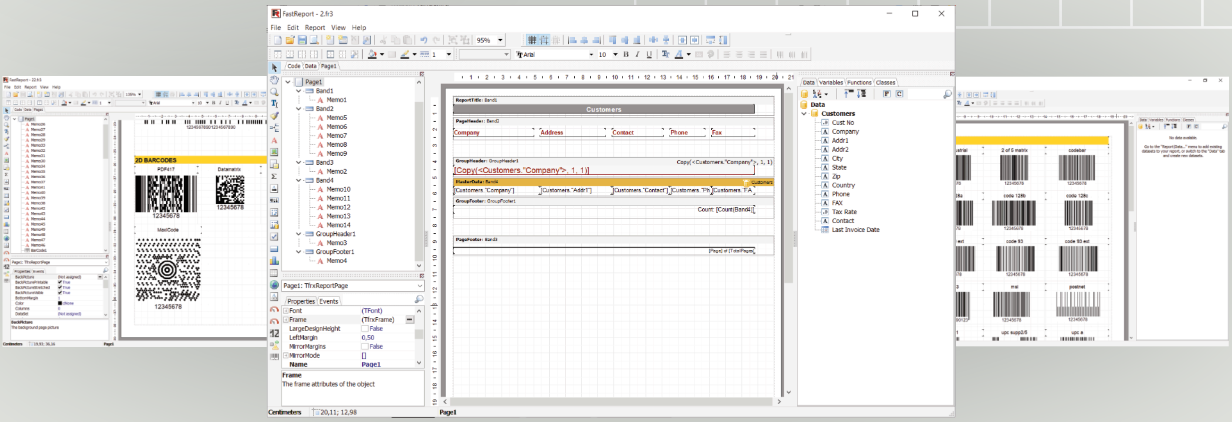
With the **JOB** technology, it is now possible to use all browser APIs in a **webassembly** program without having to resort to lots of import/export routines: **a list of 11 functions is sufficient** to create and use every possible browser object. To the best of the author’s knowledge, currently the only other compiled language – compilable to **WebAssembly** – that offers this possibility is **Rust**.

As indicated above, the job web unit is large. This is somewhat of a disadvantage:
the compiler takes a lot of time compiling this unit, well over 1 minute.
The reason is the use of interfaces, which result in a lot of hidden code to call methods on an interface, and the resulting unit is well over 65Mb. While the linker **removes all unused code** and your program will contain only the **needed code**, the unit must be compiled (*luckily only once*) and this takes time.

Fast Reports
Reporting must be Fast!

Create professionally designed reports with minimal effort with FastReport VCL.

Create professionally designed reports with minimal effort with FastReport VCL.



Multiple graphical elements for information visualization, export filters to 30+ formats, easy integration with data, secure storage. FastReport VCL, with its simplicity, convenience and small distribution size, is able to provide proper functionality and speed on any modern computer.

In the latest version 2024.2:

A new package with visual components TfrTreeView. Presentation of data in an intuitive way.

Support for GeoJSON and TopoJSON formats in the FastReport VCL Maps object.

Lazarus support in FastQueryBuilder. Integration into Lazarus projects and data workflow improvements.

Try the demo today to save time and resources on report creation.

www.fast-report.com

FPC/LAZARUS FRESNEL

Now fpc/lazarus using fresnel
Has three working backends,
A css-driven layout,
Multiple platforms,
A powerful event mechanism.

We now can:

CREATE A UNIVERSAL GRAPHICAL APPLICATION RUNNING ON ALL NATIVE PLATFORMS AND IN THE BROWSER.

All this using a single codebase,
and running at native speed.
And obviously,
all this using your favourite Programming language:

OBJECT PASCAL.

PROJECT FRESNEL: THE TIME IS NOW - AN UPDATE

BY MICHAEL VAN CANNEYT

ABSTRACT
At the end of the year 2022,
Project **Fresnel** was announced:
a new graphical interface for **Pascal** applications,
based on CSS. Since then, work has been steadily progressing
on this new framework.
In this article an overview of **what is possible today is presented**.

❶ INTRODUCTION

Project Fresnel was announced in this magazine a little over 1.5 years ago: Issue 107/108.
Work was started immediately, and work on project Fresnel has not stopped since.
As a reminder, the main goals of project Fresnel were:

- To create a **set of controls** (*or widgets*) that are streamable, so descendents of **`TComponent`**: the **widgets can be manipulated in the IDE**.
- **Layout is determined** completely **by CSS**.
- **Multiple drawing backends** must be supported.
- **No dependency on the Lazarus LCL**.
- **Fresnel-Based Forms can coexist with LCL forms** in a native application.

The end goal is a **UI framework** that will allow to create an **application UI** once, and let it **run on any OS and in the browser**. Conceptually, the architecture of such an application is depicted in figure 1 on page 1 of this article.

The application code only uses the **Fresnel** API to display the graphical user interface, other functionality is implemented on top of the operating system API. **Fresnel** components use the **Fresnel** backend to do the actual drawing.

A **Fresnel backend** uses the **graphical API** of the operating system – or a library that makes the drawing easier – to do the actual drawing.

Fresnel components do **not** access the APIs underlying the backend.
This ensures that **Fresnel** components will work with any backend. Several backends can be implemented, and when running your application, you choose the backend in function of the operating system for which you're compiling your application.

In this article, we report on the progress made on each of these goals.

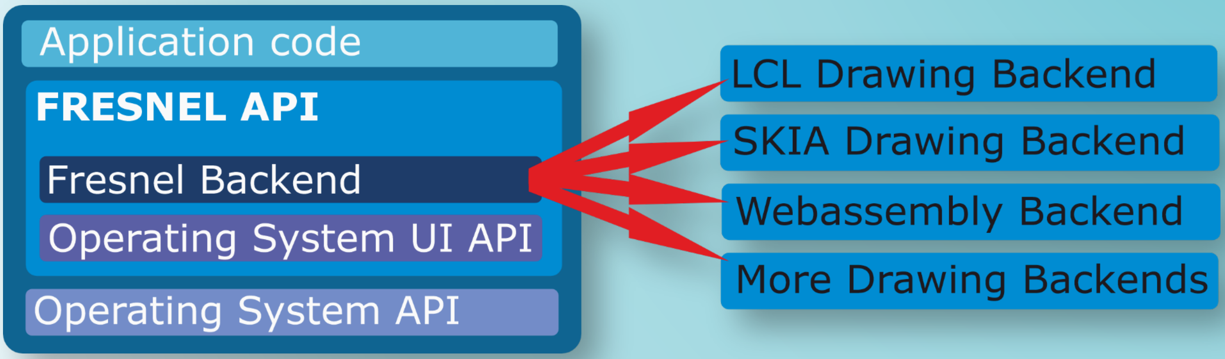


Figure 1: A Fresnel application

❷ WIDGETS OR CONTROLS

A basic set of controls (widgets) has been developed:

ViewPort This essentially encapsulates the visible portion of a form. It is the toplevel control in a **Fresnel** graphical window, and has a stylesheet associated with it that determines the style of the elements in a form.

Form is a descendent of a viewport. This is a viewport which can exist by itself.

Div is a basic building block of a graphical UI: a box for which you can specify sizes, borders, background and foreground colors etc.

Span is similar to a Div but has different layout flow behaviour: spans will be placed next to each other (*'inline' display, in CSS terms*).

Label Is exactly what the name implies: it resembles a Div but allows you to specify a caption to be shown in the box.

Image A component to show an image.

This means that today you can do the following

```
Div2:=TDiv.Create(Self);
with Div2 do
  begin
    Name:='Div2';
    Parent:=Body1;
    Style:='border-color: black; height:50px; '+
            'position: absolute; border: 2px; '+
            'left: 30px; top: 100px; width: 50px; '+
            'height: 60px;';
  end;
```

As you can see, the layout of the component is determined by the `Style` property.
Furthermore, these controls can be installed in the IDE, and you can create a **Fresnel Form** in the designer.
This part is still experimental. To do so, you need to recompile the **Lazarus IDE** with the trunk version of **Free Pascal,** as **project Fresnel** requires the use of some units that are not yet present in the released version of **Free Pascal.**.

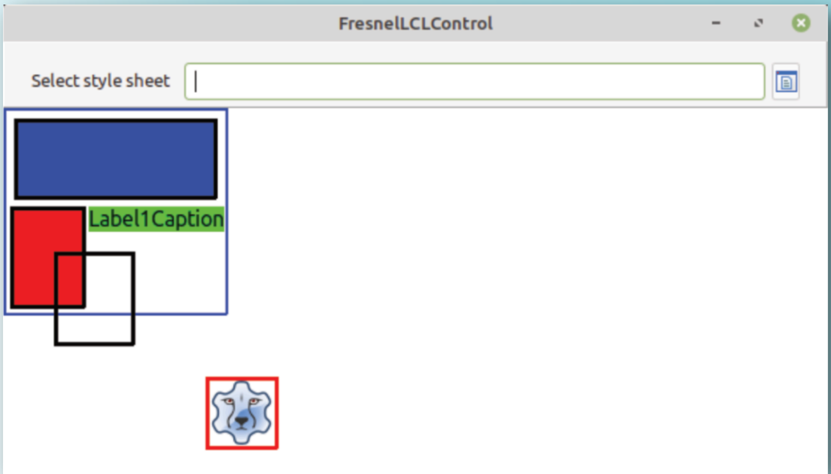


Figure 2: Using stylesheet 1

❸ CSS LAYOUT

The primary goal of project **Fresnel** is to have the layout determined by **CSS**. **CSS** originated in the browser, and became a powerful tool for creating good-looking UIs which is used in all browsers. Prior to starting **Project Fresnel**, **Free Pascal** already had a **CSS** parser available. This parser was extended to make it more robust, and an engine was developed to determine the **CSS** properties that are applicable to a given widget (control).

So today, we can specify the **CSS** of a control using the Style property, or using the style sheet of the viewport: the stylesheet can be specified in the StyleSheet property.

```
Viewport.Stylesheet.LoadFromFile('style2.css');
```

Figures 2 on page 2 and figure 3 on page 4 show the same application, but with a different stylesheet loaded. As expected, the controls adjust their properties (*and location*) according to what is specified in the **CSS**.
Needless to say, there is still a lot of work to be done: there are many **CSS** properties, and currently only the most basic properties are implemented: enough to create simple layouts, without too much of the special effects that make **CSS** such a powerful mechanism.

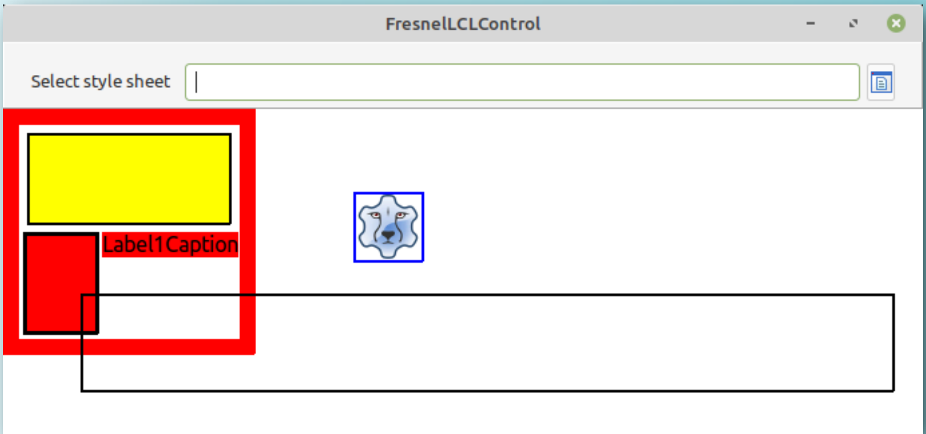


Figure 3: Using stylesheet 2

❹ FRESNEL EVENT HANDLING

In the **LCL**, the event handlers such as `OnClick, OnMouseMove` can be assigned in the Object Inspector.
The same is true for Fresnel widgets:
Fresnel is designed from the start to be **RAD-enabled**. While not specifically specified in the goals of **Fresnel**, the occasion was to used to address some of the shortcomings of the **VCL** and **LCL** event mechanisms, and a complete redesign of the event mechanism was put in place.
The first thing to mention about the new event mechanism is that the signature of the event handlers is different from in the **LCL**. The **LCL** uses the following notification event handler (with slight variations):

```
TNotifyEvent = procedure(Sender: TObject) of object;
```

Here, `Sender` is the component instance from which the event originates. You need to typecast the sender to access its properties. **The basic event handler in Fresnel looks like this:**

```
TEventHandler = Procedure(Event : TAbstractEvent) of object;
```

❹ FRESNEL EVENT HANDLING - CONTINUATION

The **Event** parameter is of type **TAbstractEvent**, which looks like this:

```
TAbstractEvent = Class(TObject)
  // Sender of the event
  Property Sender : TObject Read FSender Write FSender;
  // Event ID used for create
  Property EventID : TEventID Read FEventID;
end;
```

The `Sender` is still available as a member of the event.
Depending on the actual event, a descendent of **TAbstractEvent** is passed on, which contains the necessary information pertaining to the event. For instance, the mouse events all descend from `TFresnelMouseEvent` :

```
TFresnelMouseEvent = Class(TFresnelUIEvent)
Public
  Property ControlX : TFresnelLength;
  Property ControlY : TFresnelLength;
  Property PageX : TFresnelLength;
  Property PageY : TFresnelLength;
  Property ScreenX : TFresnelLength;
  Property ScreenY : TFresnelLength;
  Property X : TFresnelLength;
  Property Y : TFresnelLength;
  Property Buttons: TMouseButtons;
  Property Button : TMouseButton;
  Property ShiftState : TShiftState;
  Property Altkey : Boolean;
  Property MetaKey : Boolean;
  Property CtrlKey : Boolean;
  Property ShiftKey : Boolean;
end;
```

As you can see, a lot more information is available.
But there is more: for every event, multiple handlers can be registered. The basic **Fresnel** component exposes a `EventDispatcher` property, which is of type `TEventDispatcher`:

```
TEventSetupHandler      = Procedure(Event : TAbstractEvent) of object;
TEventSetupCallBack     = Procedure(Event : TAbstractEvent);
TEventSetupHandlerRef   = Reference to Procedure(Event : TAbstractEvent);

TEventDispatcher = class(TPersistent)
    // Various forms to register an event handler
    Function RegisterHandler(aHandler : TEventCallback;
                             aEventName : TEventName) : TEventHandlerItem;
    Function RegisterHandler(aHandler : TEventHandler;
                               aEventName : TEventName) : TEventHandlerItem;
    Function RegisterHandler(aHandler : TEventHandlerRef;
                               aEventName : TEventName) : TEventHandlerItem;
    // Dispatch an event.
    // Calls the registered handlers for that event,
    // in the order they were registered.
    // Returns the number of handlers that were called;
    Function DispatchEvent(aEvent : TAbstractEvent) : Integer;
end;
```

This is roughly modeled after many other event dispatching mechanisms in other toolkits (**Gtk, Qt**) and in the browser.
There are 2 things to note about this mechanism:

❶ You can register **multiple handlers for the same event.** Behind the scenes, setting the `OnClick` handler will use the event dispatcher to set one 'click' event handler. Setting the event handler to `Nil` will remove the handler from the dispatcher.

❷ Event handlers no longer need to be object methods. It can also be an **anonymous** method, or a plain procedure or a local procedure.

At the moment, you still need to typecast the `Event` to get to the properties, but a mechanism using **generics** to register a correctly typed event handler will be put in place.

❺ BACKENDS

Another important goal for **Project Fresnel** is that it must be cross platform and must support different drawing backends: the widgets or controls are not aware of the backend in use to draw them. All they get is a canvas on which they can draw themselves if so required. (*form*) and the events sent by the operating system, and the second service is to provide a canvas to draw on. These two services are defined independently and can be coded independently.

A backend needs to provide two services: The first is to manage the top-level windows This means that you could have a backend (*for example* **Gdk3/Gtk3**, *to manage the windows and events*) that uses various drawing backends (**Skia** *or* **Cairo**).
Currently, 3 working backends for the Fresnel widgets exist, and a 4th is in the works:

LCL This was the first backend created for Fresnel. The fresnel controls are drawn on an **LCL Canvas**. This can be a canvas that is embedded in a **LCL form** on a `TFresnelControl`: this is a control that embeds the **Fresnel viewport**; all drawing happens within this control. It can also be a **Fresnel LCL** form: a form that is **completely standalone**. Events are generated by the **LCL** and are transformed into **Fresnel events**. It is this backend that is used when designing a **Fresnel form** in the IDE.

Gtk3 using Skia The **Gtk3** backend is a backend which relies on the **Skia** library to render the Fresnel controls. **Skia** is a fast **2D library** by **Google** which runs on various platforms (*all major OSes and mobile devices*). By creating a **Skia** drawing backend, the **Fresnel** framework should run on all platforms that **Free Pascal**, **Skia** support. **Skia** by itself does not offer event handling, so it is paired with **Gtk3**, which is also cross-platform.

WebAssembly Lastly, a **WebAssembly** backend is made. **Free Pasca**l supports creating webassembly binaries, and these binaries can be run in the browser. A **Fresnel** backend was made which uses the browser canvas and the browser events to deliver the needed functionality to **Fresnel**. It will be presented in the rest of this article.

More backends can of course be made:
Using one of the existing backends, it should not be difficult to create a backend that sits directly on top of the OS' native UI mechanisms:

- **LCL** backend with a **Skia** renderer backend.
- **WinApi** backend with a **Skia** renderer backend.
- **WinApi** backend with a **WinApi** renderer backend.
- **WinApi** backend with a **BGRA** renderer backend.
- **Apple Cocoa** backend with a **Metal** renderer backend.
- **Apple Cocoa** backend with a **Skia** renderer backend.

One such backend which is planned by the **FPC** team is **PAS2JS**:
This would allow running a Fresnel application as a **Javascript** application.

❻ COMPILING FRESNEL

To compile **Fresnel**, the development version of **FPC** is needed: **Fresnel** uses some mechanisms which are available only in the development version of **FPC** (*for example, the* **CSS parser**).
Fresnel itself is implemented in a series of lazarus packages.
You can compile **Fresnel** and use without it the **lazarus** packages, if you so desire.

FresnelBase
this package contains the **basics of Fresnel:**
the controls, the CSS handling, the event mechanism and
the rendering backend specification
(*it is defined as an interface definition*). **This package does not depend on the LCL - this was one of the design goals**.

FresnelLCL
This contains the **LCL backend for Fresnel:** a renderer that **can render a Fresnel form on a form or in a LCL control**.

FresnelDsgn
Installing this package allows you to to **design a Fresnel form in the Lazarus IDE, as you would design a LCL form:** You can **add Fresnel forms to a standalone Fresnel application** or a **LCL application** and **drop Fresnel elements** onto the **fresnel Forms** and use the **Object Inspector to set properties** like the **Style property**.

Then there are 3 other packages that provide other backends:

fresnel
This package **automatically chooses a backend** depending on some defines.
On **linux it will choose the Gtk and Skia backend** to provide a window and an event mechanism.
The **drawing itself is done using Skia**. This is still a work in progress.

fresnelwasm
This package **contains the webassembly part** of the **WebAssembly backend**.
The webassembly backend needs two parts:
one in **webassembly**, one in the **browser**
. This package contains the **webassembly side** of the **Fresnel webassembly** backend.

p2jsfresnelapi
This package **contains the javascript part** of the **WebAssembly backend**, it must be used in the **browser host application** that loads the **Fresnel Webassembly** program.

❼ A FRESNEL APPLICATION USING THE LCL

As an example, we'll show a **Fresnel** application using an **`LCL form`** and a **`TFresnelLCLControl`** to host the **Fresnel** controls. The main form's published section only contains an **`'OnCreate'`** handler, the rest is added manually:

```
TMainForm = class(TForm)
  procedure FormCreate(Sender: TObject);
private
  Body1: TBody;
  Div1, Div2: TDiv;
  Img1 : TImage;
  Span1: TSpan;
  Fresnel1: TFresnelLCLControl;
  label1 : Fresnel.controls.TLabel;
public
  procedure CreateControls(ViewPort: TFresnelViewport);
end;
```

In the **`OnCreate`** event, we create the **LCL `TFresnelLCLControl`** that will host all Fresnel controls.
We set it to take all available space, and load a stylesheet:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
   Fresnel1:=TFresnelLCLControl.Create(Self);
   with Fresnel1 do
   begin
     Name:=’Fresnel1’;
       Align:=alClient;
       Viewport.Stylesheet.LoadFromFile(’style1.css’);
       Parent:=Self;
     end;
   CreateControls(Fresnel1.Viewport);
end;
```

In the CreateControls method, we create the Fresnel Controls:

```
Procedure TMainForm.CreateControls(ViewPort : TFresnelViewport);

Function CreateControl(aClass : TFresnelElementClass;
                       aName : String;
                       aParent : TFresnelElement = nil) : TFresnelElement;
begin
  if aParent=Nil then
     aparent:=Body1;
  Result:=aClass.Create(Self);
  Result.Name:=aName;
  Result.parent:=aParent;
end;

begin
  Body1  :=TBody(CreateControl(TBody,’Body1’,ViewPort));
  Div1   :=TDiv(CreateControl(TDiv,’Div1’));
  Span1  :=TSpan(CreateControl(TSpan,’Span1’));
  label1 :=TLabel(CreateControl(TLabel,’Label1’));
  Label1.Caption:=’Label1Caption’;
  Div2   :=TDiv(CreateControl(TDiv,’Div2’));
  Img1   :=TImage(CreateControl(TImage,’Img1’));
  Img1.Image.LoadFromFile(’image.png’);
end;
```

Note that we do not need to set any position or color properties. This is all taken care of by the **CSS**.

The `Image` property of the **`TImage`** widget deserves some extra attention.
This property is of class **`TImageData`**, which is defined as follows:

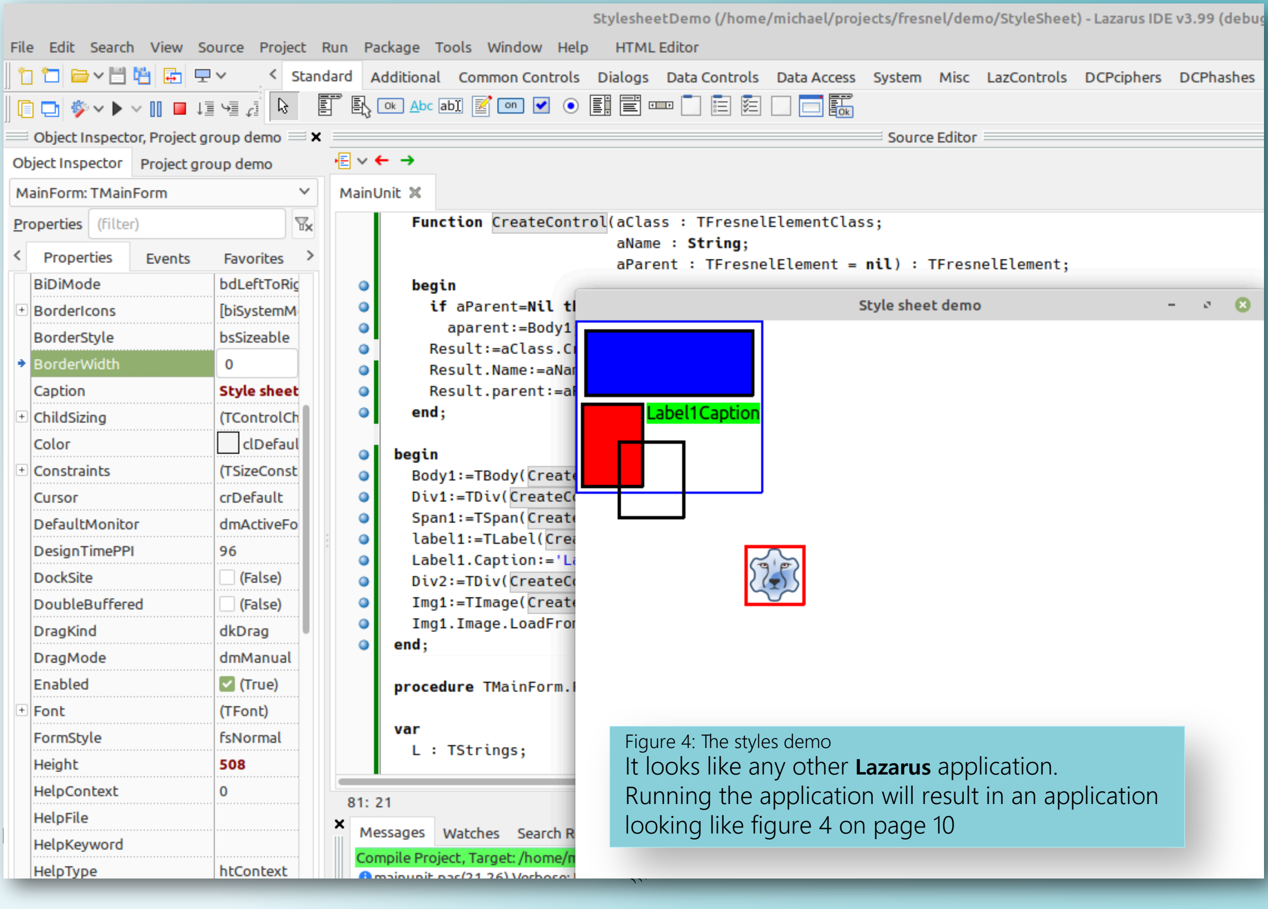
```
TImageData = class(TPersistent)
Public
  Constructor Create(aOwner : TComponent); virtual;
  Destructor Destroy; override;
  Procedure LoadFromFile(const aFilename : String);
  Procedure SaveToFile(const aFilename : String);
  Procedure LoadFromStream(const aStream : TStream;
                           Handler:TFPCustomImageReader = Nil);
  Procedure SaveToStream(const aStream : TStream;
                           Handler:TFPCustomImageWriter = Nil);
  Procedure Assign(Source : TPersistent); override;
  Property Data : TFPCustomImage;
  Property ResolvedData : TFPCustomImage;
  Property Width : Word;
  Property Height : Word;
  Property HasData : Boolean;
Published
  Property FileName : String;
  Property ImageName : String;
  Property ImageList : TBaseCustomImageList;
  Property ImageIndex : Integer;
end;
```

❼ A FRESNEL APPLICATION USING THE LCL - 2

The **LCL** and **VCL** use two approaches to specify an image: directly through a **`TGraphic`** or indirectly using an ImageList and an **`ImageIndex`** property. Which one is used depends on the actual control. In **`Fresnel`**, these two approaches have been combined in one single class: **`TImageData`**. Thus, every **Fresnel** control that needs to specify an image, has both mechanisms enabled. The `TImageData` offers also a third mechanism to load images: a `ImageName`, which is used to look up an image by name in a central image store. The central image store can look up image files by name, and can handle multiple sizes and multiple screen resolutions. It caches the images in memory.

Thus `Img1.Image.ImageName:='image'`; would look for an image file using a standard format ('`.png`') in a standard set of directories. Both the format and the directory structure are globally configurable. This mechanism makes it easy to configure a set of standard images for an application. The images are loaded and kept in memory using **Free Pascal's** `TFPCustomImage` class, which can handle many image formats by default. The main program file for our program looks like any other program:

```
program StylesheetDemo;
{$mode objfpc}{$H+}
uses Interfaces, // this includes the LCL widgetset
     Forms, MainUnit;
begin
  RequireDerivedFormResource:=True;
  Application.Scaled:=True;
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
end.
```



StylesheetDemo (/home/michael/projects/fresnel/demo/StyleSheet) - Lazarus IDE v3.99 (debu
File Edit Search View Source Project Run Package Tools Window Help HTML Editor
Standard Additional Common Controls Dialogs Data Controls Data Access System Misc LazControls DCPciphers DCPhashes
Object Inspector, Project group demo
Source Editor
Object Inspector Project group demo
MainForm: TMainForm
Properties (filter)
Properties Events Favorites
MainUnit
Style sheet demo
Label1Caption
81: 21
Messages Watches Search R
Compile Project, Target: /home/n

BiDiMode	bdLeftToRic
BorderIcons	[biSystemM
BorderStyle	bsSizeable
BorderWidth	0
Caption	Style sheet
ChildSizing	(TControlCh
Color	□ clDefaul
Constraints	(TSizeConst
Cursor	crDefault
DefaultMonitor	dmActiveFo
DesignTimePPI	96
DockSite	□ (False)
DoubleBuffered	□ (False)
DragKind	dkDrag
DragMode	dmManual
Enabled	☑ (True)
Font	(TFont)
FormStyle	fsNormal
Height	508
HelpContext	0
HelpFile	
HelpKeyword	
HelpType	htContext

```
  Function CreateControl(aClass : TFresnelElementClass;
                         aName : String;
                         aParent : TFresnelElement = nil) : TFresnelElement;

  begin
    if aParent=Nil tl
      aparent:=Body1
    Result:=aClass.C
    Result.Name:=aNa
    Result.parent:=a
  end;

begin
  Body1:=TBody(Create
  Div1:=TDiv(CreateC
  Span1:=TSpan(Create
  label1:=TLabel(Crea
  Label1.Caption:='L
  Div2:=TDiv(CreateC
  Img1:=TImage(Create
  Img1.Image.LoadFro
end;

procedure TMainForm.

var
  L : TStrings;
```

Figure 4: The styles demo
It looks like any other **Lazarus** application.
Running the application will result in an application looking like figure 4 on page 10

❽ USING THE WEBASSEMBLY BACKEND

To run a fresnel program in a webassembly backend, you need two programs. One is the **webassembly** program itself, the other is the Javascript program that loads the **Webassembly** file in the browser. This **Javascript** program we create of course with PAS2JS.
We'll start with the webassembly program itself. Due to the nature of webassembly, the fresnel program must be created as a library:
As explained in the article on using the browser's API, when the browser runs a **webassembly** program, it suspends the Javascript execution. As long as the **webassembly** program runs, no event handlers will run. So, we need to create a library that initializes the application, and then returns control to the browser, so it can start receiving events. The events are processed in the fresnel tick callback which is called at regular intervals by the browser, although this mechanism may change in the future.
This method must be exposed by the library:

```
library basic;
uses
   nothreads, fresnel.forms, fresnel.wasm.app, form.main, fresnel.wasm.api;

procedure __fresnel_tick (aCurrent,aPrevious : double);

begin
  fresnel.wasm.api.__fresnel_tick(aCurrent,aPrevious);
end;

exports
  __fresnel_tick;
begin
  Application.HookFresnelLog:=True;
  Application.Initialize;
  Application.CreateFormNew(TMainForm,MainForm);
  Application.Run;
end
```

The first line in the initialization of the library sets up a hook: the fresnel log will be written to standard output, and will show up in the browser console. The rest of the application startup code looks the same as a standard **Lazarus LCL** application. The **`CreateForm`** has been replaced with **`CreateFormNew`** so the **`CreateNew`** constructor of the form is called: Currently there are no resources in the **webassembly** (*this is being worked on*). Calling **CreateNew** makes sure no resources are loaded.
To demonstrate that the events mechanism works as expected also in the browser, our main form will also hook some events. For this reason, we define an enumeration to select the events that we want to listen to.

```
Type
  THookEvent = (heClick,heMouseMove,heMouseUp,heMouseDown,heMouseEnter,
                heMouseLeave,heFocus,heFocusIn,THookEvents = set of THookEvent;

{ TMainForm }

  TMainForm = Class(TFresnelForm)
  private
     procedure HookAllFresnelComponents;
     procedure LogEventData(Event: TAbstractEvent);
     procedure LogMouseEvent(Event: TFresnelMouseEvent; LogData: Boolean);
  Public
     procedure DoClick(Event: TAbstractEvent);
     procedure DoGeneralEvent(Event: TAbstractEvent);
     procedure DoMouseMove(Event: TFresnelMouseEvent);
     constructor CreateNew(aOwner : TComponent); override;
     procedure HookEvents(aEl: TFresnelELement; Publ: Boolean);
  end;
var
  MainForm : TMainForm;

const
AllHookEvents = [Low(THookEvent)..High(THookEvent)];
```

❽ USING THE WEBASSEMBLY BACKEND CONTINUATION 1

The form is populated in the same way as our **LCL** version, in our constructor, we call `Create-controls`. This time we pass **`Self`** as the viewport, since the Fresnel form is the actual viewport:

```
constructor TMainForm.CreateNew(aOwner : TComponent);

const
  GlobalStyle = 'div {padding: 2px; border: 3px; margin: 6px;}';
begin
  Inherited CreateNew(aOwner);
  Width :=640;
  Height:=480;
  Stylesheet.Text:=GlobalStyle;
  CreateControls(Self);
  HookAllFresnelComponents;
end;
```

NOTE that the `Width` and `Height` of the form are set: The form is the only component which has a `width` and `height` property - this is logical, since it is the top-level control. We'll come to the last line shortly, they hook all **fresnel** events for all controls on the form. Note that we only set the globally applicable **CSS styles** in the **StyleSheet** property:

these styles will be used for all controls, in addition to the **CSS** styles specified in the '`Style`' property of each control. In `CreateControls`, we demonstrate that the **CSS** styles can also be directly applied to the controls by setting the `Style` property:

```
Procedure TMainForm.CreateControls(ViewPort : TFresnelViewport);

  Function CreateControl(aClass : TFresnelElementClass; aName : String;
                         aParent : TFresnelElement = nil) : TFresnelElement;

begin
  if aParent=Nil then
    aparent:=Body1;
  Result:=aClass.Create(Self);
  Result.Name:=aName;
  Result.parent:=aParent;
end;

begin
  Body1  :=TBody(CreateControl(TBody,’Body1’,ViewPort));
  Div1   :=TDiv(CreateControl(TDiv,’Div1’));
  Span1  :=TSpan(CreateControl(TSpan,’Span1’));
  label1 :=TLabel(CreateControl(TLabel,’Label1’));
  Label1.Caption:=’Label1Caption’;
  Div2   :=TDiv(CreateControl(TDiv,’Div2’));
  Img1   :=TImage(CreateControl(TImage,’Img1’));
  Img1.Image.LoadFromFile(’image.png’);

  // Apply styles
  Body1.Style  :=’border: 2px; border-color: blue;’;
  Div1.Style   :=’background-color: blue; border-color: black; height:50px;’;
  Span1.Style  :=’width: 50px; height:70px; background-color: red; ’+
                ’border: 3px; border-color: black; margin: 3px;’;
  Label1.Style :=’background-color: green; ’;
  Div2.Style   :=’border-color: black; position: absolute; border: 2px;’+
                ’ left: 30px; top: 100px; width: 50px; height: 60px;’;
  Img1.Style   :=’border-color: red; height:50px; position: absolute; ’+
                ’border: 2px; left: 150px; top: 200px; width: 48px; height: 48px;’;
end;
```

Basically, this is the contents of the style sheet used in our previous example, but applied directly to the controls. To demonstrate events, we set some event handlers on the events. The `HookAllFresnelComponents` simply loops over all controls and calls `HookEvents`

❽ USING THE WEBASSEMBLY BACKEND CONTINUATION 2

```
procedure TMainForm.HookAllFresnelComponents;

Var
  C : TComponent;
  I : Integer;
  UsePublished : Boolean;

begin
  UsePublished:=False;
  HookEvents(Self,UsePublished);
  For I:=0 to ComponentCount-1 do
    begin
      C:=Components[I];
      if C is TFresnelElement then
        HookEvents(C as TFresnelElement,UsePublished);
    end;
end;
```

The `UsePublished` parameter allows you to select which mechanism must be used: Specifying `True` will set the traditional property, specifying `False` will use the `AddEventListener` mechanism of the event dispatcher:

```
procedure TMainForm.HookEvents(aEl: TFresnelELement; Publ : Boolean);

begin
 if Publ then
    begin
      aEl.OnClick       :=@DoClick;
      aEl.OnMouseMove   :=@DoMouseMove;
      aEl.OnMouseEnter  :=@DoMouseMove;
      aEl.OnMouseLeave  :=@DoMouseMove;
    end
 else
    begin
      aEl.AddEventListener('click',@DoGeneralEvent);
      aEl.AddEventListener('mousemove',@DoGeneralEvent);
      aEl.AddEventListener('mouseenter',@DoGeneralEvent);
      aEl.AddEventListener('mouseleave',@DoGeneralEvent);
      aEl.AddEventListener('focus',@DoGeneralEvent);
    end;
end;
```

The general event handler `DoGeneralEvent`, which is registered using `AddEventListener`, logs the event and in case of a mouse event logs a little more:

```
procedure TMainForm.DoGeneralEvent(Event: TAbstractEvent);
begin
  LogEventData(Event);
  If Event is TFresnelMouseEvent then
    LogMouseEvent(Event as TFresnelMouseEvent,False);
end;
```

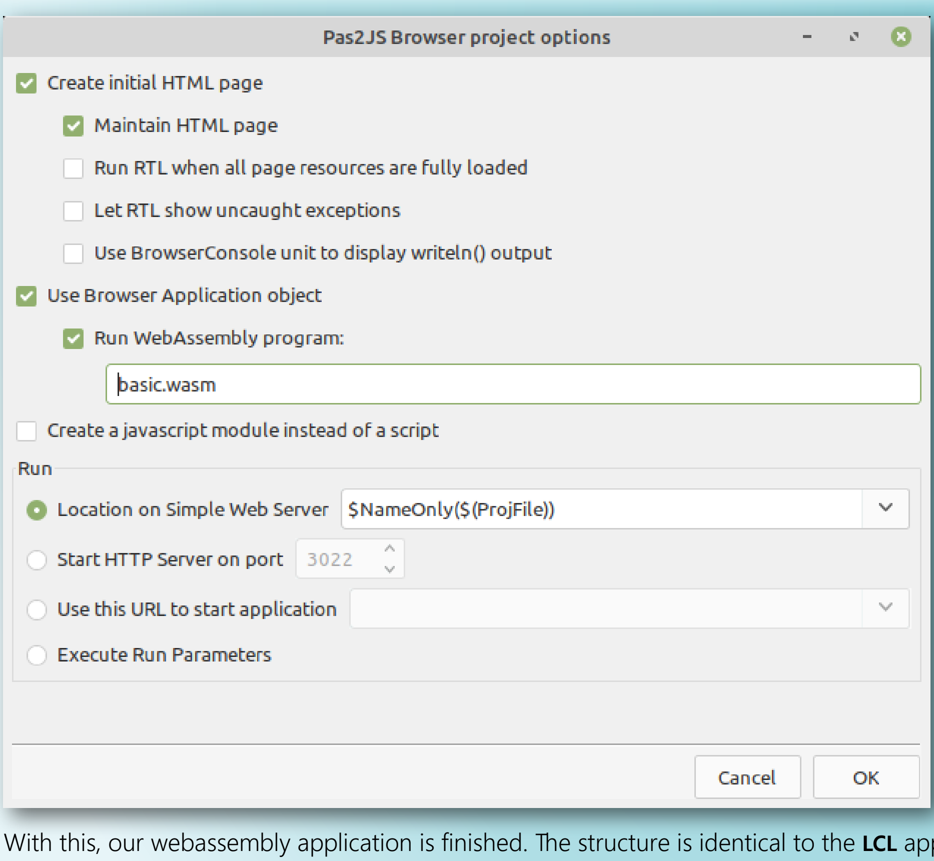
The logging events do little more than writing the event data to standard output:

```
procedure TMainForm.LogEventData(Event: TAbstractEvent);
const
  Fmt = 'Event class %s type: %s, sender : %s';
var
  S : String;
begin
  if Event.Sender=Nil then
    S:='(Nil)'
  else
    begin
      S:=Event.Sender.ClassName;
      if Event.Sender is TComponent then
      S:=TComponent(EVent.Sender).Name+' ('+S+')';
    end;
  Application.Log(etInfo,Fmt,[Event.ClassName, Event.EventName, S]);
end;
```

```
procedure TMainForm.LogMouseEvent(Event: TFresnelMouseEvent; LogData : Boolean);

const
   Fmt = 'Mouse Event (X: %f, Y: %f, Button: %s, Buttons: %s) ';
var
   Btn,Btns : String;
begin
   If LogData then
     LogEventData(Event);

   Btn  :=GetEnumName(TypeInfo(TMouseButton),Ord(Event.Button)) ;
   Btns :=SetToString(PTypeInfo(TypeInfo(TMouseButtons)),Longint(EVent.Buttons),True);
   Application.Log(etInfo,Fmt, [Event.ControlX,Event.ControlY,Btn,Btns]);
end;
```

Note the use of **RTTI** to convert button enumerated and sets to actual button names. Some event handlers have a typed version of the parameter, as can be seen in the `DoMouseMove` event handler, which receives a `TFresnelMouseEvent` parameter:

```
procedure TMainForm.DoClick(Event: TAbstractEvent);
begin
  Application.Log(etInfo,'You clicked '+(Event.Sender as TComponent).Name);
end;

procedure TMainForm.DoMouseMove(Event: TFresnelMouseEvent);
begin
  LogMouseEvent(Event,True)
end;
```


Pas2JS Browser project options

☑ Create initial HTML page
☑ Maintain HTML page
☐ Run RTL when all page resources are fully loaded
☐ Let RTL show uncaught exceptions
☐ Use BrowserConsole unit to display writeln() output
☑ Use Browser Application object
☑ Run WebAssembly program:
basic.wasm
☐ Create a javascript module instead of a script

Run
● Location on Simple Web Server $NameOnly($(ProjFile))
○ Start HTTP Server on port 3022
○ Use this URL to start application
○ Execute Run Parameters

Figure 5: Creating a loader application

With this, our webassembly application is finished. The structure is identical to the **LCL** application. The differences (*events*, *styles*) were simply additions to the code in the regular **LCL** application.

❾ THE WEBASSEMBLY LOADER

To run our **webassembly** program in the browser, we need a **Javascript** program that loads the **webassembly** in the browser, provides it with the image file and finally that provides the **Fresnel** canvas.
To this end, we create a **'Web Browser application'** in the IDE.
In the dialog that appears, we check the '**Use Browser Application object**' and **'Run Webassembly program**' options and enter a filename, as in *figure 5 on page 12*.
Setting these options will create a skeleton project which we can adapt to our needs. We'll rename the application class to `TFresnelHostApplication`. The wizard will have added in the `DoRun` method a call to `StartWebAssembly` with the filename we entered. We'll need to change that.

The first thing to do is to provide the necessary **Fresnel API methods** to the **webassembly**.
The **PAS2JS WebAssembly** hosting environment has a mechanism to do this:

to provide APIs to a webassembly module, a descendant of the `TImportExtension` class must be created and instantiated.
Such a descendant has been made for the **Fresnel API**, a class called **`TWasmFresnelAPI`**.
This class is implemented in the **`fresnel.pas2js.wasmapi`**, part of the **`P2jsfresnelapi`** package.

All that we need to do is to create an instance of the `TWasmFresnelAPI` class, passing the **WASI** environment to the constructor. We do this in the constructor of our application class:

```
constructor TFresnelHostApplication.Create(aOwner: TComponent);
begin
  inherited Create(aOwner);
  FFresnelApi:=TWasmFresnelApi.Create(WasiEnvironment);
  FFresnelAPI.LogAPICalls:=True;
  FFresnelAPI.CanvasParent:=TJSHTMLElement(document.getElementById('desktop'));
  RunEntryFunction:='_initialize';
end;
```

Since our **Webassembly** module is a library, the function to execute when running it, is not the usual start as for a program, but **initialize**, which simply executes the initializations sections of the units included in the library and the main library routine.
We set 2 properties on the **`FresnelAPI`** instance

❶. We choose to log the API calls (*Every API call is logged to the screen*)
❷ We set the parent element for the canvas: for every Fresnel form, a HTML canvas is allocated.
All these canvases are positioned below the **`CanvasParent`** element.

❿ FILESYSTEM SUPPORT FOR WEBASSEMBLY

The **Fresnel** application loads an image from file using the usual **Object Pascal** file handling mechanisms.
How can we provide this file ?
The **WASI** standard provides all the **API** calls to open files and read data from files, as well as directory listing mechanisms.
It is up to the hosting environment to provide an implementation of these calls.
The **Pas2JS webassembly** hosting environment has implemented these API calls, and uses a plugin mechanism to handle the actual reading from file.
The browser offers a standardized API to access the computer's filesystem in a sandboxed manner:
`https://developer.mozilla.org/en-US/docs/Web/API/FileSystem`

⑩ FILESYSTEM SUPPORT FOR - CONTINUATION 1

This basically reserves a (*hidden*) directory for use of your web application. Your application can only access files and directories inside this directory, and these directories are **private** to each webpage This **API** would seem ideal to provide a filesysem to a **webassembly**. However, there is a catch: the filesystem **API** is an **asynchronous API**. The **WASI API** is **synchronous**, and this means that currently, the **filesystem API** is not usable. So something else must be found. Before the **FileSystem API** was generally available, a pure **Javascript** implementation of a **FileSystem** emulation was created, called **BrowserFS**. It was modeled after the **NodeJS filesystem API.**
This implementation is now known as **ZenFS**:
https://github.com/zen-fs
The **ZenFS API** comes with various backends that are synchronous:
InMemory : Stores files **in-memory**. This is cleared when the webpage is closed.
WebStorage : Stores files in **local or session** storage.

This means the filesystem can be persisted, even when the webpage is closed.
PAS2JS comes with the necessary units to make use of this **API**, and here is a plugin for the **WebAssembly** hosting mechanism to provide a filesystem. So, how to use the **ZenFS** filesystem to provide an image file to the **webassembly** module? Before starting the **webassembly**, we load the necessary files from the webserver, and store them in our in-browser filesystem emulation.
Since loading the files from the server is **asynchronous**, this loading needs to be completed **before we can start the webassembly**. The **ZenFS** filesystem needs to be initialized. This initialization is also **asynchronous**, so we must wait for it to complete **before we can start our webassembly program.** To make our life a little easier, we will introduce an **asynchronous** method:

```
procedure RunWasm ; async;
```

This means we can use await in the **RunWasm** method to let the filesystem initialization finish before calling **StartWebAssembly**. The code from the **DoRun** method generated by the application wizard in **Lazarus** is replaced with the following:

```
procedure TFresnelHostApplication.DoRun;
begin
   RunWasm;
end;
```

The actual work now happens in **RunWasm**, which starts by initializing the **ZenFS** file system. The initialization means that you tell **ZenFS** where to mount various file systems, similar to the way this happens on a typical unix or linux operating system. You can use various filesystems at the same time, but for our needs, we'll mount a single filesystem using **WebStorage**:

```
procedure TFresnelHostApplication.RunWasm;
var
  aCount : Integer;
begin
  Terminate;
  await(tjsobject, ZenFS.configure(
    new(
      [’mounts’, new([
        ’/’, DomBackends.WebStorage
        ])
      ])
    )
  );
  FS:=TWASIZenFS.Create;
  WasiEnvironment.FS:=FS;
  aCount:=await(LoadFiles);
  Writeln(’Loaded ’,aCount,’ files.’);
  StartWebAssembly(’basic.wasm’,true,@OnBeforeStart,@OnAfterStart);
end;
```

⑩ FILESYSTEM SUPPORT FOR - CONTINUATION 2

After the **ZenFS** filesystem is initialized, we create an instance of the **WASIZenFS** class, and assign it to the **WasiEnvironment**. Before starting the **webassembly**, we load the needed files into our **virtual filesystem** using **`LoadFiles`**. As mentioned before, this is an **asynchronous** call, so we wait for it to complete.
Lastly, the **webassembly** is started, **specifying 2 callbacks:** one to be executed before, one to be executed after the start of the webassembly.

Before diving into these calls, let's see how we can load files into our browser -based filesystem. The **`TWasiHostApplication`** class offers 3 calls to preload files from the server into the filesystem emulation:

```
function PreLoadFiles(aFiles : TPreLoadFileDynArray) : TPreLoadFilesResult; async;
function PreLoadFiles(aFiles : Array of string) : TPreLoadFilesResult; async;
function PreLoadFilesIntoDirectory(aDirectory: String;
                                              aFiles: array of string): TPreLoadFilesResult; async;.
```

As you can see, all calls are **asynchronous**. The first call is the raw download mechanism. You specify the files to preload using an array of records:

```
TPreLoadFile = record
  url : String;
  localname : string;
end;
```

The URL contents will be downloaded and put into the **local filesystem as a file with the given path and name**. *(NOTE that if you specify directories, you must create any directories before loading files into them*) The second form of the **PreLoadFiles** call accepts an **array of strings**.
This should be an even amount of strings, where each pair is a **URI** and a **local filename:** these are simply transformed into an array of **`TPreLoadFile`** records. The **`PreLoadFilesIntoDirectory`** is a utility function that stores all downloaded files in a single directory. The **`LoadFiles`** function uses this latter utility function, and is really simple:

```
function TFresnelHostApplication.LoadFiles: Integer;

const
   Files : TStringDynArray = ('image.png','style1.css','style2.css');

var
   Res: TPreLoadFilesResult;
   I : Integer;
begin
   result:=-1;
   Res:=await (PreloadFilesIntoDirectory('/',Files));
   For I:=0 to Length(Res.failedurls)-1 do
     With Res.failedurls[i] do
        Writeln('Failed to preload file: ',url,' : ',error);
   Result:=res.loadcount;
end;
```

The **`TPreLoadFilesResult`** gives info about the number of loaded files and any errors that may have occurred.
All that remains to be discussed are the 2 callbacks that were passed to the **`StartWebassembly`** call.
The **`OnBeforeStart`** event is called before the **`webassembly`** is started, and we use it to pass the functions that are exported from the **`webassembly`** to the **Fresnel API:**

```
function TFresnelHostApplication.OnBeforeStart(Sender: TObject;
                                     aDescriptor: TWebAssemblyStartDescriptor): Boolean;
begin
   FFresnelApi.InstanceExports:=aDescriptor.Exported;
   Result:=true;
end;
```

⑩ FILESYSTEM SUPPORT FOR - CONTINUATION 3

The fresnel API needs access to the exported functions in order to call the timer `function_fresnel_tick`. This is exactly why the `OnAfterStart` event handler is needed: once the **webassembly** module has been initialized, we start the fresnel timer:

```
procedure TFresnelHostApplication.OnAfterStart(Sender: TObject;
                    aDescriptor: TWebAssemblyStartDescriptor);
begin
  Writeln(’Starting timer’);
  FFresnelApi.StartTimerTick;
end;
```

Last but not least, we need the main program code to set the ball rolling. This looks like any **Free Pascal or Lazarus** code, with a small addition to set up the console output: the **WriteLn** statements from the **webassembly** are caught and displayed in the browser console log, but also in a special HTML element (*with id ”pasjsconsole”*):

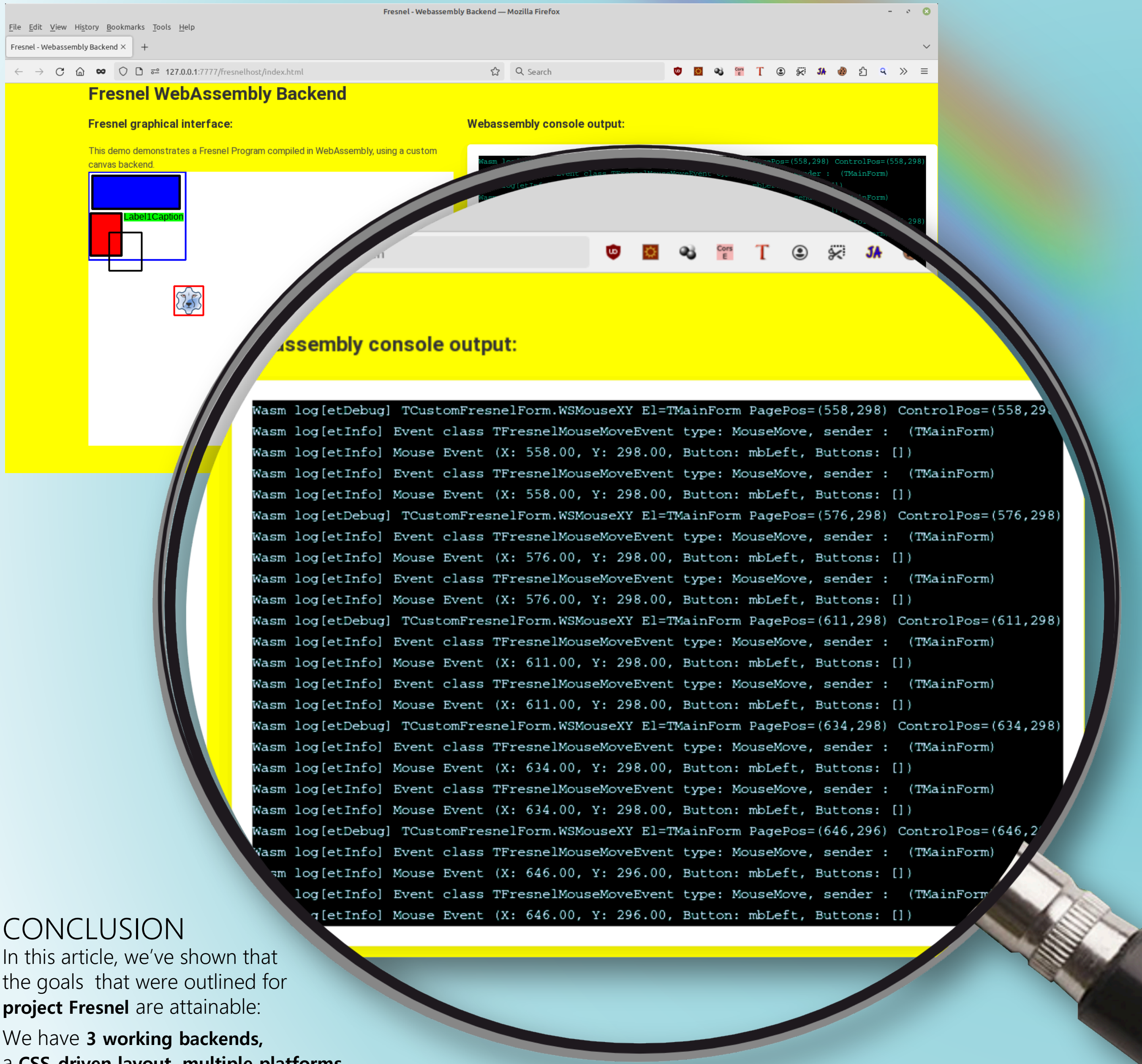
```
var
  Application : TFresnelHostApplication;
begin
  ConsoleStyle:=DefaultCRTConsoleStyle;
  HookConsole;
  Application :=TFresnelHostApplication.Create(nil);
  Application.Initialize;
  Application.Run;
end
```

With this, the loader program is finished. All we need now is a HTML page which will execute the code. We need 2 special tags in the HTML: one is the parent for the canvas (*with id ”desktop”*) and one is needed to display the output of the `WriteLn` statement. We also need to load **ZenFS** : 2 **Javascript** files are needed: **`browser.min.js.`** This is the core **ZenFS** module. **`browser.dom.fs.`** This is the **ZenFS** module that allows to store files in the browser local storage. Add some **CSS** styling with **Bulma CSS** to the mix, and this is our web page:

```
<!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>Fresnel - Webassembly Backend</title>
   <link href="bulma.min.css" rel="stylesheet">
   <script src="browser.min.js"></script>
   <script src="browser.dom.js"></script>
   <script src="fresnelhost.js"></script>
</head>
<body style="background-color: yellow">
   <div class="container">
      <h1 class="title is-3">Fresnel WebAssembly Backend</h1>
      <div class="columns">
         <div class="column">
            <h1 class="title is-5">Fresnel graphical interface:</h1>
            <p>This demo demonstrates a Fresnel Program compiled in WebAssembly,
                using a custom canvas backend.</p>
            <div id="desktop" style="min-height: 480px;">
            </div>
         </div>
         <div class="column">
            <h1 class="title is-5">Webassembly console output:</h1>
            <div class="box" id="pasjsconsole"></div>
         </div>
      </div>
   </div>
   <script>
      rtl.showUncaughtExceptions=true;
      window.addEventListener("load", rtl.run);
   </script>
</body>
</html>
```

⑩ FILESYSTEM SUPPORT FOR - CONTINUATION 4

When all this is loaded in the browser, the application will look like figure 6 on page 17. Note the yellow background on the HTML body. This is done to demonstrate clearly that the fresnel background (white) is observed when showing an image with transparency, such as the lazarus icon.



Fresnel - Webassembly Backend — Mozilla Firefox
File Edit View History Bookmarks Tools Help
Fresnel - Webassembly Backend
127.0.0.1:7777/fresnelhost/index.html
Search
Label1Caption

Fresnel WebAssembly Backend

Fresnel graphical interface:

This demo demonstrates a Fresnel Program compiled in WebAssembly, using a custom canvas backend.

Webassembly console output:

```
Wasm log[etDebug] TCustomFresnelForm.WSMouseXY El=TMainForm PagePos=(558,298) ControlPos=(558,29
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 558.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 558.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etDebug] TCustomFresnelForm.WSMouseXY El=TMainForm PagePos=(576,298) ControlPos=(576,298)
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 576.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 576.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etDebug] TCustomFresnelForm.WSMouseXY El=TMainForm PagePos=(611,298) ControlPos=(611,298)
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 611.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 611.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etDebug] TCustomFresnelForm.WSMouseXY El=TMainForm PagePos=(634,298) ControlPos=(634,298)
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 634.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
Wasm log[etInfo] Mouse Event (X: 634.00, Y: 298.00, Button: mbLeft, Buttons: [])
Wasm log[etDebug] TCustomFresnelForm.WSMouseXY El=TMainForm PagePos=(646,296) ControlPos=(646,2
Wasm log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm)
 sm log[etInfo] Mouse Event (X: 646.00, Y: 296.00, Button: mbLeft, Buttons: [])
    log[etInfo] Event class TFresnelMouseMoveEvent type: MouseMove, sender :  (TMainForm
     [etInfo] Mouse Event (X: 646.00, Y: 296.00, Button: mbLeft, Buttons: [])
```

CONCLUSION

In this article, we've shown that the goals that were outlined for **project Fresnel** are attainable:

We have **3 working backends,**
a **CSS-driven layout, multiple platforms,**
a powerful event mechanism. With the Skia renderer available,
there should be no problem to create a universal graphical application which **runs on all native platforms and in the browser.** All this using a **single codebase**, and **running at native speed.** And obviously, all this using your favourite programming language: **Object Pascal.**

APPENDIX

code can be downloaded here
`https://gitlab.com/freepascal.org/lazarus/fresnel`

Fresnel
This repository contains the sources for Project Fresnel

What is project Fresnel ?
Project Fresnel is a new UI paradigm* for Lazarus projects.
Instead of using LCL controls, CSS-based custom drawn controls will be used to create your UI.

Why is this project needed ?
The design of the **VCL** and **LCL** is old. In the browser, **UX (***User eXperience***)** and **UI** have evolved far beyond what the **LCL** has to offer, largely thanks to the strength of **CSS**.
The choice for **CSS** as a mechanism for lay outing and display is therefore logical.
This will also allow to reuse existing **CSS** frameworks.

What's with the name ?
UI is about look and feel. Look and feel means light.
FRESNEL is a French Physicist who made important contributions to the wave theory of light.

Goals of project Fresnel:

100% Pascal code.
Create a **set of controls** that are independent of the **LCL**.
The **layout and look** of the controls are governed by **CSS**.
A **Lazarus** application must be able to run **LCL** forms alongside **'Fresnel'** forms.
This will ensure easy porting.
Different drawing backends must be possible.

Skia:
To use Skia4Delphi you must put the library into the library path:

```
Linux 64bit: export LD_LIBRARY_PATH=fresnel/
                    bin/Binary/Shared/Linux64
Macos X64: export DYLD_LIBRARY_PATH=fresnel/
                    bin/Binary/Shared/OSX64
```

For more insight in Fresnel see also the article of
Issue 107/108 Page 65 of Blaise Pascal Magazine

In science and philosophy, a paradigm is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitute legitimate contributions to a field. The word paradigm is Greek in origin, meaning "pattern."

LAZARUS HANDBOOK 2
LAZARUS HANDBOOK 1
LAZARUS HANDBOOK
FOR PROGRAMMING WITH FREE PASCAL
1
LAZARUS HANDBOOK
FOR PROGRAMMING WITH FREE PASCAL AND LAZARUS
PDF
934 PAGES
BLAISE PASCAL MAGAZINE 117
LAZARUS FACTORY
FRESNEL

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

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

BLAISE PASCAL MAGAZINE

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

Blaise Pascal

COMPONENTS 4 DEVELOPERS

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

COMPONENTS 4 DEVELOPERS

D11