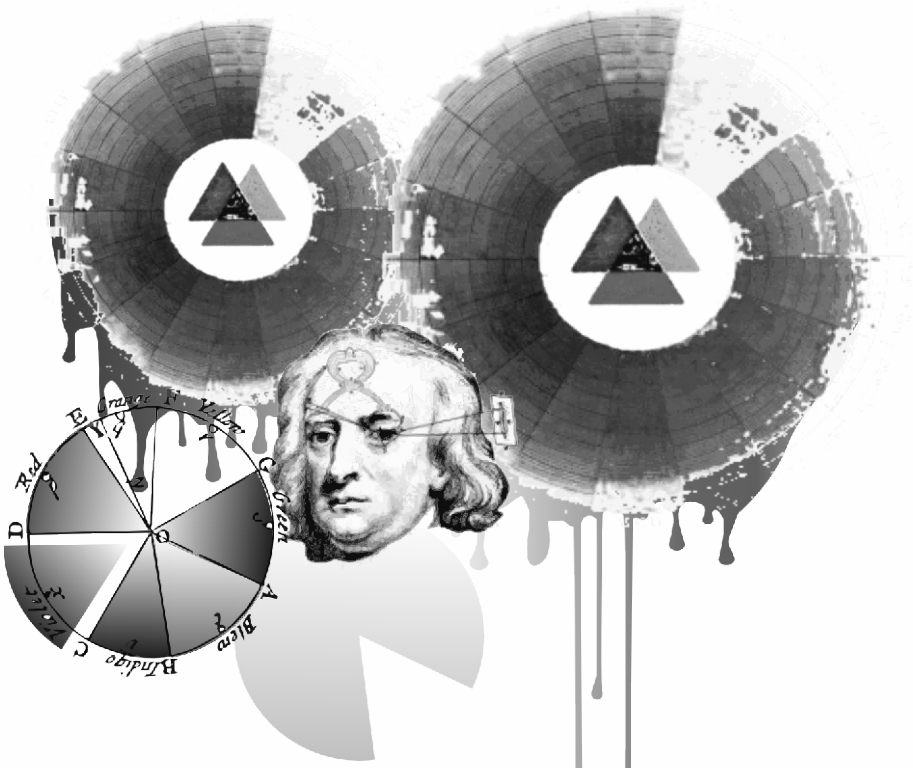
BLAISE PASCAL MAGAZINE 105

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



CIFAR-10 Image Classifier Maxbox
Lorentz waterwheel simulator
Viewing PDF files in the browser using Pas2Js
Hacking / Sata cable as antenna
Design-time components for Pas2js
Installing a Postgres Database
kbmMW WebSockets

CONTENT

ARTICLES

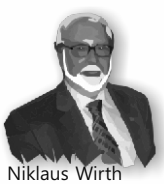
The Newton disc, also known as the Disappearing Colour Disc, is a well-known physics experiment with a rotating disc with segments in different colors (usually Newton's primary colors: red, orange, yellow, green, blue, indigo, and violet or ROYGBIV) appearing as white (or off-white or gray) when it spins very fast.

This type of mix of light stimuli is called temporal optical mixing, a version of additive-averaging mixing.[1] The concept that human visual perception cannot distinguish details of high-speed movements is popularly known as persistence of vision.

The disc is named after Isaac Newton. Although he published a circular diagram with segments for the primary colors that he had discovered, it is uncertain whether he actually ever used a spinning disc to demonstrate the principles of light.

Transparent variations for magic lantern projection have been produced.

ADVERTISERS


Niklaus Wirth

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

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

Stephen Ball
http://delphiaball.co.uk
@DelphiABall

Dmitry Boyarintsev
dmitry.living @ gmail.com

Michaël Van Canneyt,
michael @ freepascal.org

Marco Cantù
www.marcocantu.com
marco.cantu @ gmail.com

David Dirkse
www.davdata.nl
E-mail: David @ davdata.nl

Benno Evers
b.evers @ everscustomtechnology.nl

Bruno Fierens
www.tmssoftware.com
bruno.fierens @ tmssoftware.com

Holger Flick
holger @ flixments.com

Mattias Gärtner
nc-gaertnma@netcologne.de

Max Kleiner
www.softwareschule.ch
max @ kleiner.com

John Kuiper
john_kuiper @ kpnmail.nl

Wagner R. Landgraf
wagner @ tmssoftware.com

Vsevolod Leonov
vsevolod.leonov@mail.ru

Andrea Magni www.andreamagni.eu
andrea.magni @ gmail.com
www.andreamagni.eu/wp

Paul Nauta PLM Solution Architect
CyberNautics
paul.nauta @ cybernautics.nl

Kim Madsen
www.component4developers.com

Boian Mitov
mitov @ mitov.com

Jeremy North
jeremy.north @ gmail.com

Detlef Overbeek - Editor in Chief
www.blaisepascal.eu
editor @ blaisepascal.eu

Howard Page Clark
hdpc @ talktalk.net

Heiko Rompel
info @ rompelsoft.de

Wim Van Ingen Schenau -Editor
wisone @ xs4all.nl

Rik Smit
rik @ blaisepascal.eu

Bob Swart
www.eBob42.com
Bob @ eBob42.com

B.J. Rao
contact @ intricad.com

Daniele Teti
www.danieleteti.it
d.teti @ bittime.it

Anton Vogelaar
ajv @ vogelaar-electronics.com

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 (2022 prices)

	Internat. excl. VAT	Internat. incl. 9% VAT	+Shipment	TOTAL
Printed Issue ±60 pages	€ 155,96	€ 170	€ 100,00	€ 270,00
Electronic Download Issue 60 pages	€ 64,20	€ 70		€ 70

Member and donator of
Member of the Royal Dutch Library
KB KONINKLIJKE BIBLIOTHEEK

WIKIPEDIA

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

From your editor

Dear Reader,
We can announce quite some good news:
We have almost finished the online Blaise Pascal Magazine Library as PDF.
With a lot of extra properties, which will become our future service.
That means that you can - after login - always make use of this browser and all of its capabilities. It will probably launched next month.

In this issue we have an article about the creation of a simple version for your own use, complete with sources.
We chose this because usually a PDF editor's quality is very dependant on the OS and the creator. In this case we use the chrome version which works in all browsers: Linux / Mac and Windows.

To go on about web development Michael van Canneyt has designed a number of design time web components so you can now built your website and create a blog in Pascal even with the use of a database we chose Postgres.
It is kept very basic so all of you can start it: made in PAS2JS.
The installation of Postgres is also explained.
Postgres is a free Database and you can simply download it.

Max Kleiner explains about the CIFAR Image classifier as an ongoing subject about AI.
AI is to become more and more interesting and I am thinking of writing more articles about this.
Then last but not least:
at this moment Delphi 11.2 was released. Ill write about that in the next item.
For Lazarus we will publish or Road Map and I can assure you:
you'll be surprised!
Enjoy your readings and tryouts...

Detlef

"Every thing these days needs a password.
I think I pulled a muscle in my brain."

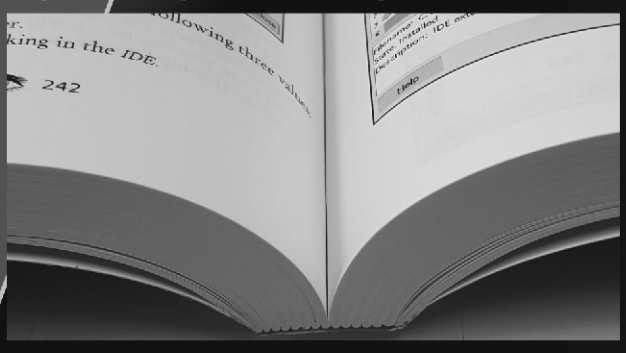
Sale Lazarus Handbook Pocket

Price: € 26,50 Excluding VAT and Shipping

- English
- Printed black & white
- 2 Volumes
- PDF included
- 934 Pages
- Weight: 2kg
- Shipment included
- Extra protected
- Including 40 Example projects and extra programs

- **BlaisePascalMagazine PDF viewer included**

printed on FSC paper
https://fsc.org/en/forest-management-certification

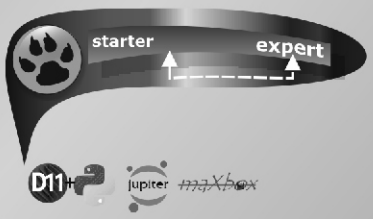


https://www.blaisepascalmagazine.eu/product-category/books/

CNN PIPELINE REPORT

ABSTRACT: This machine learning tutor explains training the so called CIFAR-10 Image Classifier with loading and testing a pre-trained model.

The pre-trained model is:
SimpleSeparableImageClassifier124_50_2.nn



INTRODUCTION

This machine learning tutor explains training the so called CIFAR-10 Image Classifier with loading and testing a pre-trained model.

The pre-trained model is: **SimpleSeparableImageClassifier124_50_2.nn**

This command line tool and script runs the CAI Network with CIFAR10 files; Utility to load cifar-10 image data into training and test data sets based on the script. Download the cifar-10 (python) version dataset from here, and extract the cifar-10-batches-1..5 folder into the same directory as the script (for validating use only test_batch.bin).

```
CIFAR-10 and CIFAR-100 datasets (toronto.edu)
```

Here are the classes in the dataset, as well as 10 random images from each

Note about to build **SimpleSeparableImageClassifier**

The code contains example usage, and runs under **Python3, JupyterNotebook, Lazarus** and **maXbox4.** Note that the load_cifar_10_data() function has the option to load the images as negatives using negatives=True. There are 50000 training images and 10000 test images.
For the first script `1135_Cifar10SeparableConvolution_50.pas` you need the 50000 training images `(cifar-10-batches-1..5)`.

For the second script 1135_uvisualcifar10test_mX4_1.pas you need only the 10000 test images.

So we do have the following 10 files as our pipeline:

```
data_batch_1.bin (30,730,000 bytes)
data_batch_2.bin (30,730,000 bytes)
data_batch_3.bin (30,730,000 bytes)
data_batch_4.bin (30,730,000 bytes)
data_batch_5.bin (30,730,000 bytes)

1135_Cifar10SeparableConvolution_50.pas      to train model (~10 hours)
SimpleSeparableImageClassifier124_50_2.nn    output pre-trained model
SimpleSeparableImageClassifier124_50_2.csv   result report

test_batch.bin      (30,730,000 bytes)       input to script 1135_uvis..
SimpleSeparableImageClassifier124_50_2.nn    input to script
1135_uvisualcifar10test_mX4_1.pas            to test and evaluate model
```

We start with a short introduction to the **CIFAR-10 Separable CNN Classification Example SimpleSeparable** (comments in code).

As I said you don't need to run the first script `(about 9 hours and 2 GByte RAM)` because you can experiment and explore **direct with the second script** and the pre-trained model `*.nn` and direct test with a `TestBatch()` procedure.

Adding neurons and neuronal layers is often a possible way to improve artificial neural networks when you have enough hardware and computing time.

In the case that you can't afford time, parameters and hardware, you'll look for efficiency with Separable Convolutions (SCNN).

This is what the first script does:

```
NN.DebugWeights();
NN.DebugStructure();
WriteLn('Layers: '+itoa( NN.CountLayers()));
WriteLn('Neurons: '+itoa( NN.CountNeurons()));
WriteLn('Weights: '+itoa( NN.CountWeights()));

CreateCifar10Volumes(ImgTrainingVolumes, ImgValidationVolumes,
                                          ImgTestVolumes, csEncodeRGB);

NeuralFit:= TNeuralImageFit.Create;
NeuralFit.FileNameBase:= 'SimpleSeparableImageClassifier124_50_2';
NeuralFit.InitialLearningRate:= 0.001; //0.001
NeuralFit.LearningRateDecay:= 0.1 //0.01;
NeuralFit.StaircaseEpochs:= 10;
NeuralFit.Inertia:= 0.9;
NeuralFit.L2Decay:= 0.0001; //0.00001;
NeuralFit.Fit(NN, ImgTrainingVolumes, ImgValidationVolumes,
    ImgTestVolumes, {NumClasses=}10, {batchsize=}128, {epochs=}50);
NeuralFit.Free;
```

As you can see the output is the NeuralFit.FileNameBase. As can be seen on above script, a separable convolution is a composition of 2 building blocks:

- A depth-wise convolution followed by
- a point-wise convolution.

And we can see the progress during learning rate:
Epochs: 40 Examples seen:1600000 Test Accuracy: 0.7010 Test Error: 0.8364 Test Loss: 0.8692 Total time: 218.73min
Epochs: 40. Working time: 3.85 hours.
Epochs: 50 Examples seen:2000000 Test Accuracy: 0.7242 Test Error: 0.7753 Test Loss: 0.8144 Total time: 292.09min
Epoch time: 3.2000 minutes. 50 epochs: 2.7000 hours.
Epochs: 50. Working time: 4.87 hours.

Now we jump to the second script as our main topic. The origin is based on a Lazarus Experiment.
`https://sourceforge.net/p/cai/svncode/HEAD/tree/trunk/lazarus/experiments/visualCifar10test/uvisualcifar10test.pas`

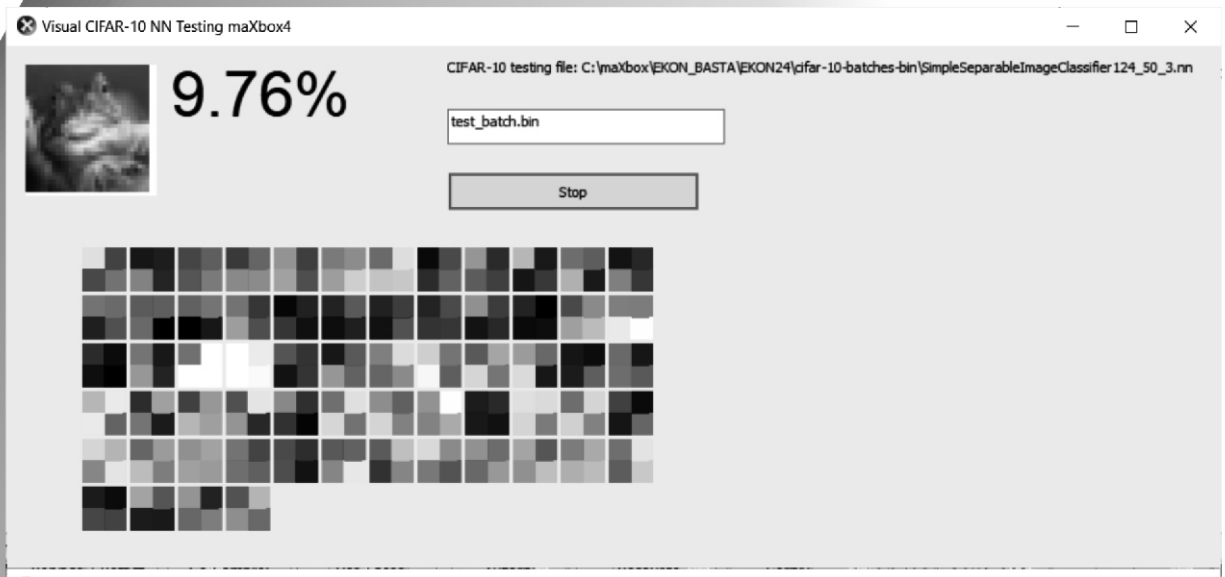
Pre-trained models means the models which have been already trained on some sort of data like cifar with different number of classes. Considering this fact, the model should have learned a robust hierarchy of features, which are spatial, rotation, and translation invariant, as we have seen before with regard to features learned by CNN models.
So the pipeline of the process goes like this:

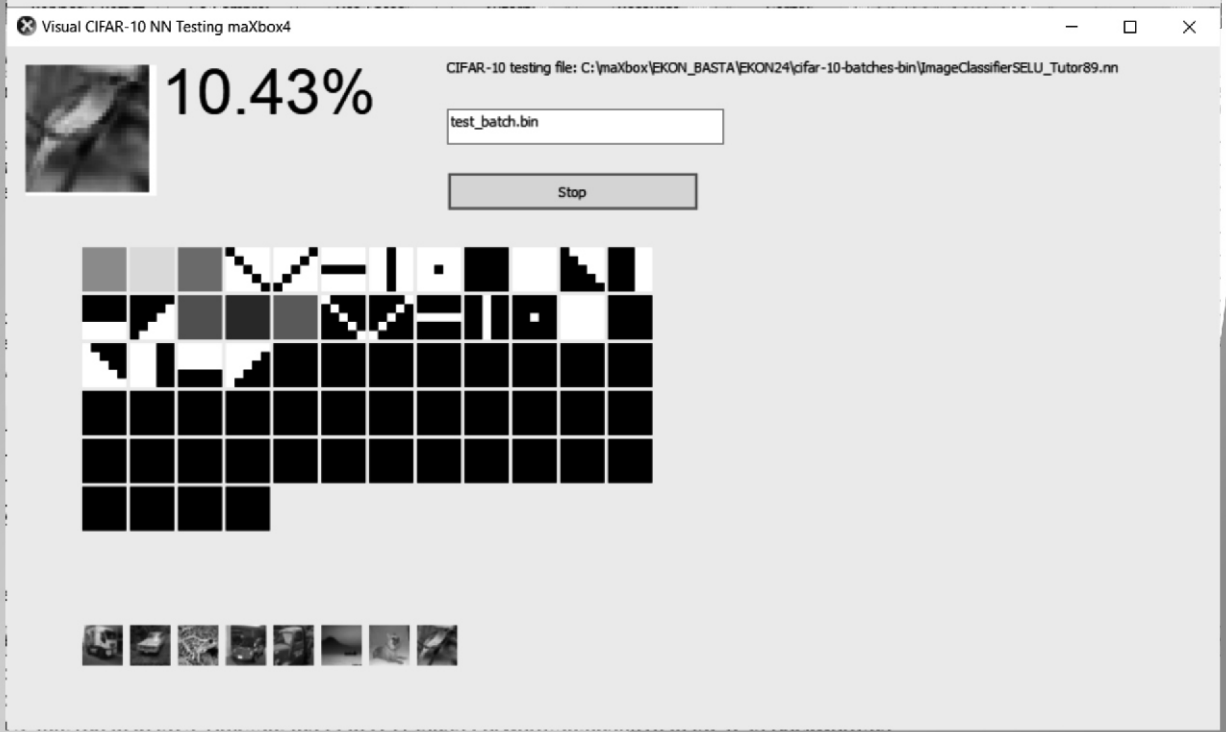
1. Train (or fit) a model to get the feature map with weights.
2. Test the classifier on unseen date with the pre-trained model.
3. Evaluate the classifier with different pre-trained models.

The last point means we can change in our second script the pre-trained model to compare the score or benchmark for better accuracy:
We can compare ImageClassifierSELU_Tutor89_5.nn with SimpleSeparableImageClassifier124_50_2.nn.

```
Const PReModel_NN = 'SimpleSeparableImageClassifier124_50_2.nn';
  //PReModel_NN = 'SimpleSeparableImageClassifier.nn';
  //PReModel_NN = 'SimpleSeparableImageClassifier124.nn';
  //PReModel_NN = 'SimpleSeparableImageClassifier124_50_3.nn';
  //PReModel_NN = 'ImageClassifierSELU_Tutor89.nn';

NN := TNNet.Create();

writeln('Creating CNeural Network...');
ImgVolumes := TNNetVolumeList.Create(true);
NumClasses := 10;

fileName:= Exepath+PReModel_NN; //OpenDialogNN.FileName;

writeln('Loading neural network from file: '+fileName);
NN.LoadFromFile(fileName);
NN.EnableDropouts(false);
firstNeuronalLayer := NN.GetFirstNeuronalLayerIdx(0);

pOutput := TNNetVolume.Create0(NumClasses,1,1,0);  //or 1
vOutput := TNNetVolume.Create0(NumClasses,1,1,0);
vDisplay:= TNNetVolume.Create0(NumClasses,1,1,0);

SetLength(aImage, NN.Layers[firstNeuronalLayer].Neurons.Count);
SetLength(sImage, NumClasses);
```

Also a view of the neurons is possible the get some insights of the power of segmentation and finally discrimination. So a pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published platforms. We use a platform process pipeline ;-).
Lets start with a simple sample to compare (evaluate) this 2 models with our second script:





So it's obvious that the 2 models are underperformed. There is a list of models and their performance and parameter count here:

```
https://keras.io/api/applications/
```

LeNet for example is the most popular **CNN** architecture it is also the first **CNN** model which came in the year 1998. LeNet was originally deve-loped to categorise handwritten digits from 0-9 of the **MNIST** Dataset. It is made up of seven layers, each with its own set of trainable para-meters.

```
  Application.ProcessMessages();
 if pOutput.GetClass() = ImgVolumes[ImgIdx].Tag then begin
  Inc(Hit);
  // WriteLn(' Tag Label: '+ itoa(ImgVolumes[ImgIdx].Tag));
 end else begin
  Inc(Miss);
 end;
```

To get a better conclusion there's a simple representation of a neural net with some attributes and a TestBatch() procedure below:

```
NN.DebugWeights();
//WriteLn(' Layers: ', NN.CountLayers() );
//WriteLn(' Neurons:', NN.CountNeurons() );
WriteLn('Neural CNN network has: ');
WriteLn(' Layers: '+ itoa(NN.CountLayers() ));
WriteLn(' Neurons: '+ itoa(NN.CountNeurons() ));
WriteLn(' Weights: '+ itoa(NN.CountWeights() ));
WriteLn('Computing...');
```

```
Neural CNN network has:        Pre-Model:
 Layers: 13                    Layers: 11
 Neurons: 205                  Neurons: 124
 Weights: 20960                Weights: 14432
Computing...
```

```
Directory of C:\maXbox\EKON_BASTA\EKON24\cifar-10-batches-bin\
```

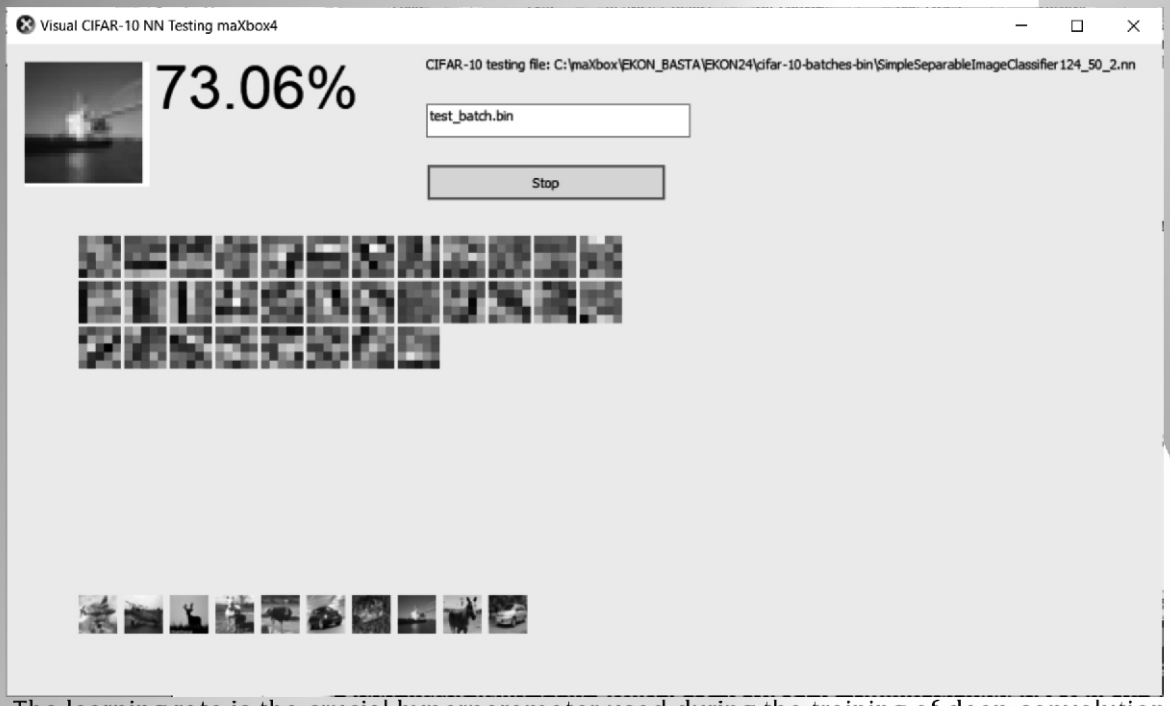
This function tests a neural network on the passed ImgVolumes:

```
{procedure TestBatch
  (
   NN: TNNet; ImgVolumes: TNNetVolumeList; SampleSize: integer;
  out Rate, Loss, ErrorSum: TNeuralFloat
  ); }

    rate:= 0; loss:= 0;
    ErrorSum:= 0;
    TestBatch(NN, ImgVolumes, 1000, rate, loss, ErrorSum);
    writeln('Test batch score: '+Format(' Rate:%.4f, Loss:%.4f, ErrorSum:%.4f ', [rate, loss, ErrorSum]));
    LabClassRate.Caption:= format('Ø %.2f%% ',[rate*100]);
end;
```

```
Ver: 4.7.6.10 (476). Workdir: C:\maXbox\EKON_BASTA\EKON24\cifar-10-batches-bin

Test batch score:  Rate:0.7130, Loss:0.1772, ErrorSum:905.7636
```

By following these ways you can
make a CNN model that has a validation set accuracy of more than 95 %
but the question is how specific is this validation.

STARTING TESTING.
Epochs: 50 Examples seen:2000000 Test Accuracy: 0.7386 Test Error: 0.7142 Test Loss: 0.7534 Total time: 595.02min
Epoch time: 7.5000 minutes. 50 epochs: 6.3000 hours.
Epochs: 50. Working time: 9.92 hours.
CAI maXbox Neural Fit Finished.
terminate__
mX4 executed: 06/07/2022 18:58:08
Runtime: 9:55:6.882 Memload: 48% use



The learning rate is the crucial hyperparameter used during the training of deep convolution neural networks (DCNN) to improve model accuracy; this you can follow in the SimpleSeparableImageClassifier124_50_2.csv.

epoch	training accur	training loss	training error	validation accur	validation loss	validation error	learning rate	time	test accurac	test loss	test error
34	0.6847	0.8597	0.8348	0.7437	0.7539	0.7168	0.0000424	24224			
35	0.679	0.8116	0.8321	0.744	0.7518	0.7156	0.0000424	24898			
36	0.679	0.9	0.8579	0.7449	0.7506	0.7143	0.0000424	25566			
37	0.6824	0.8187	0.7777	0.7444	0.7493	0.7132	0.0000424	26237			
38	0.6803	0.8277	0.8318	0.7454	0.7488	0.7129	0.0000424	26905			
39	0.6797	0.7477	0.803	0.7458	0.748	0.7126	0.0000424	27575			
40	0.6802	0.9165	0.8089	0.746	0.7478	0.7118	0.0000424	28520	0.7374	0.7564	0.719
41	0.6814	1.0169	0.9353	0.7457	0.7476	0.7113	0.0000148	29189			
42	0.6833	1.1278	0.9435	0.7458	0.7468	0.7106	0.0000148	29858			
43	0.6813	0.9797	0.9069	0.7455	0.747	0.7103	0.0000148	30527			
44	0.6828	0.9151	0.8086	0.7458	0.7465	0.7102	0.0000148	31197			
45	0.6857	1.2414	1.0189	0.7464	0.7464	0.7094	0.0000148	31873			
46	0.6834	0.76	0.7693	0.7461	0.7462	0.7091	0.0000148	32552			
47	0.6851	0.9391	0.8937	0.7457	0.746	0.7086	0.0000148	33234			
48	0.6825	0.9652	0.8961	0.746	0.7455	0.7079	0.0000148	33906			
49	0.6893	0.931	0.8649	0.7465	0.7456	0.7074	0.0000148	34654			
50	0.6853	0.7832	0.7657	0.7462	0.7453	0.7074	0.0000148	35701	0.7386	0.7534	0.7142

CONCLUSION:

The proper way to use a CNN doesn't exists.

The advice for ugly score is to use a smaller learning rate or larger batch size for the weights that are being fine-tuned and a higher one for the randomly initialized weights (e.g. the ones in the softmax classifier) TNNetSoftMax. Pre-trained weights are already good, need to be fine-tuned, not distorted.

So the process pipeline will be:

1. Train the CNN (1. script – 50000 img.)
2. Test the CNN (1. script – 10000 img.)
3. Validate CNN (1.t script – check hyperparameters)
4. Evaluate CNN (2.script– 10000 img., compare pretrained models *.nn)

```
PReModel_NN = 'SimpleSeparableImageClassifier124_50_21.nn';
PReModel_NN = 'SimpleSeparableImageClassifier124.nn';
PReModel_NN = 'SimpleSeparableImageClassifier124_50_3.nn';
PReModel_NN = 'ImageClassifierSELU_Tutor89.nn';
PReModel_NN = 'EKON25_SimpleImageClassifier-60.nn';
PReModel_NN = 'C:\maXbox\works2021\maxbox4\crypt\yolo-tiny.h5';
```

The scripts can be found:

```
http://www.softwareschule.ch/examples/1135_Cifar10SeparableConvolution_50.pas
http://www.softwareschule.ch/examples/1135_uvisualcifar10test_mX4_1.pas
```

All scripts & data:

```
https://github.com/maxkleiner/neural-api/tree/master/examples/SeparableConvolution
```

Reference:

```
https://sourceforge.net/p/cai/svncode/HEAD/tree/trunk/lazarus/experiments/
visualCifar10test/uvisualcifar10test.pas
```

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

Script Ref: `1073__CAI_3_LearnerClassifier22_Tutor_89_2.txt`

```
http://www.softwareschule.ch/examples/1073__CAI_3_LearnerClassifier22_Tutor_89_2.txt
https://entwickler-konferenz.de/blog/machine-learning-mit-cai/

https://www.freecodecamp.org/news/
convolutional-neural-network-tutorial-for-beginners/
```

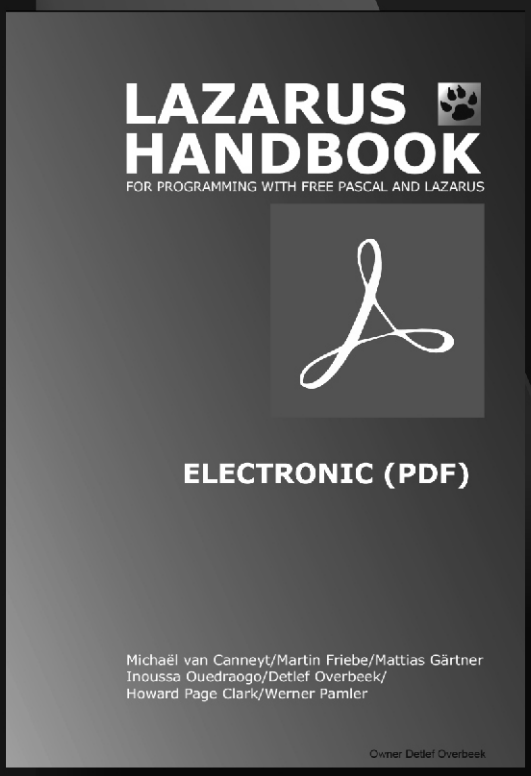
Appendix: show neurons from maXbox4 integration

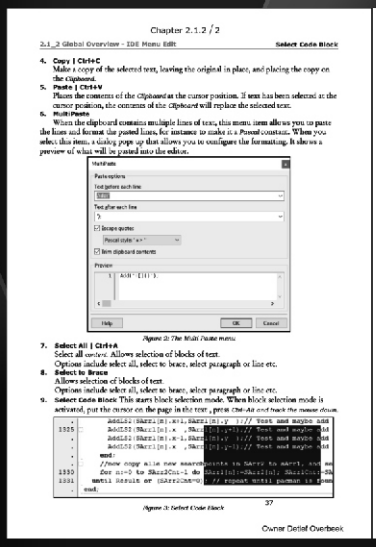
```
{*-------------------------------------------------------------------------*)
 for NeuronCount:= 0 to NN.Layers[firstNeuronalLayer].Neurons.Count- 1 do begin
    aImage[NeuronCount]:= TImage.Create(FormVisualLearning);
    aImage[NeuronCount].Parent := FormVisualLearning;
    aImage[NeuronCount].Width  :=
              NN.Layers[firstNeuronalLayer].Neurons[NeuronCount].Weights.SizeX;
    aImage[NeuronCount].Height :=
              NN.Layers[firstNeuronalLayer].Neurons[NeuronCount].Weights.SizeY;
    aImage[NeuronCount].Top    := (NeuronCount div 12) * 38 + 160;   //120
    aImage[NeuronCount].Left   := (NeuronCount mod 12) * 38 + 60;
    aImage[NeuronCount].Stretch:=true;
 end;
```

Lazarus Handbook PDF

- English
- Black & white edition
- 934 Pages
- Signed with your own name
- Electronic indexed
- Index of 240 pages
- List of chapters is completely clickable for moving to the page
- Including 40 Example projects and extra programs

- BPM PDF VIEWER for Windows included




Chapter 2.1.2 / 2

2.1_2 Global Overview - IDE Menu Edit — Select Code Block

4. **Copy | Ctrl+C**
Make a copy of the selected text, leaving the original in place, and placing the copy on the *Clipboard*.
5. **Paste | Ctrl+V**
Places the contents of the *Clipboard* at the cursor position. If text has been selected at the cursor position, the contents of the *Clipboard* will replace the selected text.
6. **Multi Paste**
When the clipboard contains multiple lines of text, this menu item allows you to paste the lines and format the pasted lines, for instance to make it a *Pascal* constant. When you select this item, a dialog pops up that allows you to configure the formatting. It shows a preview of what will be pasted into the editor.

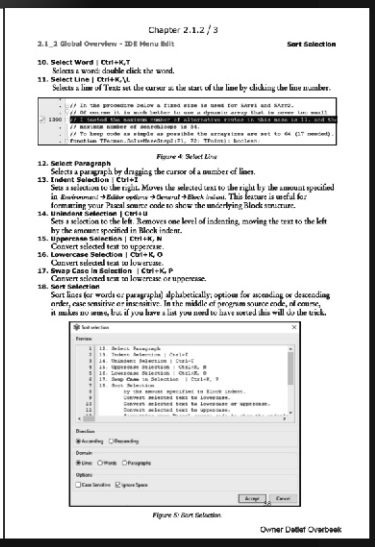
Figure 2: The Multi Paste menu.

7. **Select All | Ctrl+A**
Select all content. Allows selection of blocks of text.
Options include select all, select to brace, select paragraph or line etc.
8. **Select to Brace**
Allows selection of blocks of text.
Options include select all, select to brace, select paragraph or line etc.
9. **Select Code Block** This starts block selection *mode*. When block selection mode is activated, put the *cursor* on the page in the text , press *Ctrl-Alt and track the mouse down*.

Figure 3: Select Code Block

37

Owner Detlef Overbeek


Chapter 2.1.2 / 3

2.1_2 Global Overview - IDE Menu Edit — Sort Selection

10. **Select Word | Ctrl+K,T**
Selects a word: double click the word.
11. **Select Line | Ctrl+K,U**
Selects a line of Text set the cursor at the start of the line by clicking the line number.

Figure 4: Select Line

12. **Select Paragraph**
Selects a paragraph by dragging the cursor of a number of lines.
13. **Indent Selection | Ctrl+I**
Sets a selection to the right. Moves the selected text to the right by the amount specified in *Environment→Editor options→General→Block indent*. This feature is useful for formatting your Pascal source code to show the underlying Block structure.
14. **Unindent Selection | Ctrl+U**
Sets a selection to the left. Removes one level of indenting, moving the text to the left by the amount specified in Block indent.
15. **Uppercase Selection | Ctrl+K, N**
Convert selected text to uppercase.
16. **Lowercase Selection | Ctrl+K, O**
Convert selected text to lowercase.
17. **Swap Case in Selection | Ctrl+K, P**
Convert selected text to lowercase or uppercase.
18. **Sort Selection**
Sort lines for words or paragraphs alphabetically, options for ascending or descending order, case sensitive or insensitive. In the middle of program source code, of course, it makes no sense, but if you have a list you need to have sorted this will do the trick.

Figure 5: Sort Selection.

Owner Detlef Overbeek

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

A LORENTZ'S WATERWHEEL SIMULATOR

BY DAVID DIRKSE

D11 starter expert

INTRODUCTION

After the discovery of the laws of motion and gravity by **Isaac Newton**, the philosophy of determination emerged. Philosophers extended the laws of physics to life in general and assumed the future was perfectly predictable just like planetary orbits. Even the existence of free will was questioned.

This rigid vision was challenged by **Edward Lorentz** (*1917 to 2008*), an American mathematician. Fascinated by weather changes **Lorentz** noticed that deterministic systems may show unpredictable behavior if small differences in initial conditions cause large variations in outcome.

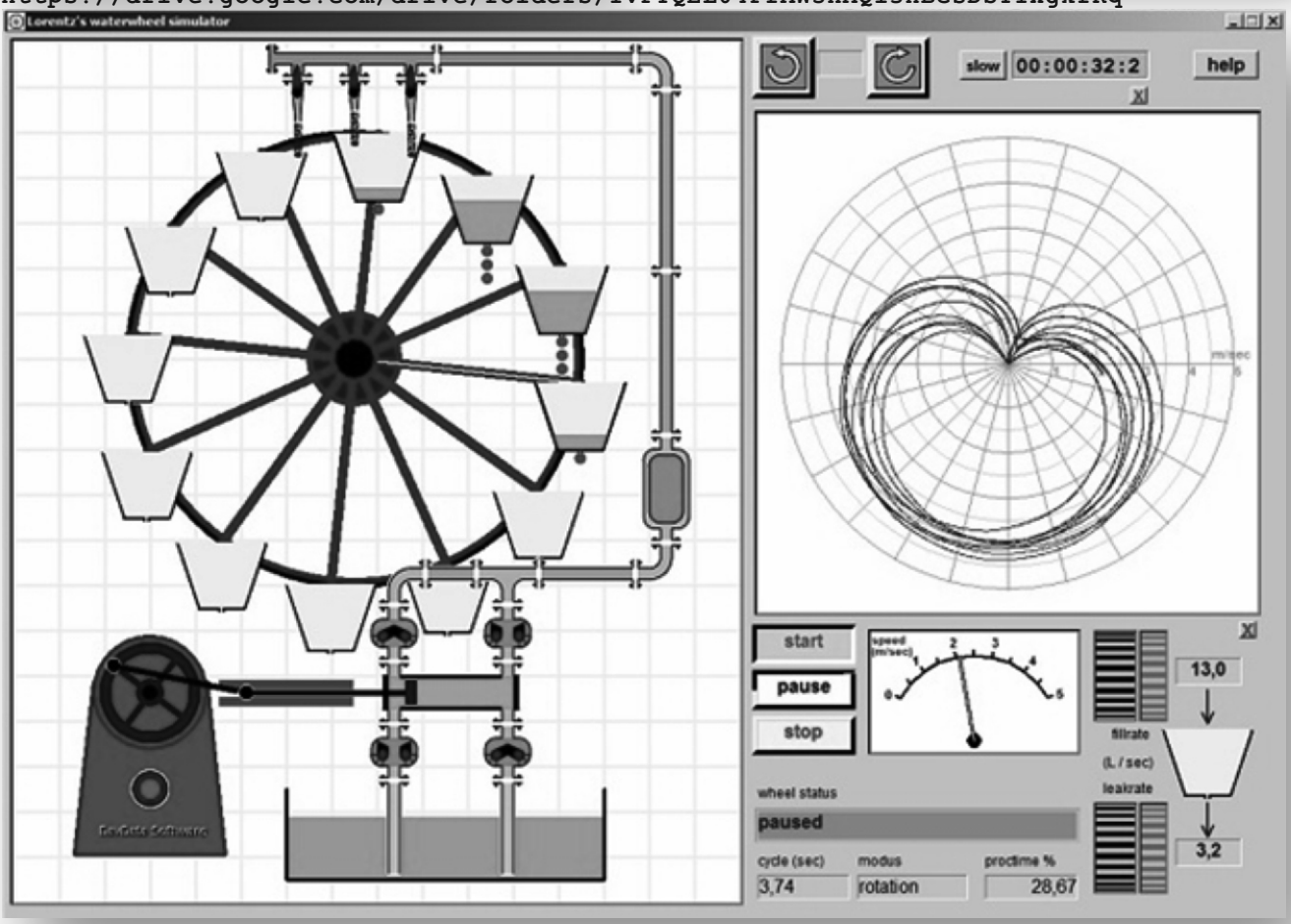
Lorentz is known for his chaos theory. Chaos is not randomness but unpredictability. The waterwheel is a simple mechanic to illustrate this principle. The wheel was constructed early 1970 by **William Markus**. A demonstration may be watched here:

This Delphi project is a waterwheel simulator.

Buckets are attached to a vertically placed wheel. Valves supply water at the top buckets. The buckets have small holes so this water is lost gradually. The result is unpredictable motion, rotation and pendulum movement alternates. However, as Lorentz discovered, chaotic movement may obey to underlying laws just like weather and climate.

Figure: 1 Below is a reduced picture:

https://drive.google.com/drive/folders/1vFTQEL04PfAW5nAQi3nBGsDSYIxgkfKq



OPERATION

See above picture.
Just above the wheel are three valves to supply water. These valves are opened and closed by a mouse click.
One spoke of the wheel is colored yellow - this is the reference spoke.
The upward position is zero degrees.
Horizontal right is 90 degrees.
Right of the wheel is a graph showing the history of wheel movement.
Two buttons, just above the graph, turn the wheel to the start position.
Right of this button the simulation time is displayed in 10th of seconds.
Pressing the slow button shows the simulation process at half speed.
Below the graph are the:

- Start button to start the pump
- Pause button to halt the process temporarily for observation
- Stop button to terminate the simulation process

Right of the speed indicator are rotation buttons to set the inlet - and leak rates (litres/sec).
At the right bottom are displayed

- Cycle time of either a rotation or pendulum swing.
- Modus: rotation or pendulum type movement.
- Proctime: the percentage of time needed by the processor to perform the simulation and display the results.

THE SIMULATION

The variables are:

❶: the position of the wheel
(which is the angle of the reference spoke, clockwise is positive)
❷:the speed of the wheel
❸ : the amount of water in each bucket
❹: the fill rate
❺ : the leak rate
❻: the opened water valves above the wheel

Variables 1,2,3 change permanently during the process.
Time is sliced in periods of one millisecond.
During this small time, the speed of the wheel and the water volume in the buckets are considered constant. Then depending on this fixed situation changes are calculated and the wheel speed, wheel position and bucket volumes are updated.
After ten of these steps, a picture of the wheel is repainted to reflect the new situation.
Next the processor waits until the clock has advanced 10 milliseconds.
So, 100 new pictures per second are displayed.
It shows that the processor needs about 2.5 to 3 milliseconds for 10 simulations plus painting and display of a new wheel and the graph.

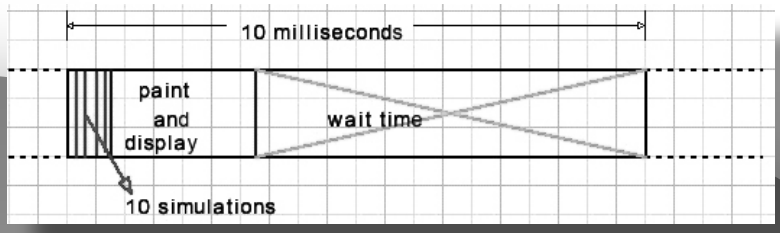


Figure: 2

The wheel diameter is 2 meters.
Buckets have the shape of topped pyramids. (measured in cm.)

A bucket volume is 28 litre.
The mass of the wheel and empty buckets is 25kg.
Originally it was set to 150kg. but it showed to be very difficult to obtain chaotic movement. Now the wheel is aluminum, the buckets are plastic instead of steel.

Accelerating forces are caused by gravity: filled buckets moving down.
Braking forces are from filled buckets that move upward, some static friction (set to 4 Newton), some air friction which is proportional to the squared velocity (Cd value set to 0.5) but mostly from the water inlet. This water has no horizontal velocity and is accelerated thus slowing down the wheel.

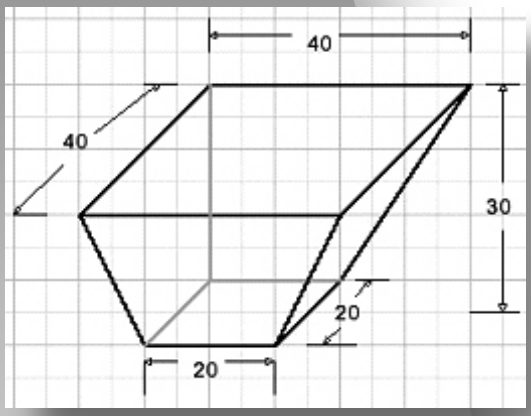


Figure: 3

α is the bucket position
W is the water in the bucket (litres)
g is the gravity acceleration on earth (9,8m/s ^ 2)
F is the driving force in Newtons.

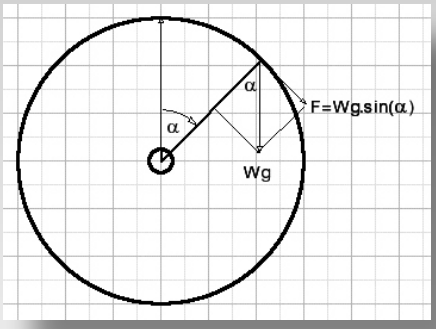


Figure 4:

Upward moving buckets automatically count negative by the sine function. Only the force perpendicular to the radius causes acceleration.

The new wheel speed is calculated in two steps:

❶ By the law of conservation of energy
For all buckets we add:
`oldmass` : the total initial water volume plus the mass of the wheel
`Leak:` the leaked water
`Overflow:` the lost water exceeding the maximum bucket volume. (*this overflow is expected to be water at wheel speed*)
`Newmass` : the final water volume plus the mass of the wheel.
`F` : the gravitational forces.

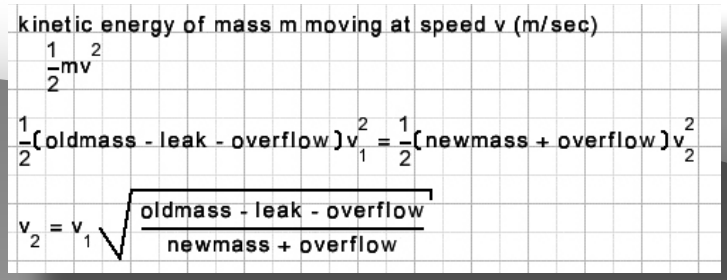


Figure: 5

Being V1 the initial velocity (*m/sec*) of the wheel :

V2 is the new wheel speed.
Note:
since the wheel radius is 1 meter, the speed at the wheel perimeter is the same as the wheel speed in radians.

2. ACCELERATING FORCES
F is first decreased by static- and dynamic frictions.
`Acceleration a = F / newmass`
The new wheel velocity V becomes
`V = V2 +at = V2 + 0.001*F/newmass` (*t is 1 millisecond)*

THE PROJECT

To show the simulation in real time requires that all calculations, painting and display operations are finished within time.

What to expect?
A stone released two meter above ground reaches the ground with a speed of 6.28m/s after 0.639 seconds.
In reality, due to braking forces, the wheel speed will not exceed 5m/sec.

Painting the wheel radius of 1 meter as `200 pixels` amounts to `5mm` distance per pixel.
Per millisecond a speed of `5000mm/second` results in a position change of `1pixel`.

At maximum speed, displaying 100 new pictures per second will show the wheel movement in 10 pixel steps.

- `Moveto[ ]` and `lineto[ ]` canvas methods are relative slow.
- The draw( ) method is fast.
- So pictures have to be prepared before as much as possible and then assembled by using the draw( ) method of the canvas .

Next is displayed the form in design time with components:

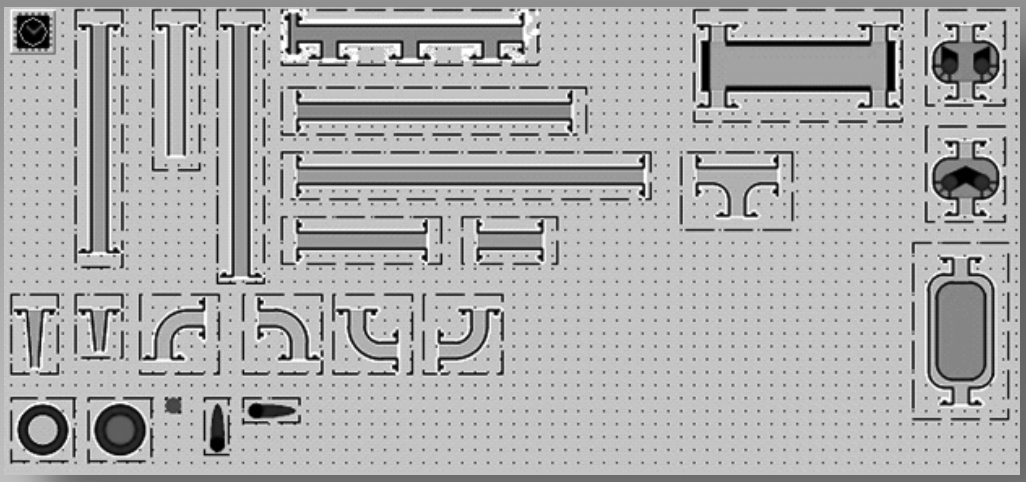


Figure: 6

Images (transparent) and a timer component that operates on the CPU clock.
These images are drawn on bitmap map on which also the wheel is painted.

The following bitmaps are created:

`Map` :	holds the wheel and pump system. Is drawn on paintbox1 to become visible.
`Motormap`:	holds the pump motor without driving wheel. Painted once at create time.
`Bucketmap` :	holds the picture of an empty bucket. Water is painted after placing this bitmap on map.
`Speedmap`:	holds the dial plate of the wheelspeed indicator. The needle is painted later.
`Jetmap`:	holds the water jet.

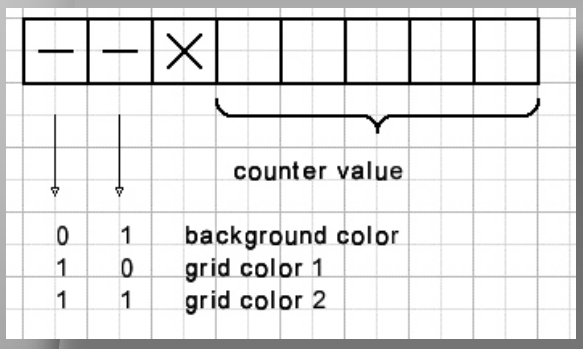
NOTE:
the pipes, pump and valves are images of Timage components.
To increase speed, the pixelformat of these images is set to pf32bit.
24bit images cause a slower draw( ) method.

Operation starts by calling `procedure Rotate` which calls `procedure simulate` 10 times to perform the wheel speed, wheel position and bucket volume updates.
`Procedure repaint` paints the wheel and buckets using the updated values.
`Control` is centralized: events call `procedure controlcenter (controlmessage)`.
`Controlcenter` calls helper `procedures startwheel, pausestop, pausego, stopwheel, movewheelleft, movewheelright, stopwheelmotion` to do the actual work.

Variables
In this article I do not show code. The lengthy paint procedures are standard stuff.
Please refer to the source code for details.

```
Var    Awheel         : single;       //in radians, clockwise is positive
       watersupply    : single = 400; //volume of water in bottom supply
       valveControl   : byte = 2;     //bits 0: left; 1:middle; 2: right open
       valveselector  : byte;         //1,2,3.. used by mouse move to select valve
       Amotorwheel    : single;       //pump motor wheel angle
       pumpspeed      : single;       //set with rotation buttons
       fillspeed      : single;       //actual water flow
       leakspeed      : single;
       simtime        : dword;        //total simulation time in msecs.
       timeout        : dword;        //counter to detect stagnation of wheel
movement
       wheelstatus    : TWheelstatus =wsStopped;
       oldcode        : byte;         //detection of motion type , see later
       refpoint       : Trefpoint;    //reference point to detect cycles, see later
```

THE POLAR GRAPH
The wheel speed is represented by the distance to the center of the plot.
The angle is the position of the reference spoke.
The graph procedures are found in unit `pgunit`.
This graph is painted in bitmap `pgmap` which is drawn on `paintbox2`
to become visible to show the history of wheel speed and movement.
To avoid a clobbered display of old data is faded after time.
This is accomplished by using bits `24..31` of the `32bit` pixels in bitmap `pgmap` to store extra information:



Bits 30,31 store the original color of the graph, bits 24..28 form a counter. Procedure pgTimeErase is called by procedure rotate to decrease the counter value line by line for all pixels. A pixel is repainted lighter as the counter decreases. When zero is reached bits 30,31 are examined to restore the original pixel color.

Figure: 7

Motion type detection and cycle time reporting
Motion type is either pendulum or rotation.

```
Type    Trefstatus = (rsNone,rsXright,rsXLeft,rsUright,rsUleft);
Trefpoint = record
          status : Trefstatus;
          angle  : single;
          time   : int64;  //CPU time
                     end;
```

The reference point is a wheel position with recorded CPU time and status:

- `rsXright, rsXleft` : this point was crossed before in right (clockwise) or left direction.
- `rsUright, rsUleft` : this point was a turning point before in clock- or counterclockwise direction.

A 2 bit code represents wheel movement

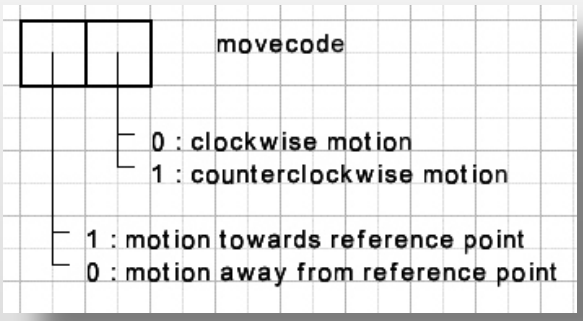


Figure : 8

`function makemovecode : byte;` produces the move code as the name suggests.
`procedure CycleEvent;` is called by procedure simulate.

`CycleEvent` calls `makemovecode` to make a new move code to be compared with the previous code.

Upon detection of a possible rotation or pendulum swing,
`procedure processturn` or `procedure processrotation` is called
to perform the reporting.
A turn sets a new reference point.
These procedures are a little tricky because angles are measured `0..359` in degrees but relative to the reference point angles are counted `-180..179` degrees.

CONCLUSIONS

I hope to have issued a clear general picture of the simulation process without boring details. To really generate chaotic movement which means alternating pendulum swings and rotation takes some experimenting with fill- and leak rates and starting wheel angles.

In many cases the wheel locks in a stable rotation just after a few pendulum swings.
But sometimes this takes several minutes.
Randomness is not build in.

In reality of course the wheel motion is far more complex as:

- buckets will wobble&
- bucket gravity points shift as they fill with water
- buckets movement is on a circle with center somewhat 15cm. below the wheel center.
- supply of water will always be somewhat irregular.

In this simulation I assume all weight and bucket gravity points at the wheel perimeter.

I use two of my own components:

1. A microsecond timer

The time is delivered as double variable where the integer part represents microseconds.
The time is obtained from the CPU clock.
The methods timer.start and timer.stop may be somewhat misleading.
The CPU clock never stops.
Start and stop store the time so later the time difference may be calculated.
A single start may be followed by multiple stops to obtain the elapsed time.

2.Rotation buttons

These are laboratory equipment type buttons that present an integer value.
The buttons are operated by a mouse move while holding down the mousebutton.

Version 3
The empty mass of the wheel is reduced to 15 kg.
The 12 buckets of 28 liters are replaced by 15 smaller cylindrical buckets of 10 liters.
At the right bottom of the form a rotation button selects the amount of friction,
which counts in Newton. A setting of 2 at a speed of 3 (m/sec)
causes a friction of 2*3^2 = 18 Newton.
Some more friction causes more chaotic movement of the wheel.
Chaotic means alternating pendulum and rotation movement.
Without friction the wheel sooner locks into stable rotational movement.

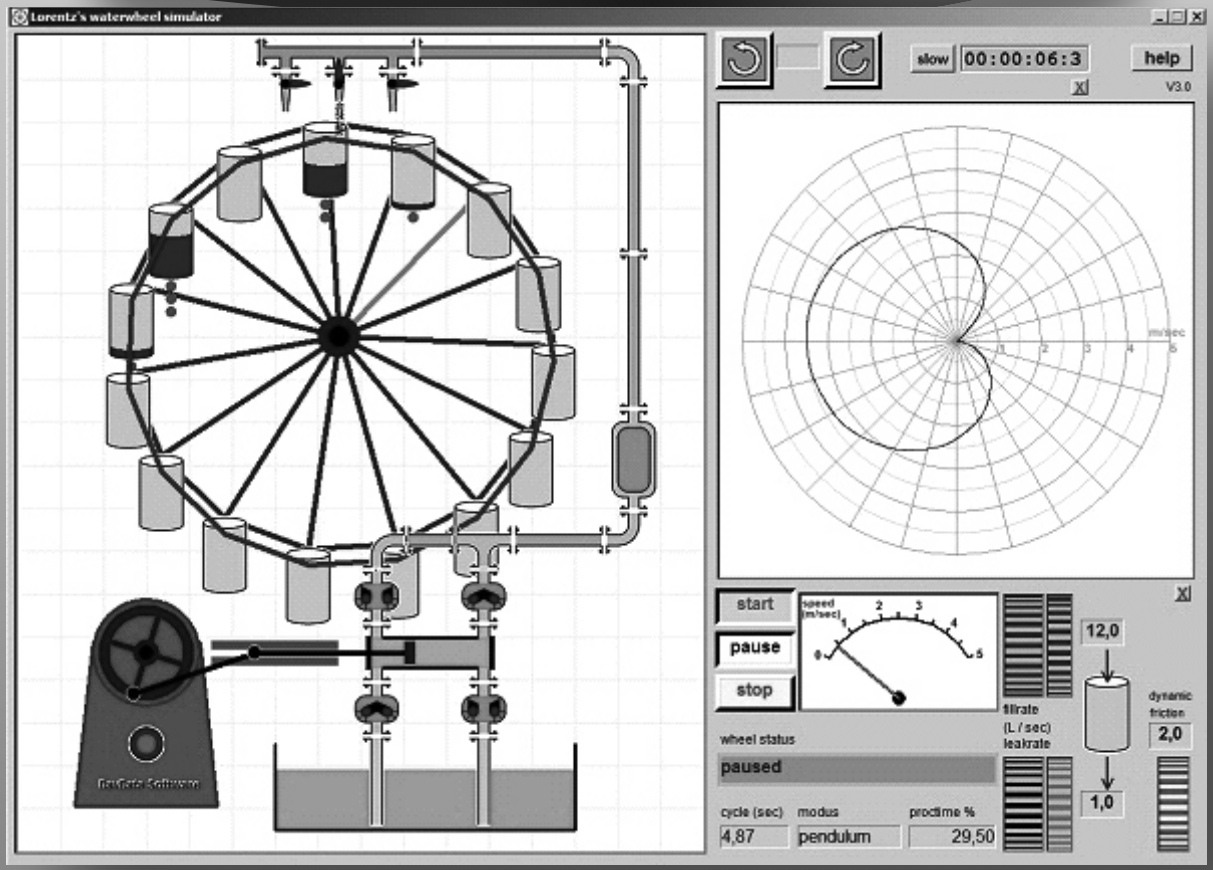


Figure : 9

The volume of the buckets is painted by vertical (blue) lines.
To save time volumeOffset : array[1..38] of byte; is preset at create time.

```
procedure presetVolumeOffsets;
var i : byte;
begin
  for i := 1 to 38 do
    volumeOffset[i] := round(0.25*sqrt(400 - sqr(i-20)));
end;
```

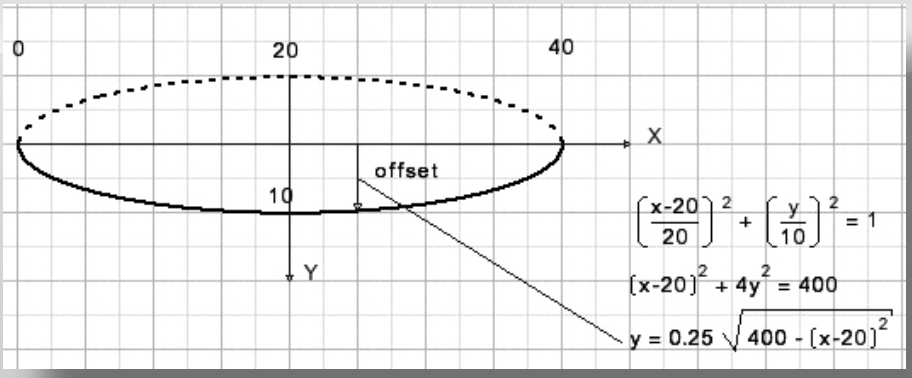


Figure : 10

BY MICHAËL VAN CANNEYT

starter expert

PAS2JS 5

ABSTRACT

PDF is probably the most used document format on the web. You don't need to install any special software to view it: The browser can perfectly display a **PDF.** The technology to view a **PDF** is available as a **Javascript** library and so we can use it in pas2js. In this article we show how.

1 INTRODUCTION

Chances are that if you order some product or service online, you will get an invoice or order confirmation as a **PDF** document. When properly configured, the browser will display this document for you, without the need to save it and open an application such as **Acrobat Reader** or **Foxit** reader.

The browser can do this thanks to a Javascript library (`pdf.js`) developed by the **Mozilla** team: This javascript library takes care of everything, and can even create a complete GUI to navigate the **PDF,** search, select text etc.
The implementation is very complete, and will result in a very qualitative rendering of a **PDF.** Since it is **Javascript** and open source, we can use this library to handle display of a **PDF** ourselves in a **Pas2JS** program.

The `Pdf.js` implementation is available on github:
`https://github.com/mozilla/pdf.js`
The library comes in 2 versions: one for older browsers, one for more modern browsers. In this text, we'll assume a modern browser.
There are 2 parts to the library: the first part is the actual **PDF.js** API, which can parse and draw a **PDF** file. The second part is a viewer, which provides a standard UI for manipulating the **PDF:** browsing through the pages, searching, copying text etc.
In this article, we'll limit ourselves to the first part and use that to build a small PDF viewing Application
Pre-built versions of this library can be found on:

```
https://github.com/mozilla/pdfjs-dist
```

To include it in your **pas2js** program, the following line can be added to your HTML page:

```
<script src="https://mozilla.github.io/pdf.js/build/pdf.js"></script>
```

2 THE JAVASCRIPT

That is sufficient to get started. The **API** of `PDF.js` exists as a typescript declaration module, and this has been translated to pascal using the **dts2pas** tool that comes with **pas2js.** The resulting file has been included in the **pas2js** distribution as a unit called pdfjs. This unit is not to be confused with the **jspdf** unit inthe **jsPDF** package which also exists, but serves to generate a **PDF** file using **Javascript,** instead of displaying it. The **Javascript** library makes heavy use of promises: the heavy lifting is done using web workers (*a kind of threads*), and communication with the web workers is done through asynchronous messages. There is a global object in the library which has some configuration variables that can be set, and which also contains the starting point for displaying **PDF** files. The global object is declared as follows:

```
TPDFJSStatic = class external name 'Object' (TJSObject)
  maxImageSize : Double;
  cMapUrl : string;
  cMapPacked : boolean;
  disableFontFace : boolean;
  imageResourcesPath : string;
  disableWorker : boolean;
  workerSrc : string;
  disableRange : boolean;
  disableStream : boolean;
  disableAutoFetch : boolean;
  pdfBug : boolean;
  postMessageTransfers : boolean;
  disableCreateObjectURL : boolean;
  disableWebGL : boolean;
  disableFullscreen : boolean;
  disableTextLayer : boolean;
  useOnlyCssZoom : boolean;
  verbosity : Double;
  maxCanvasPixels : Double;
  openExternalLinksInNewWindow : boolean;
  isEvalSupported : boolean;
  GlobalWorkerOptions : TGlobalWorkerOptions;
  Procedure PDFSinglePageViewer(params : TPDFViewerParams);
  Procedure PDFViewer(params : TPDFViewerParams);
  Function getDocument(url : string;
    pdfDataRangeTransport : TPDFDataRangeTransport;
    passwordCallback : TgetDocument_passwordCallback;
    progressCallback : TgetDocument_progressCallback):
      TPDFLoadingTask;
  Function getDocument(url : string): TPDFLoadingTask;
  Function getDocument(url : string;
  pdfDataRangeTransport : TPDFDataRangeTransport): TPDFLoadingTask;
  Function getDocument(url : string;
  pdfDataRangeTransport : TPDFDataRangeTransport;
  passwordCallback : TgetDocument_passwordCallback): TPDFLoadingTask;
  Function getDocument(data : jsvalue;
    pdfDataRangeTransport : TPDFDataRangeTransport;
    passwordCallback : TgetDocument_passwordCallback;
    progressCallback : TgetDocument_progressCallback): TPDFLoadingTask;
  Function getDocument(data : jsvalue): TPDFLoadingTask;
  Function getDocument(data : jsvalue;
    pdfDataRangeTransport : TPDFDataRangeTransport): TPDFLoadingTask;
  Function getDocument(data : jsvalue;
    pdfDataRangeTransport : TPDFDataRangeTransport;
    passwordCallback : TgetDocument_passwordCallback): TPDFLoadingTask;
  Function getDocument(source : TPDFSource;
    pdfDataRangeTransport : TPDFDataRangeTransport;
    passwordCallback : TgetDocument_passwordCallback;
    progressCallback : TgetDocument_progressCallback): TPDFLoadingTask;
  Function getDocument(source : TPDFSource): TPDFLoadingTask;
  Function getDocument(source : TPDFSource;
    pdfDataRangeTransport : TPDFDataRangeTransport): TPDFLoadingTask;
  Function getDocument(source : TPDFSource;
    pdfDataRangeTransport : TPDFDataRangeTransport;
    passwordCallback : TgetDocument_passwordCallback): TPDFLoadingTask;
end;
```

2 THE JAVASCRIPT

The meaning of most of these properties are pretty obvious from their names, we will not discuss them here except the ones we need. This global object is not declared in the pdfjs library, but can be inserted in your program as follows:

```
var   pdfjsLib : TPDFJSStatic; external name 'pdfjsLib';
```

As you can see, there are only 3 calls in this global object:

1. `PDFSinglePageViewer` creates a single-page **PDF** viewer **UI**.
2. `PDFViewer` Creates a multi-page **PDF** viewer **UI.**
3. `getDocument` downloads a document (*or obtain one from in-memory data*) and parse it.

   A task object is returned.

Before using these calls, however, the `workerSrc` property must be set: this is the location of the worker script that contains the background processing logic. It can be set as follows:

3 A SMALL DEMO

To demonstrate this API, we'll create a simple pas2js application that allows to enter the URL of a PDF. It will download and show the PDF. To navigate, we'll add the usual previous/next/first/last buttons and similar buttons to set the zoom. We will use again bulma as a CSS framework.
For this, in the Project - New project menu of the **Lazarus** IDE, we select the 'Web Browser Application" type,
and we select the option to use the browser application class. All code will be put in the browser application class. The HTML for the navigation buttons (2 buttons, an edit for a page

```
<div class="field has-addons">
  <p class="control">
    <button id="btnFirst" class="button is-info">
      <span class="icon is-small">
        <i class="las la-angle-double-left"></i>
      </span>
    </button>
  </p>
  <p class="control">
    <button id="btnPrevious" class="button is-info">
      <span class="icon is-small">
        <i class="las la-angle-left"></i>
      </span>
    </button>
  </p>
  <p class="control is-small">
    <input id="edtPageNo" class="input" style="max-width: 3em;">
  </p>
  <p class="control">
    <button id="btnNext" class="button is-info">
      <span class="icon is-small">
    <i class="las la-angle-right"></i>
  </span>S
    </button>
  </p>
  <p class="control">
    <button id="btnLast" class="button is-info">
      <span class="icon is-small">
        <i class="las la-angle-double-right"></i>
      </span>
    </button>
  </p>
</div>
```

3 A SMALL DEMO

For the zoom buttons and edit, the **HTML** looks similar, so we will not repeat it here. As usual, we start the application by binding all elements to an identifier in the application class, and hook up events to the elements:

```
procedure TMyApplication.BindElements;
begin
  btnPrevious:=TJSHTMLButtonElement(GetHTMLElement('btnPrevious'));
  btnPrevious.addEventListener('click',@onPrevPage);
  btnFirst:=TJSHTMLButtonElement(GetHTMLElement('btnFirst'));
  btnFirst.addEventListener('click',@onFirstPage);
  btnNext:=TJSHTMLButtonElement(GetHTMLElement('btnNext'));
  btnNext.addEventListener('click',@onNextPage);
  btnLast:=TJSHTMLButtonElement(GetHTMLElement('btnLast'));
  btnLast.addEventListener('click',@onLastPage);
  btnLoad:=TJSHTMLButtonElement(GetHTMLElement('btnLoad'));
  btnLoad.addEventListener('click',@onLoad);

  btnZoomIn:=TJSHTMLButtonElement(GetHTMLElement('btnZoomIn'));
  btnZoomIn.addEventListener('click',@onZoomIn);
  btnZoomOut:=TJSHTMLButtonElement(GetHTMLElement('btnZoomOut'));
  btnZoomOut.addEventListener('click',@onZoomOut);
  lblZoom:=GetHTMLElement('lblZoom');

  edtPageNo:=TJSHTMLInputElement(GetHTMLElement('edtPageNo'));
  edtPageNo.onkeyup:=@DoPageKeyUp;
  FCanvas:=TJSHTMLCanvasElement(GetHTMLElement('PDFCanvas'));
  FCtx:=TJSCanvasRenderingContext2D(FCanvas.getContext('2d'));
end;
```

The purpose of the buttons and edits are clear: they will be the **UI** elements for our application. The last 2 lines need some explanation: The `PDF.js` **API** needs a canvas element to draw the **PDF**. The application that makes use of the `PDF.js` **API** must provide the 2D rendering context this element. In our **HTML** page we have included such an element below the buttons:

```
<!-- pdf -->
<div class="is-flex is-justify-content-center">
  <canvas id="PDFCanvas" height="737" width="538"></canvas>
</div>
```

and the code in the last 2 lines of the `BindElements` routine saves a reference to the 2D rendering context of the canvas element. The `btnLoad` element is the button to press when you wish to select a **URL** to download. The `onClick` handler is not very exciting:

```
procedure TMyApplication.onLoad(aEvent: TJSEvent);
begin
  StartPDFRender;
end;

procedure TMyApplication.StartPDFRender;
var aSource : TPDFSource; aURL : String;
begin
  aUrl:=window.prompt('Please enter the URL of a PDF to view');
  if IsNull(aURL) then exit;
  aSource:=TPDFSource.New;
  aSource.URL:=aURL;
  ShowPdf(aSource);
end;
```

3 A SMALL DEMO

The `StartPDFRender` routine is also called when the application is started. The result of this routine looks like figure 1 on page 5. The last 3 lines create a `TPDFSource` class, which is the input expected by the `Pdf.js` **API.** It passes the created class to the `ShowPDF` routine. This routine does the actual work, using the `getDocument` call mentioned earlier:

```
procedure TMyApplication.ShowPDF(aSource: TPDFSource);

  function pdfLoaded(aValue : JSValue) : JSValue;
  begin
    pdfDoc:=TPDFDocumentProxy(aValue);
    Writeln('PDF loaded');
    FpageNum:=1;
    QueueRenderPage(FpageNum);
    Result:=True;
  end;

  Procedure pdfLoadError(aReason : String);
  begin
    Writeln('PDF loading failed: '+aReason);
  end;

var loadingTask : TPDFLoadingTask;
begin
  loadingTask:=pdfjsLib.getDocument(aSource);
  loadingTask.promise.&then(@pdfLoaded,@pdfLoadError);
end;
```

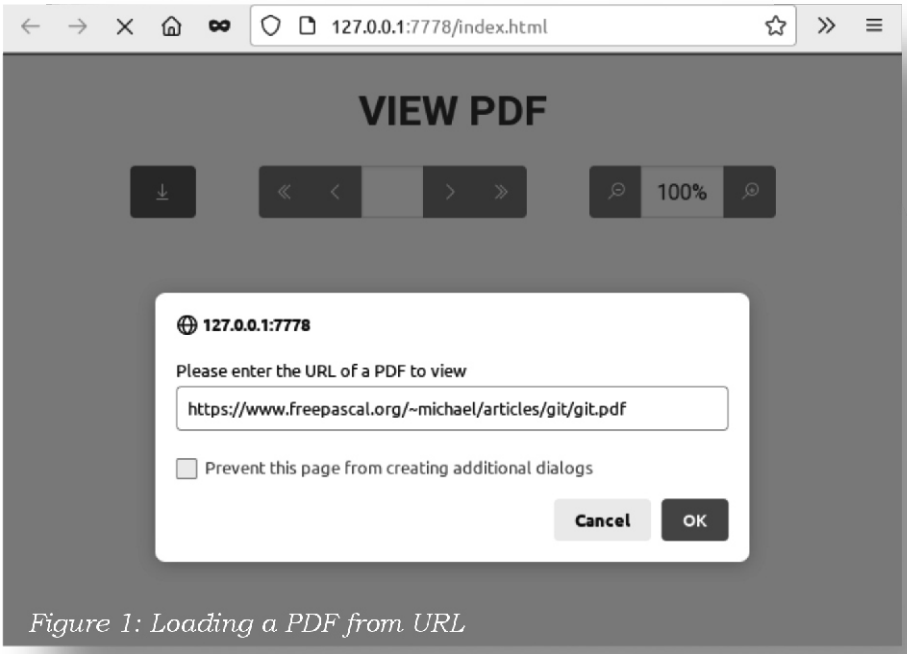


Figure 1: Loading a PDF from URL

The `loadingTask` has a promise member which we can use to wait for the **PDF** to load: The loading of a **PDF** is an asynchronous operation, and we can only display the **PDF** when the **PDF** has been downloaded and parsed. The promise resolves into a `PDFDocumentProxy` class which can be used to execute commands on the downloaded and parsed **PDF.** The `PdfLoaded` routine does exactly that: it saves the **PDF** proxy object, and schedules a render of the first page. Again, rendering is asynchronous, so we must be careful not to interrupt a previous rendering task by starting new one. If a page is currently being rendered, we simply save the number of the page to render. If no page is being rendered, we schedule a rendering task for the requested page:

```
procedure TMyApplication.queueRenderPage(aNum: Integer);
begin
  if FPageRendering then
    FpageNumPending:=aNum
  else
    renderPage(aNum)
end;
```

The `RenderPage` routine is where the actual drawing procedure starts. It starts with a setting a variable, and calling the `pdfDoc.GetPage` routine: this will fetch all the data necessary to draw the page: it returns a `Promise`.

```
procedure TMyApplication.renderPage(aNum: Integer);
begin
  FPageRendering:=True;
  pdfDoc.getPage(aNum).& then (@HavePage);
  edtPageNo.Value:=IntToStr(anum);
end;
```

Note that the page number is set in the edit control. When the page data is fetched, the `promise` is resolved and the local function `HavePage` is called with a `TPDFPageProxy` value. The `HavePage` function will do the actual painting:

```
function havePage (aValue : JSValue) : JSValue;
var page : TPDFPageProxy absolute aValue; viewport : TPDFPageViewport;
    renderContext: TPDFRenderParams; renderTask : TPDFRenderTask; viewportParams : TViewportParameters;
begin
  viewportParams:=TViewportParameters.new;
  viewportParams.scale:=FScale;
  viewport:=page.getViewport(viewportParams);
  Fcanvas.height := viewport.height;
  Fcanvas.width := viewport.width;
  renderContext:=TPDFRenderParams.New;
  renderContext.canvasContext:=Fctx;
  renderContext.viewport:=viewport;
  renderTask:=page.render(renderContext);
  renderTask.promise.&then(@renderOK);
  Result:=True;
end;
```

The drawing is done by preparing some parameters for a **RenderTask.** As the name implies, this task renders a page. This task needs several parameters, one of which is the canvas rendering context which we saved at the beginning of the program.
The canvas is first prepared: the `width` and `height` are set to match the needed `width` and `height` of the **PDF** page using the current zoom level: these dimensions can be obtained from the `getViewPort` call of the `TPDFPageProxy` object.
All this is then passed to the `Render` call of the Page. This call returns a `TPDFRenderTask` object which is again a promise: when this promise resolves, the page will have actually been drawn on the canvas, and `RenderOK` is called.
Because drawing is **asynchronous,** we need to check whether the user navigated to another page while the rendering took place, and call `RenderPage` again with the new page number - the page to render was saved in `FPageNumPending` in the `QueueRenderPage` presented earlier.

```
Function renderOK(aValue : JSValue) : JSValue;
Var N : Integer;
begin
  FPageRendering:=false;
  if (FPageNumPending <> -1) then
    begin
      N:=FPageNumPending;
      FpageNumPending:=-1;
      renderPage(N);
    end;
  Result:=True;
end;
```

The result of all this work can be seen in figure 2 on page 8.
With these routines, the heavy lifting is actually done. Navigation is now easy. We just determine the page that the user wishes to see, and queue the rendering of the desired page. Here are the `'onClick'` events of the 'previous' and 'first' buttons.

```
procedure TMyApplication.onPrevPage(aEvent : TJSEvent);
begin
  if (FpageNum <= 1) then exit;
  Dec(FpageNum);
  queueRenderPage(FpageNum);
end;

procedure TMyApplication.onFirstPage(aEvent : TJSEvent);
begin
  if not assigned(pdfDoc) then exit;
  FPageNum:=1;
  queueRenderPage(FpageNum);
end;
```

The 'Next' and 'Last' buttons have of course similar procedures. The user can enter a page number in the page edit box, and hit the Enter key to jump to the desired page. This is handled in the key up event of the page number edit:

```
function TMyApplication.DoPageKeyUp(aEvent: TJSKeyBoardEvent): boolean;
Var aPage : Integer;
begin
  if not assigned(pdfDoc) then exit;
  Result:=False;
  if (aEvent.Key<>TJSKeyNames.Enter) then exit;
  aPage:=StrToIntDef(edtPageNo.Value,0);
  if (aPage>0) and (aPage<=PdfDoc.numPages) then
  queueRenderPage(aPage);
end;
```

We've seen in the **HavePage** routine that the rendering of the current page is parametrized with a zoom level. This means that to change the zoom, it suffices to change the zoom level variable **(FScale)**, and request a re-rendering of the current page:

```
procedure TMyApplication.onZoomIn(aEvent : TJSEvent);
begin
  if not assigned(pdfDoc) then exit;
  FScale:=FScale+ScaleStep;
  DisplayZoom;
  queueRenderPage(FpageNum);
end;

procedure TMyApplication.DisplayZoom;
begin
  lblZoom.innerText:=IntToStr(Round(FScale*100))+'%';
end;
```

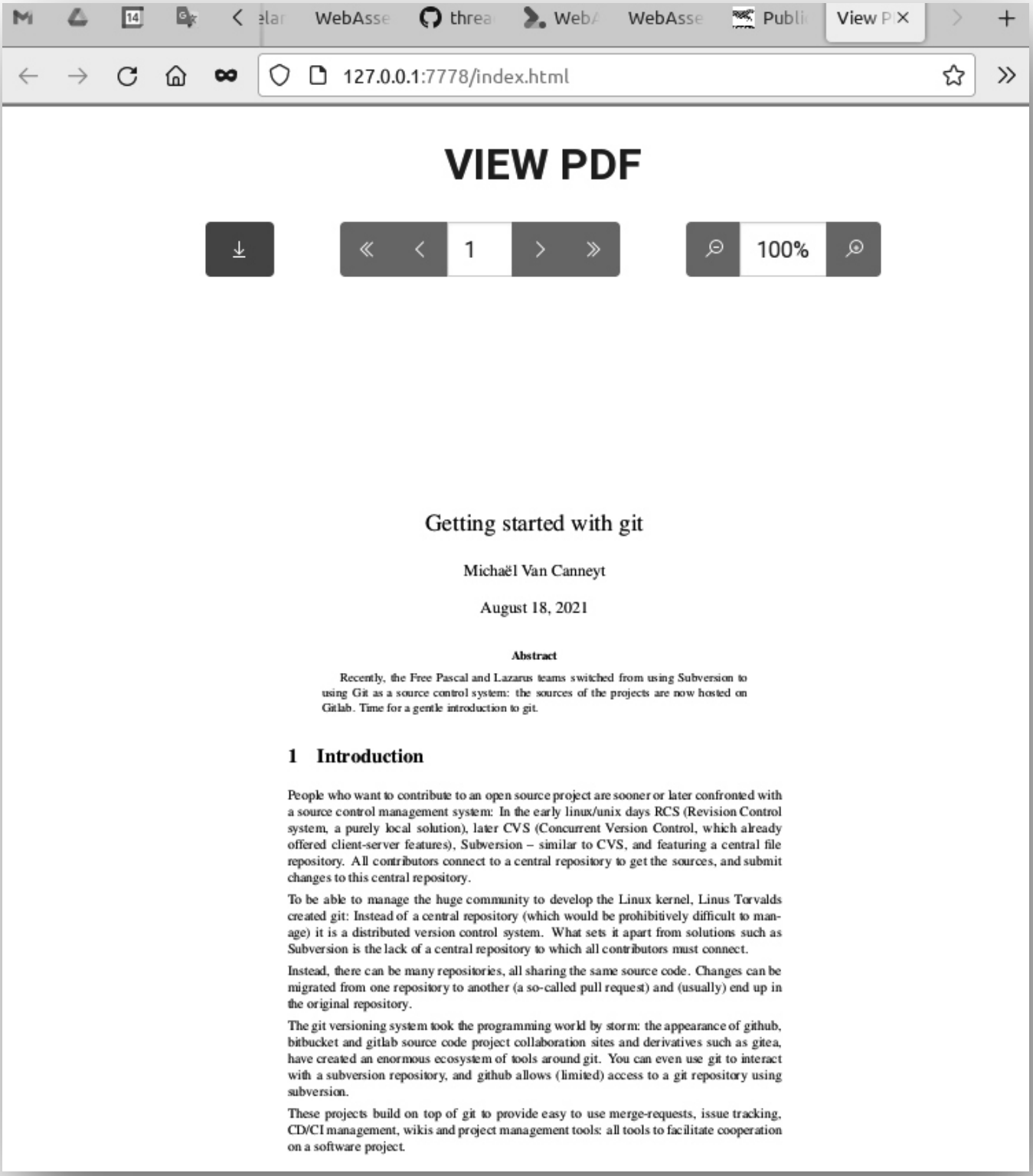


Figure 2: Showing the loaded PDF

The `OnZoomOut` routine is of course similar to the routine to zoom in.

4 SELECTING A FILE FROM DISK

The previous paragraphs show that it is really not difficult to load a **PDF** from a **URL.** In the demo we showed how to load a **PDF** from a **URL.** In most programs, the user will of course not need to enter a **URL** manually: most of the time, the application will know the **URL** where the PDF can be found.

But what if you want to allow the user to select a **PDF** from disk for upload to a server, and wish to present the user with a preview of the **PDF** before actually uploading the file?

It would be silly to first upload the file to a temporary area and then download it again to preview it...

Fortunately, showing a file selected from disc can easily be implemented. To do so, we add an 'upload' button to the demo application.

In **HTML,** an input tag with type "file" can be used to select a file. The **Bulma CSS** framework has some special **CSS** classes that can be used to mask this file input edit so it is invisible, and just shows the button that opens the file picker dialog. The necessary **HTML** looks like this:

```
<div class="field has-addons mr-6">
   <div class="file">
     <label class="file-label">
       <input id="edtPDFFile" class="file-input" type="file" name="pdfFile">
       <span class="file-cta is-link">
            <span class="file-icon is-link">
            <i class="las la-upload"></i>
         </span>
         <span class="file-label is-link ">
            Load PDF...
         </span>
       </span>
     </label>
   </div>
</div>
```

We bind the input element and handle the `OnChange` event:

```
edtPDFFile:=TJSHTMLInputElement(GetHTMLElement('edtPDFFile'));
edtPDFFile.onchange:=@DoLoadFile;
```

The `DoLoadFile` is a short routine.

```
function TMyApplication.DoLoadFile(Event: TEventListenerEvent): boolean;
Var aFileName : string;

  Function LoadFromFile(aData : JSValue) : JSValue;
  var aSource : TPDFSource;
  begin
     Result:=False;
     pdfDoc:=Nil;
     aSource:=TPDFSource.New;
     aSource.data:=aData;
     ShowPDF(aSource);
     DisplayFileLocation(aFileName);
  end;

begin
  Result:=False;
  if (edtPDFFile.files.length=1) then
     begin
       aFileName:=ExtractFileName(edtPDFFile.files[0].Name);
       edtPDFFile.files[0].arraybuffer._then(@loadFromFile);
     end;
end;
```

4 SELECTING A FILE FROM DISK

The start of the routine uses the input file **API** to get the data into an array buffer using the `arraybuffer` function: calling this will load the file into memory, and of course works with a promise. The promise resolves in an array buffer with the data of the file. The real work then happens in the local `LoadFromFile` routine: it creates a `TPDFSource` object, but instead of setting the `Url` element of that object (as we did before), it now sets the data element. The data element is assumed to contain the data of the **PDF** file to display.
We add a little routine to display the file location in the browser title bar and the title at the top of the page.

Figure 3: Showing an uploaded PDF

The result of this little piece of code can be seen in figure 3 on page 10.

5 DISTRIBUTING PDF FILES ON DISK

What if you wanted to make and distribute a **PWA (Progressive Web Application)** with multiple **PDF** files and allow the user to select one from a list, which is then viewed in the **PWA** without needing to download it?
You could include the data from all the **PDF** files as array variables in the **HTML** page.
But if you have a lot of files, this would cause the page load to be very slow, and the memory consumption could become very high when the files are large.
It would be better to load only the needed **PDF** into memory. But how to do so without asking the user to select a file ? This can be done using a small trick:
It is possible at runtime to add a script tag to the **HTML** page, simply inject

```
<script src="mypdf.js"></script>
```

into the **DOM** of the **HTML.**
This can be used to inject a variable definition in the browser namespace.

(let's call it **myPDF**), if the **mypdf.js** file contains simply a variable definition, like this:

```
var myPDF = atob("JVBERi0xLjcNCiXi48/TDQo5ODIgMCBvYmoNC...");
```

The string literal (truncated in the above sample) is the **PDF** file contents, encoded as **base64**, and **atob** is the browser function that allows to decode the **base64** string. By injecting different script tags, the contents of the variable can be changed: the browser will happily redefine the **myPDF** variable every time a new script tag is included. The contents of the **PDF** can then be shown using the following code, similar to what was shown earlier. The first line defines the global **myPDF** variable in the **Pascal** program:

```
var
  myPDF : JSValue; external name 'myPDF';

Procedure LoadFromFile;
var aSource : TPDFSource;
begin
  pdfDoc:=Nil;
  aSource:=TPDFSource.New;
  aSource.data:=myPDF;
  ShowPDF(aSource);
end;
```

As you can see, the code is no different from what was presented earlier, the source of the **PDF** data is simply a global variable. The following code is part of a class called **TFileToJSBase64Var**, and can be used to create the script file:

```
procedure TFileToJSBase64Var.Convert(aInputStream, aOutputStream: TStream);
Var BES : TBase64EncodingStream; VN,SPrefix,SSuffix : UTF8String;
begin // Set the variable name from the VarName property.
  VN:=VarName;
  if VN='' then VN:='file';
  SPrefix:='var '+VN+' = ';
  SSuffix:="";
  // if UseAToB is true, we include the call to atob in the variable definition:
  if UseAToB then
    begin
      SPrefix:=SPrefix+'atob(';
      SSuffix:=SSuffix+')';
    end;
  SPrefix:=SPrefix+'"';
  SSuffix:=SSuffix+';'#10;
  aOutputStream.WriteBuffer(SPrefix[1],Length(SPrefix));
  BES:=TBase64EncodingStream.Create(aOutputStream);

  try
    BES.SourceOwner:=False;
    Bes.CopyFrom(aInputStream,0);
    BES.Flush;
    aOutputStream.WriteBuffer(SSuffix[1],Length(SSuffix));
  finally
    BES.Free;
  end;
end;
```

The full class is available with the source code of this article in the **Filetojsvar** unit. By creating a **Javascript** file for each **PDF** you wish to include, you can inject the appropriate file into the **DOM** after the user selected a **PDF**, and display it with the above code. This can for example be used to distribute a USB stick with **PDFs** and include a web page that can be used to view the **PDF** files. No actual executables will be needed, so this is a completely cross-platform solution to distribute **PDFs** with a **PDF** viewer included.

6 ADDING A SEARCH OPTION

The **PDF.js** library allows you to retrieve the text fragments in the **PDF.** This functionality can be used to add an option to search the text in the **PDF.**
For demonstration purposes, we'll add a double mechanism to the **PDF** viewer. The user can enter a search term in an edit box. When matches are found, he or she can browse through the pages that contain a match. For this, we add the following **HTML,** which uses a popup mechanism to show the matching pages browser;

```
<div class="field has-addons ml-6">
  <p class="control is-small">
    <div class="popover is-popover-bottom is-not-popover-hover"
       id="searchPopover">
     <input id="edtSearch" class="input" style="max-width: 15em;">
     <div class="popover-content">
       Page <span id="lblCurrentSearchResult">1</span>/
           <span id="lblSearchResultCount">10</span>
       <span   class="icon is-small ml-2" id="btnNextSearchResult">
          <i   class="las la-angle-down"></i>
       </span>
       <span class="icon is-small" id="btnPreviousSearchResult">
          <i class="las la-angle-up"></i>
       </span>
       <button id="btnCloseSearchResult"
          class="delete ml-4 is-small"></button>
     </div>
    </div> <!-- .popover -->
  </p>
  <p class="control">
    <button id="btnSearch" class="button is-info">
     <span class="icon is-small">
       <i class="las la-search"></i>
     </span>
    </button>
  </p>
</div>
```

The elements with **IDs** **searchPopover, edtSearch, lblCurrentSearchResult, lblSearchResultCount btnPreviousSearchResult btnNextSearchResult, btnCloseSearchResult and btnSearch** are added as fields to the application class and bound to the **HTML** tags in the usual manner. Clicking the **btnSearch** button starts the search, as well as hitting the Enter key in the search edit;

```
function TMyApplication.DoSearchKeyUp(aEvent: TJSKeyBoardEvent): boolean;
begin
  Result:=False;
  if (aEvent.Key<>TJSKeyNames.Enter) then exit;
  DoSearch;
end;

procedure TMyApplication.onSearch(aEvent: TJSEvent);
begin
  DoSearch;
end;
```

The **DoSearch** starts the search mechanism, which is implemented in a separate class **TPDFSearch**. An instance of this class is created if it did not yet exist.
The search class has a method **searchDocument**, which results in a promise.

```
procedure TMyApplication.DoSearch;
var aterm : string;
begin
  if not assigned(pdfDoc) then exit;
  aTerm:=edtSearch.Value;
  if Length(aTerm)<=1 then exit;
  if FSearch=Nil then FSearch:=TPDFSearch.Create(Self);
  FSearch.PDF:=pdfDoc;
  FSearch.searchDocument(aTerm)._then(@OnHaveResults);
end;
```

The promise resolves in a `TPDFSearchResult`, which is defined as follows:

```
TPDFSearchMatch = Record
  Page : Integer;
  Text,
  Context1,
  Context2 : String;
end;

TPDFSearchMatchArray = Array of TPDFSearchMatch;

TPDFSearchResult = Record
  Matches : TPDFSearchMatchArray;
end;
```

For every match in the text, a `TPDFSearchMatch` exists, which has the text, page and 2 pieces of context (surrounding text). The local method `OnHaveResults` will handle the display of results using the `TPDFSearchMatch` records. It starts by creating an array of unique page numbers containing a match and saving that in the form variable `FSearchResultPages`:

```
Function OnHaveResults(aValue:JSValue):JSValue;
var Results:TPDFSearchResult absolute aValue;
    Pages:Array of Integer; I:Integer;
begin
  Result:=Undefined;
  If Length(Results.Matches)>0 then
  begin
    Pages:=[Results.Matches[0].Page];
    // Create page number list
    For I:=1 to Length(Results.Matches)-1 do
      if TJSArray(Pages).indexOf(Results.Matches[i].Page)=-1 then
        TJSArray(Pages).Push(Results.Matches[i].Page);
        FSearchResultPages:=Pages;
        ShowSearchResultMatchCount;
        ShowSearchResultPage(0);
        FSearchResultsPopover.classList.add('is-popover-active');
        ClearResultPane;

    For I:=1 to Length(Results.Matches)-1 do
      ShowResultMatch(I,Results.Matches[i]);
      ShowResultPane;
  end;
end;
```

After constructing the page array, the count is shown in the popover, and the first page with a result is displayed:

```
Procedure TMyApplication.ShowSearchResultMatchCount;
begin
  lblSearchResultCount.InnerText:=IntToStr(Length(FSearchResultPages));
end;

procedure TMyApplication.ShowSearchResultPage(aIdx: Integer);
begin
  FCurrentSearchResultPage:=aIdx;
  lblCurrentSearchResult.InnerText:=IntToStr(aIdx+1);
  queueRenderPage(FSearchResultPages[aIdx]);
end;
```

6 ADDING A SEARCH OPTION

The arrow buttons `btnPreviousSearchResult` `btnNextSearchResult` can be used to navigate the page numbers in the `FSearchResultPages` array:

```
procedure TMyApplication.onNextSearchResult(aEvent: TJSEvent);
begin
  if FCurrentSearchResultPage>=(Length(FSearchResultPages)-1) then exit;
  ShowSearchResultPage(FCurrentSearchResultPage+1);
end;

procedure TMyApplication.onPreviousSearchResult(aEvent: TJSEvent);
begin
  if FCurrentSearchResultPage<1 then exit;
  ShowSearchResultPage(FCurrentSearchResultPage-1);
end;
```

If the user found the page he was looking for, the `btnCloseSearchResult` button can be used to close the popover:

```
procedure TMyApplication.onCloseSearch(aEvent: TJSEvent);
begin
  FSearchResultsPopover.classList.Remove('is-popover-active');
end;
```

Scrolling through the list of pages with a match for the search term is nice, but not altogether satisfying. It is of course easier if the user can see a list of the matches together with some context for each match. For this, we create a side panel (*initially hidden*) that will show the full result list with more details. The user can scroll through the list, click on a match and jump to the particular page. To make things more interesting, we'll allow the user to filter the search result list as well. The **HTML** for our side pane looks like this:

```
<div id="pnlSidebar" class="sidebar sidebar_closed">
  <!-- close/open button -->
  <button id="btnClosePane"
      class="close-button has-background-link has-text-white">
    <i class="las la-arrow-right la-lg" ></i>
  </button>
  <!-- side bar content -->
  <div id="pnlResults" class="panel is-link">

    <!-- 1. title -->
    <p class="panel-heading">
      Search results
    </p>

    <!-- 2. search bar -->
    <div id="pnlSearch" class="panel-block panel-search">
      <p class="control has-icons-left">
        <input id="edtSearchInResults" class="input is-link"
               type="text" placeholder="Search in results">
        <span class="icon is-left">
          <i class="las la-search" aria-hidden="true"></i>
        </span>
      </p>
    </div>
    <!-- here follow items -->

  </div> <!-- panel -->
</div> <!-- bar -->
```

The button `btnClosePane` can be used to close the search panel, and the `edtSearchInResults` edit can be used to filter the shown results even further.
Each item will be displayed using a **HTML** snipped like this

6 ADDING A SEARCH OPTION

```
<a class="panel-block result-item">
  <span class="panel-icon"
    <i class="las la-book" aria-hidden="true"></i>
  </span>
  <div class="subtitles">
    <p class="has-text-weight-semibold">Page X:</p>
    <p>
      <span>some leading text </span>
      <span class="has-text-weight-bold">matched text in bold</span>
      <span>some trailing text </span>
    </p>
  </div>
</a>
```

the following 3 lines shown earlier take care of displaying the search results in the search pane:

```
ClearResultPane;
For I:=1 to Length(Results.Matches)-1 do
   ShowResultMatch(I,Results.Matches[i]);
ShowResultPane;
```

The **ClearResultPane** and **ShowResultPane** are quite simp

```
procedure TMyApplication.ShowResultPane;
begin
  pnlSidebar.classList.Remove('sidebar_closed');
  edtSearchInResults.Value:='';
end;

procedure TMyApplication.ClearResultPane;
Var Rem,Child:TJSHTMLElement;
begin
  Child:=TJSHTMLElement(pnlResults.firstElementChild);
  While Assigned(Child) do
  begin
    Rem:=TJSHTMLElement(Child);
    Child:=TJSHTMLElement(Rem.nextElementSibling);
    if Rem.classlist.contains('result-item') then
      pnlResults.removeChild(Rem);
  end;
end;
```

The loop simply removes all elements that have result-item in their class list.
The **ShowResultMatch** does the actual work of constructing the **HTML** with the results.
It does this by creating an anchor element, and filling the anchor element with a **HTML** snippet.
The **ResultContent** constant contains the **HTML** snippet shown above (*minus the anchor element itself*), with some formatting placefolders for the call to Format:

```
Function TMyApplication.ShowResultMatch(Idx : Integer;
  aResult : TPDFSearchMatch) : TJSHTMLElement;
begin
  Result:=TJSHTMLElement(Document.CreateElement('a'));
  Result.Dataset['page']:=IntToStr(aResult.Page);
  Result.Dataset['idx']:=IntToStr(Idx);
  Result.ClassName:='panel-block result-item';
  Result.InnerHTML:=Format(ResultContent,[aResult.Page,
       aResult.Context1,aResult.Text,aResult.Context2]);
  Result.AddEventListener('click',@DoShowMatch);
  pnlResults.appendChild(Result);
end;
```

As you can see in the above code, the index of the match, and the page on which the match occurs are saved in the **Dataset** property of the result element; The resulting anchorelement also gets an **onclick** handler which will use the 'page' variable to determine the page to jump to:

6 ADDING A SEARCH OPTION

```
Procedure TMyApplication.DoShowMatch(aEvent : TJSEvent);
var aPage : Integer; aAnchor : TJSHTMLElement;
begin
  pnlResults.querySelectorAll('result-item').foreach(@MakeInactive);
  aEvent.currentTargetElement.classList.add('is-active');
  aAnchor:=TJSHTMLElement(aEvent.currentTargetElement);
  aPage:=StrToIntDef(String(aAnchor.Dataset['page']),-1);
  if aPage<>-1 then queueRenderPage(aPage);
end;
```

Note that the clicked result is shown visually as active, while the other results are made visually inactive first:

```
procedure MakeInactive(currentValue : TJSNode; currentIndex: NativeInt; list : TJSNodeList);
begin
  TJSHTMLElement(CurrentValue).classList.remove('is-active');
end;;
```

The result of all this can be seen in figure 4 on page 16.
As mentioned earlier, to close the search result panel, the user can click the button btnClosePane:

```
procedure TMyApplication.onClosePane(aEvent : TJSEvent);
begin
  HideResultPane;
end;

procedure TMyApplication.HideResultPane;
begin
  pnlSidebar.classList.Add('sidebar_closed');
end;
```

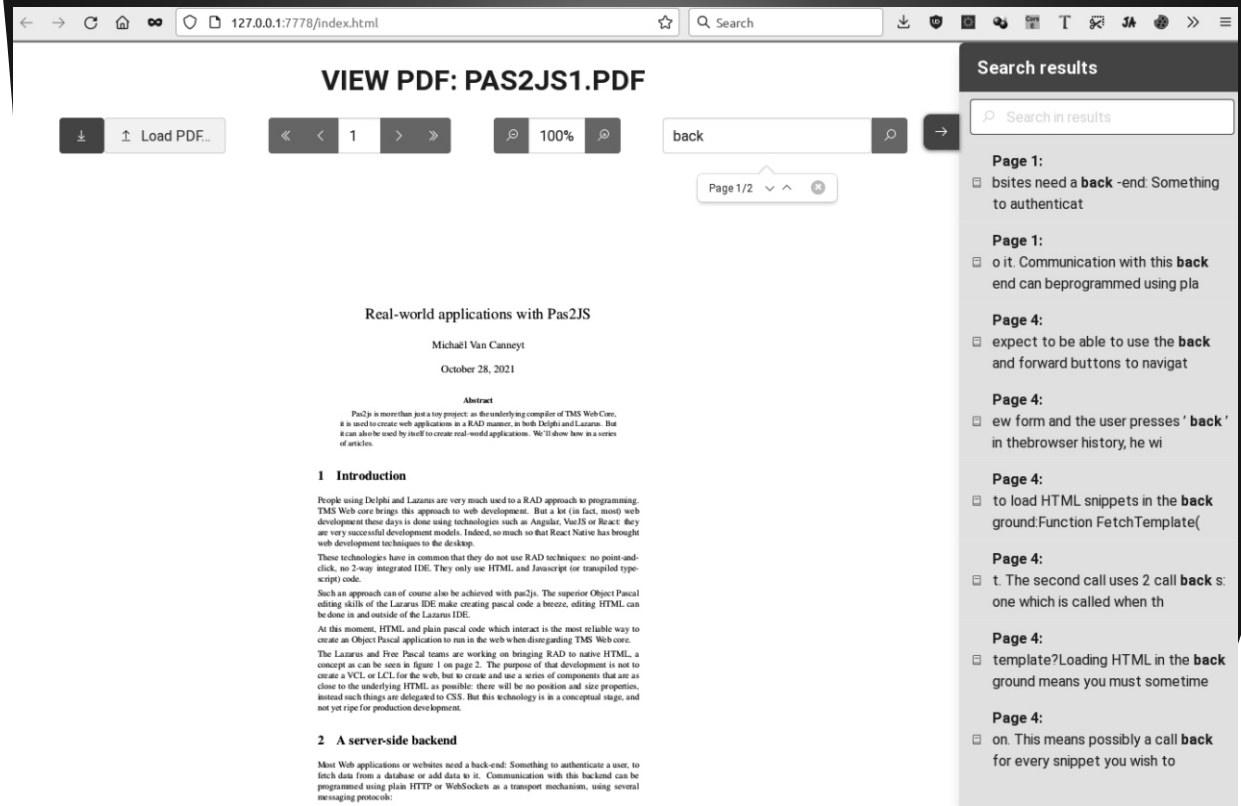


Figure 4: Search in action

The search result list can be further filtered by typing in the filter box at the top of the result pane. The results are filtered by simply hiding the results that do not contain the filter term. This can be done by adding or removing the `is-hidden` **Bulma CSS** class from the anchor element:

```
Function TMyApplication.DoSearchResultsKeyUp(aEvent: TJSKeyBoardEvent): boolean;
Var I,J,Len : Integer; List1,List2 : TJSNodeList; Pnl,Span : TJSHTMLElement; Vis : Boolean; aTerm : String;
begin
  Result:=True;
  aTerm:=UpperCase(edtSearchInResults.value);
  List1:=pnlResults.querySelectorAll('.result-item');
  for I:=0 to List1.Length-1 do
  begin
    Pnl:=TJSHTMLElement(List1.item(I));
    List2:=pnl.querySelectorAll('div p span');
    Vis:=(aTerm='');
    Len:=List2.Length;
    J:=0;
    While (Not Vis) and (J<Len) do
    begin
      Span:=TJSHTMLElement(List2.item(J));
      if (ATerm='') or (Pos(aTerm,UpperCase(Span.InnerText))>0) then Vis:=True;
      Inc(J);
    end;
    if vis then pnl.ClassList.Remove('is-hidden')
    else pnl.ClassList.add('is-hidden');
  end;
end;
```

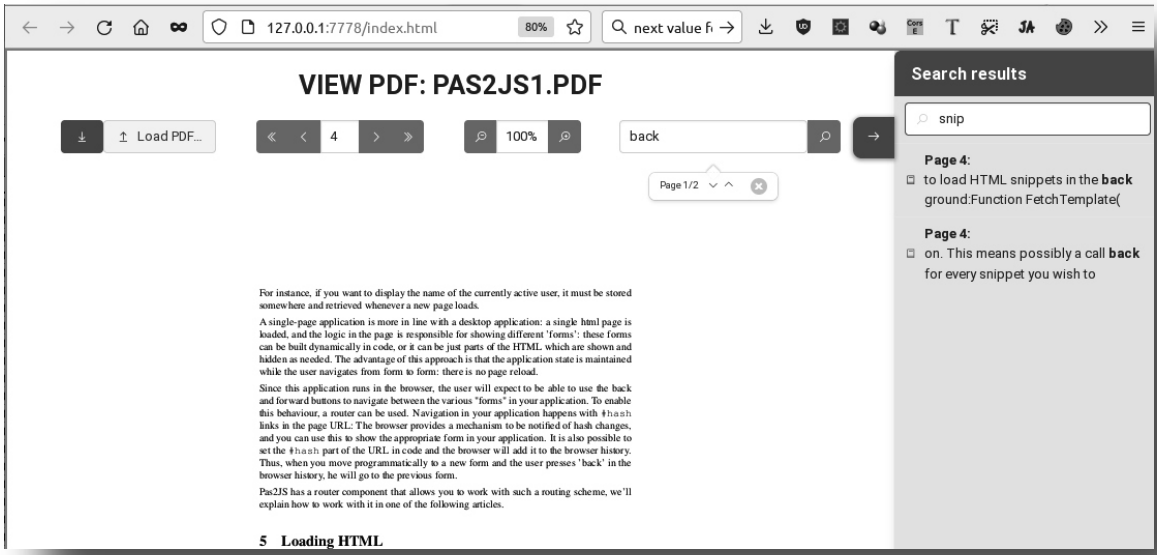


Figure 5: Filtered search results

To determine whether an element matches the filter term, all tags containing text (*'div' 'p' and 'span'*) are examined: if the innertext of one of them contains the filter text, the anchor is set to visible - or `invisible otherwise`.

Note that the double search/filter mechanism is probably not something one would actually implement like this, but the above is meant to show how one could go about it in a real application. The above code is just the display of the results. The actual searching happens in the `TPDFSearch` class:

```
TPDFSearchPageResultsEvent = Reference to Procedure(aPage : Integer; Matches : TPDFSearchMatchArray);
TPDFSearch = Class(TComponent)
Public
  function searchPage(aPageNo : Integer; aSearchTerm : String) : TPDFPromise;
  function searchDocument(aSearchTerm : String) : TJSPromise;
  Property PDF : TPDFDocumentProxy Read FPDF Write FPDF;
  Property ContextChars : Integer;
  Property OnPageResult : TPDFSearchPageResultsEvent;
end;
```

The **PDF** property is of course the document to search. The `ContextChars` is the number of characters to use in the search context. The `OnPageResult` event will be called when all results for a single page are in: the `PDF.js` implementation delivers the text per page, so the search mechanism searches one page at a time, and using this event, you can notify the user of the progress of the search algorithm.
One reason for putting this in a separate class (*except reusability*) is that the `PDF.js` implementation uses promises when returning the text of a page in the PDF, and some state must be kept.

The `SearchDocument` call is the call that starts the actual search. It actually starts a search on every page using the `SarchPage` method. This method returns a promise. All promises are collected in an array, and the `TJSPromise.All` class method is then to wait till all results of all promises are in: this `promise` is the result of the `SearchDocument` call.

The `TJSPromise.All` class method in itself returns a promise that resolves to an array with all the results of all individual promises, in the same order as the original array of promises. When the results are in, the `GotAllResults` method concatenates all the results of all promises, and uses this to resolve the `All promise`.

```
Function TPDFSearch.searchDocument(aSearchTerm: String): TJSPromise;

    Function GotAllResults(aValue : JSValue) : JSValue;
      Var  AllRes : TPDFSearchResultArray absolute aValue;
           aRes : TPDFSearchResult; DocumentResults : TPDFSearchResult;
    begin
      DocumentResults:=Default(TPDFSearchResult);
      for aRes in AllRes do
      DocumentResults.Matches:=Concat(DocumentResults.Matches,aRes.Matches);
      Result:=DocumentResults;
    end;

var  results : array of TPDFPromise;
     I : Integer;
begin
   Results:=[];
   For I:=1 to PDF.numPages do
   Results:=Concat(Results,[SearchPage(I, aSearchterm)]);
   Result:=TJSPromise.all(TJSPromiseArray(Results))._then(@GotAllResults);
end;
```

6 ADDING A SEARCH OPTION

The **SearchPage** method is where the actual search for the search term in a page happens. Because **PDF.js** returns the page (using **getPage**) in a promise, and you must get the the page text with **getTextContent** which again returns a **promise**, 2 callbacks are needed:

```
Function TPDFSearch.searchPage(aPageNo: Integer;
  aSearchTerm: String): TPDFPromise;

    Function GotPage(aValue: JSValue) : JSValue;
    Var aPage : TPDFPageProxy absolute aValue;
    begin
      Result:=aPage.getTextContent();
    end;

begin
  Result:= PDF.getPage(aPageNo).&then(@GotPage).&then(@GotContent);
end;
```

The **GotContent** method does the actual work. It gets as incoming parameter the text content of the page, which is the result of the **getTextContent** call, a record of type **TTextContent**. This record contains an element **items**, which is an array of **TTextContentItem** records. Each **TTextContentItem** has a member **str** which is a chunk of text found in the **PDF.**

The routine starts by concatenating all text chunks using **TJSArray.map** and **join**, and uses a regular expression to search for the search term:

```
Function GotContent(aValue: JSValue) : JSValue;

  Function DoMap(Itm : JSValue; index: NativeInt; anArray : TJSArray) : JSValue;
  var aItem : TTextContentItem absolute itm;
  begin
    Result:=aItem.Str;
  end;

Var aContent : TTextContent absolute aValue; aTerm,aReg, aText : String; aRegEx : TJSRegexp;
    aRegexMatches : TStringDynArray; Res : TPDFSearchResult; aMatch : TPDFSearchMatch;
begin
  Res:=Default(TPDFSearchResult);
  aText:=TJSArray(TJSArray(aContent.items).map(@DoMap)).join('');
  aTerm:=aSearchTerm; // Todo: escape specials
  if FContextChars=0 then FContextChars:=30;
  aReg:=Format('(.{0,%d})%s(.{0,%d})',[FContextChars,aTerm,FContextChars]);
  aRegEx:=TJSRegExp.New(aReg,'gi');
  aRegexMatches:= aRegEx.exec(aText);

  While Length(aRegexMatches)>0 do
  begin
    aMatch:=Default(TPDFSearchMatch);
    aMatch.Page:=aPageNo;
    aMatch.Text:=aSearchTerm;
    if isString(aRegexMatches[1]) then aMatch.Context1:=aRegexMatches[1];
    if isString(aRegexMatches[2]) then aMatch.Context2:=aRegexMatches[2];
    Res.Matches:=Concat(Res.Matches,[aMatch]);
    aRegexMatches:=aRegEx.exec(aText);
  end;
    Result:=Res;
    If Assigned(FOnPageResult) then FOnPageResult(aPageNo,Res.Matches);
  end;
```

In this code, the regular expression is constructed with the following line:

If the search term is 'back' and the number of context characters is 20, this will result in a regular expression:

which is in essence saying search "all occurrences of 'back' with at most 20 characters before and after" The result of executing this regular expression is then an array of 3 elements:

1. Element 0: the full search text
2. Element 1: the content of the first bracket expression.
3. Element 2: the content of the second bracket expression.

A loop is used to execute the regular expression till no more results are found. In each step of the loop, the regular expression result is converted to a `TPDFSearchMatch` record, which is appended to the total result. If the `OnPageResult` event is assigned, it is called with the results for the page.

7 CONCLUSION

The `PDF.js` library is a powerful library which allows to show a PDF in the browser without the need for an external program. This is the same basis that has been useed for the PDF LIbrary - which has some extra options. In this article we show how to use this to display a **PDF** and search in it. With this, we have not yet exhausted the possibilities of the `PDF.js`: the mechanism for searching can also be used to allow highlighting search results and allowing the user to select, copy and paste the **PDF** text. These possibilities will be explored in a future contribution.

https://www.blaisepascalmagazine.eu/product-category/special-offer/

Lazarus Handbook Pocket + Subscription + PDF

- English
- Printed black & white
- 2 Volumes
- Sewn +2 ribbons
- PDF included
- 934 Pages
- Weight: 2kg
- Shipment included
- Extra protected
- Including 40 Example projects and extra programs

SPECIAL OFFER € 75

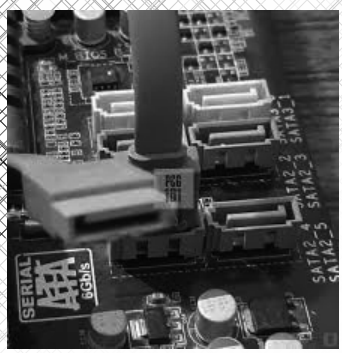
printed on FSC paper
https://fsc.org/en/forest-management-certification

HACKING / SATA CABLE AS ANTENNA?

BY DETLEF OVERBEEK

The pdf belonging to this article can be downloded here:
https://downloads.blaisepascal.eu/2207.07413.pdf
Reference arxiv.org/abs/2207.07413
New Scientist/Magazine issue3398, published 6 August 2022

Computer cables (especially **SATA**) appear to be able to be transformed into an antenna, making it possible to steal data. It is often a precautionary measure to leave an internet connection behind in any case, in order not to be hacked.
It now appears that there is a way of using a hard disk cable to capture and send information via radio waves. It is potentially tricky to do, but possible.



Hackers can surreptitiously turn a computer cable into an improvised antenna that can imperceptibly transmit data, even from air-gapped* devices that are deliberately not connected to the Internet.
Computers that are not and cannot connect to the internet are often used by security services and infrastructure control systems to prevent hackers from gaining access anyway. But that does not mean it is impossible to get data out.
Mr. "Mordechai Guri" of the **Ben-Gurion University in the Negev, Israel,** has been working for years on developing a series of proof-of-concept attacks that use different parts of computers as unusual transmitters. In the past, he has found a way to extract information by encoding it in rapid adjustments to screen brightness, which cause deliberate temperature changes within a machine or rapidly and sharply changing **power LEDs**. It sounds very unlikely, but it worked.

His latest test attack was aimed at the **Serial Advanced Technology Attachment (SATA)** cables that connect CD, DVD and hard disk drives to the motherboard of most computers.
He discovered that by deliberately reading or writing a very specific set of redundant data to or from the drives, **the cables can initiate a radio wave of about 6 gigahertz.**
That wave can apparently be used to encode and send data to a waiting hacker standing nearby - no more than a few metres.
To get a computer to create these radio waves, a piece of **malware*,** which **Mr Guri** humorously calls **"SATAn"**, must be installed on this machine with this so-called **"air-gap"*.** This is certainly not an easy task, but it is possible. It is amazing how wide your imagination has to be to keep finding new possibilities.

WIKIPEDIA **Malware, or malicious software**, *is a blanket term for any kind of computer software with malicious intent. Most online threats are some form of malware.*

An **air gap**, *air wall, air gapping or disconnected network is a network security measure employed on one or more computers to ensure that a secure computer network is physically isolated from unsecured networks, such as the public Internet or an unsecured local area network. It means a computer or network has no network interface controllers connected to other networks, with a physical or conceptual air gap, analogous to the air gap used in plumbing to maintain water quality*

According to a report published in 2021 by security firm **ESET** (`https://www.eset.com/`), there are at least 17 types of **malware** that target machines with air-gaps, but they mainly use USB drives to infect machines.
They also use USB drives for the subsequent deletion of data, which is sent back to the attacker once the drive is plugged into an Internet-connected machine.

The report states that only one piece of malware, known as **"BadBIOS"**, has ever been claimed to use covert channels similar to **"SATAn"** to send data - but its existence is disputed by researchers. The **STUXNET WORM***, which targeted Iranian nuclear centrifuges and gave them commands that caused deliberate damage, was allegedly introduced into eavesdropping networks via a **USB stick**, but was not designed to delete data.

Mr Guri says he does not know if attacks similar to the one on **SATAn** are actually happening, but says they are highly plausible. "This attack is highly available because hard drives are found in all systems such as workstations and servers. In addition, the malware uses legitimate read and write operations on hard drives, which are very difficult to detect and identify as **malicious."**

THE SOLUTION

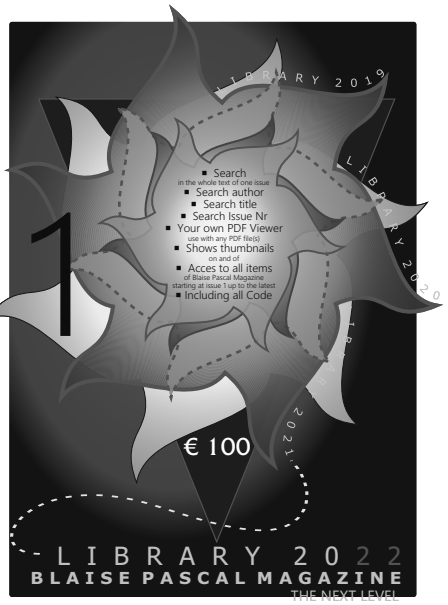
One solution could be a **Faraday** cage around the computer that blocks all electromagnetic signals, which would prevent this kind of attack, but for most applications this is just not practical. Another possible measure would be to cause constant noise by reading and writing redundant data to the hard disk, but this has the disadvantage of wearing out the component unnecessarily.

By Antoine Taveneaux - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4238427

A Faraday cage or Faraday shield is an enclosure used to block electromagnetic fields.
A Faraday shield may be formed by a continuous covering of conductive material, or in the case of a Faraday cage, by a mesh of such materials. Faraday cages are named after scientist Michael Faraday, who invented them in 1836.

A Faraday cage operates because an external electrical field causes the electric charges within the cage's conducting material to be distributed so that they cancel the field's effect in the cage's interior. This phenomenon is used to protect sensitive electronic equipment (for example RF receivers) from external radio frequency interference (RFI) often during testing or a alignment of the device. They are also used to protect people and equipment against actual electric currents such as lightning strikes and electrostatic discharges, since the enclosing cage conducts current around the outside of the enclosed space and none passes through the interior.

1



1

NEW DEVELOPED
LIB STICK
PROGRAM

1

SUPER OFFER
Normal Price € 271 € 150

LAZARUS HANDBOOK
FOR PROGRAMMING WITH FREE PASCAL AND LAZARUS

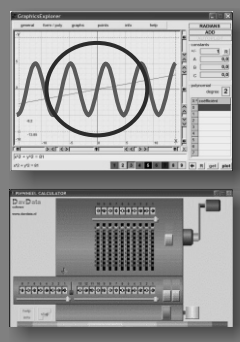
2 including 30 example projects

934 PAGES

3 including 19 example projects

LEARN TO PROGRAM USING LAZARUS
HOWARD PAGE-CLARK

DAVID DIRKSE
including 50 example projects



4

BLAISE PASCAL MAGAZINE
www.blaisepascal.eu
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

5

CIFAR-10 Image Classifier Maxbox
Lorentz waterwheel simulator
Viewing PDF files in the browser using Pas2Js
Hacking / Sata cable as antenna
Design-time components for Pas2js
Installing a Postgres Database
kbmMW WebSockets

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

https://www.blaisepascalmagazine.eu/product-category/special-offer/

BY MICHAËL VAN CANNEYT

starter expert

PAS2JS

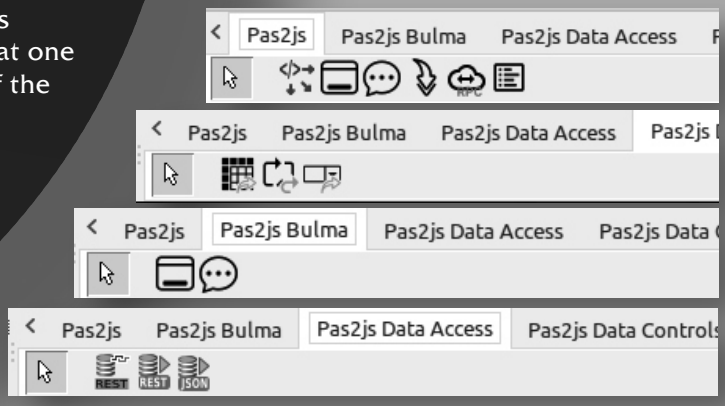
ABSTRACT

In this article we present a visual way to use some components that have been introduced in previous articles. Because of the use of the Database POSTGRES the installation and testing is explained in an extra article at Page .....of this issue. We also updated the creating article of a trunc version of **Lazarus** and **FPC.**
While **WYSIWYG** is not yet possible for **HTML** applications,it is perfectly possible to use the object inspector in **Lazarus** for speeding up development of **PAS2JS** applications considerably.

1 INTRODUCTION

In the previous articles about programming with **PAS2JS** everything was always done in code:
with the exception of some wizards to generate or update code, everything had to be done in the source editor.
However, **Delphi** and **Lazarus** are **RAD** environments *(Rapid Application Dvelopment)*. There is no intrinsic reason why **RAD** development should not be possible for **PAS2JS** applications.
The **RAD** mechanism in **Delphi** and **Lazarus** is based on a streaming mechanism:
a form file contains definitions for all components placed on a form or data module. At runtime, the components are created by the streaming mechanism, and the published properties which you manipulate using the **Object Inspector** are set from the values found in the `.dfm` or `.lfm` file.
The streaming mechanism exists in **PAS2JS**as well.
That makes it possible to create a form file in the **IDE** and use it to recreate at runtime - in the Javascript environment - the form definition that you created in the **IDE.**

Indeed, all classes presented in the previous articles - in particular, the widgets - have been created as descendents of `TComponent`, with the intent that one day they would usable in the **Object Inspector** of the IDE.
There is only one problem:
The components in the **PAS2JS** world assume a browser and do their work with the browser classes. So how can we compile these components for installation in the **IDE**, where the browser environment is not available ?
At this



2 STUB COMPONENTS

In order to use the **object inspector** for a component, the **IDE** only needs the published properties of a component. It does not need *(all)* the actual functionality of the components to be able to display them in the object inspector.
Keeping this in mind, it is possible to create a **'stub'** component: this is a component (*a native class*) that has the same published properties as the **PAS2JS** class, and enough code to
set the various properties and make them behave as they would at runtime:
if changing the value of a property changes another property, then the stub code should

❶ By taking the actual class and put all code that somehow refers to the browser in a conditional define:

```
Procedure TMyLabel.SetCaption(aCaption : string);
begin
  FCaption:=aCaption;
  {$IFDEF PAS2JS}
  // FElement is a TJSHTMLElement instance
  if Assigned(FElement) then  FElement.InnerText:=FCaption;
  {$ENDIF}
end;
```

This approach has the advantage that only one set of source files is needed:
one for the browser, one for the **IDE.** The disadvantage is that it is a lot of work to keep the code compilable on both platforms.

❷ By taking a copy of the class and removing all code that is not directly involved in the handling of the published property:
all public, private and protected methods can be removed.
Obviously, properties that descend from `TPersistent` must still be created.
The advantage is that the code is simple, the disadvantage is that there are two sets of files to be maintained.

PAS2JS comes with a tool *(makestub)* which helps creating such a stub class using the second method: it scans the original file for classes and outputs a new class with only the published properties.

It is not perfect, but can be used to make a first version of a stub.
Once the stub component is made, the component can be installed in the **IDE,** property and component editors can be registered and it can be dropped on a datamodule or a form.
Currently, the trunk version of the **Lazarus IDE** comes with 15 components that were made this way. They will be presented in the below.

3 DATA MODULES AND FORMS

A data module is a container for components that is not visible: it can be created like a form, components can be dropped on it, but at runtime it exists only in memory, it has no visible counterpart. It should not come as a surprise then that using a `TDatamodule` is possible in **PAS2JS:** you can create a datamodule in the **IDE,** drop some components on it and compile it.
If the components are supported by **PAS2JS** (*or have a stub*), the data module can be created at runtime, the form file will be read, and all will be as in a native application.

BUT HOW TO DO FORMS ?
Forms are deeply embedded in the **VCL** or **LCL,** and there is no **HTML** counterpart for this concept. Indeed, what would constitute a form in the browser?
The complete browser **HTML** window area or just a part of it?
Various answers are possible.
It would be possible to mimic `TForm` and most of its properties in the browser, but this would be enforcing an alien concept on the browser:
Visually, the browser is simply a powerful engine to display **HTML** in a window.
Trying to hide the use of **HTML** in **LCL-like** components is maybe not the best idea: rather, the power of **HTML** and **CSS** should be embraced.
The internet is full of beautiful frameworks, many components that render **HTML** directly. It makes sense to allow us to be able to use of this rich ecosystem.

So, how can we visually represent a part of the browser window in the **IDE** as if it were a form and allow the user to use the object inspector and component palette?

The answer is the `THTMLFragment` "form" component.
In the **IDE,** this is currently simply a `TDataModule` descendant, you can create it with the **'File-New'** menu; and drop components on it and manipulate them in the **IDE:** it has a form file as a datamodule or form, so it can be streamed.
You will notice the familiar resource directive in the source code:

```
{$R *.lfm}
```

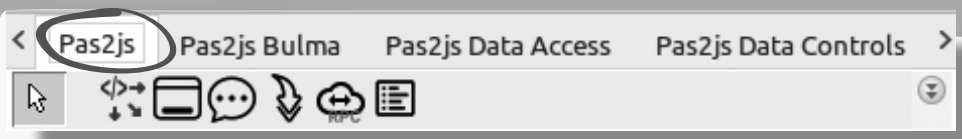
PAS2JS will recognize this directive and will link in the resource (*more about that later in this article*). With the `THTMLFragment` component, you can associate a fragment of **HTML.**
At runtime, this **HTML** fragment will be inserted in the browser **DOM** tree, at a location that you specify: much like the 'form' classes presented in the previous articles.

At the same time a component has been developed that allows to associate **HTML** elements to a pascal identifier, and attach event handlers for most common **HTML** events to this tag in a visual manner using the object inspector.
This component is called the `THTMLElementActionList` and can be found on the component palette. Its workings are analogous to the `TActionList` class found in the **LCL** and **VCL.**

4 THE PAS2JSCOMPONENTS PACKAGE.

The concepts presented in the previous sections have been implemented in a package `pas2jscomponents`.
The package is available in the trunk version of Lazarus in the directory `Components/Pas2js/Components`. When installed in the **IDE,** *(currently)* 15 components are installed on the component palette on 4 different tabs:

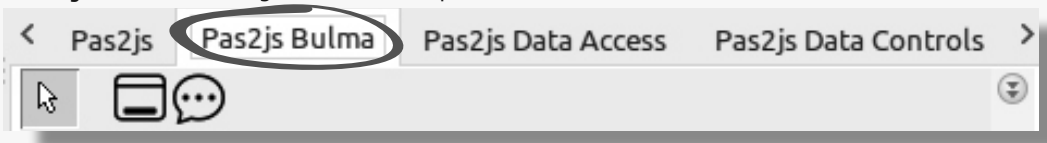
PAS2JS Some general purpose components, plus some **Bootstrap CSS** based UI-omponents.



The components are (*in order of appearance on the component palette tab*):

- `THTMLElementActionList` a component to access the **HTML** tags and associate events handlers to various events in a visual manner.
- `TBootstrapModal` a bootstrap modal dialog component.
- `TBootstrapToastWidget` a bootstrap toast message component.
- `TPas2JSRPCClient` The RPC Client presented in the previous pas2js article. The name is changed to prevent a clash with the native `TRPCClient` class.
- `THTMLEditor` a component for a simple HTML editor.

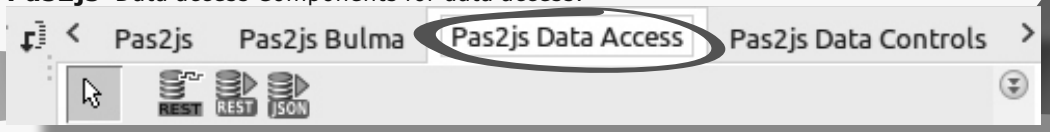
Pas2js Bulma A dialog and toast component based on Bulma CSS



2 components based on Bulma CSS:

- `TBulmaModal` a Bulma-based modal dialog component.
- `TBulmaToastWidget` a Bulma-based toast message component.

Pas2js Data access Components for data access.

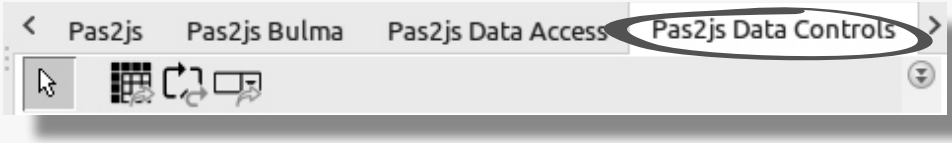


This tab contains 3 components for data access:

- `TSQLDBRESTConnection`
  The connection component for a **SQLDBRest** server, presented in the previous article.
- `TSQLDBRESTDataset`
  The dataset component for a **SQLDBRest** server, presented in the previous article.
- `TLocalJSONDataset`
  A local dataset component: you can use this for storing local data, by defining a set of fields, like in `TClientDataset` in **Delphi,** or `TBufDataset` in **Lazarus.**
  The component can load and save the **(JSON)** data from and to the browser's local storage.

Pas2js Data Controls

These controls are data-aware: they generate **HTML** based on the contents of a dataset.



The components are (*in order of appearance on the component palette tab)*:

- `TDBBootstrapTableWidget` Fill a grid based on a dataset.
  It uses bootstrap-table for the styling and handling.
- `TDBLoopTemplateWidget` Repeat a HTML snippet for every record in a dataset, replacing template variables based on the field values in the dataset.
- `TDBSelectWidget` Fill a **HTML SELECT** tag with **OPTION** tags, based on records in a dataset.

In addition to the components, the pas2jscomponents package registers the `THTMLFragment` as an item in the **'File' - 'New'** dialog, as shown in figure 1 on page 5.

5 A DEMO APPLICATION

To demonstrate the use of these components and the visual `HTML`,
we'll rewrite the application we created in the previous article using the `HTMLFragment`
and the visual components.

For some components, nothing changes:
instead of creating them in code and setting the properties,
this can now simply be done visually.

For some others, some changes are needed.
As a quick reminder, the application was a small **blogging application**.
The server part has an **RPC** mechanism to log in, the client has several forms:

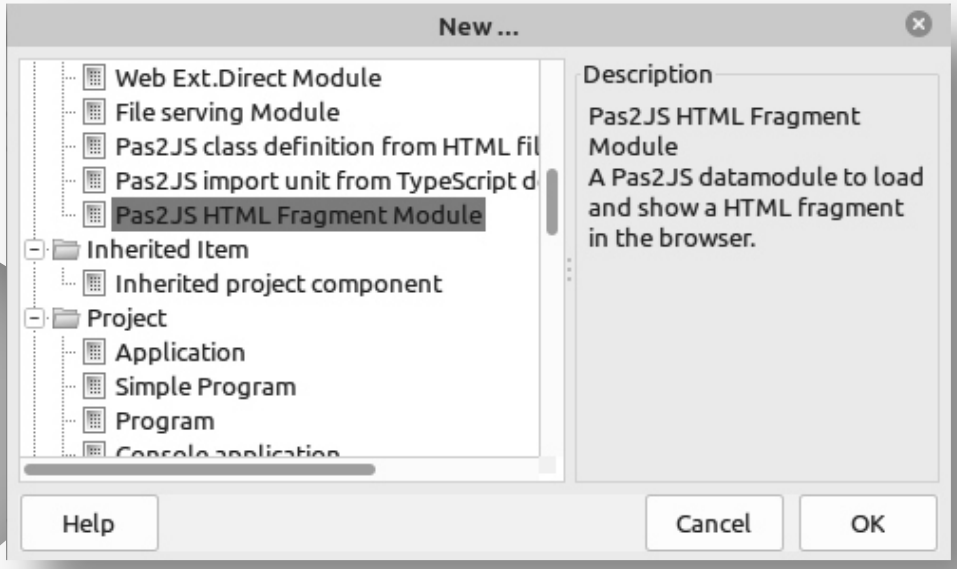


Figure 1: The HTMLFragment menu item

❶ A data module which has a **RPC** client and the **SQLDB Rest** connection.
❷ A main form to handle the main **HTML** page `(index.html)`
❸ A form to log in.
❹ A form to show an overview of available posts
❺ A form to view or edit a post.

The **server part** does need one change: the `sqldbrestcds` unit must be added to the uses clause. The reason for this is that the design-time support for various components queries the server and expects data to be returned in the format read by `TBufDataset` format.
Adding the `sqldbrestcds` unit to the server enables the support of this format.
So the server program can be compiled as it was (*with the added* `sqldbrestcds` *unit),* and started - this is needed to enable some of the data handling functionalities described below.

The **client part** needs more changes, obviously.
To get started, you need of course to have the latest trunk version of lazarus (see the article that explains how to do this **in the last issue 103/4 page...)** and the package **pas2jscomponents** needs to be installed.
Just as in the original application, we again start with creating a new **PAS2JS** browser application, and we set the option to use the `TBrowserApplication` class.
After this, we use the '**File - new**' dialog to create a data module:
we will name it `ServerModule` and save the unit with the name `dmServer`.

On this data module, we drop 2 components from the Pas2js and Pas2js Data access tab:

1. a `TPas2JSRPCClient` component. We call it `RPCClient` set the **URL** property to `http://localhost:8080/RPC` if you used another port than 8080 for the server application, obviously you must set the URL property accordingly.

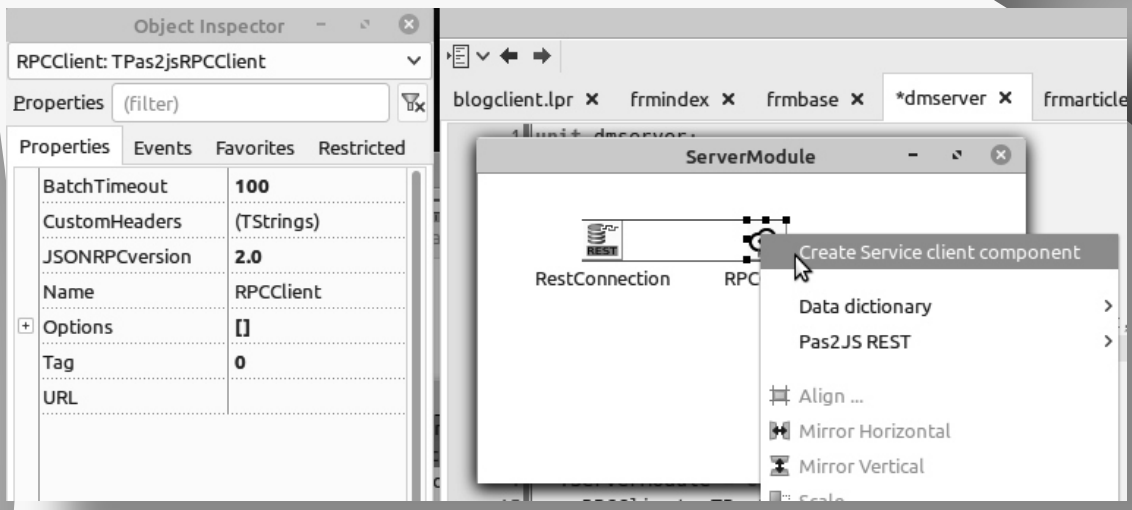


Figure 2: The context menu of the RPCClient component.

2. a `TSQLDBRestConnection` component.
   We call it RestConnection and set the BaseURL property to `http://localhost:8080/REST`
   Here as well, the port number must be changed if necessary.

The IDE support for `TPas2JSRPCClient` contains a wizard to generate the service definition: in the context menu of the `RPCClient` (see figure 2 on page 6), select **'Create Service Client component'**.
This will bring up the '**Service Client generation**' dialog. The dialog does the same as the `apiclient` website page presented in the second article in this series:
based on the **RPC** service **URL,** it creates a unit with a client-side proxy class for the service that has all the calls presented by the service. It has some options, most of which are obvious in what they do:

❶ The URL of the service, this is taken from the RPCClient component.
❷ The name of the unit to generate.
❸ The file name of the unit to generate.
❹ Options which influence the code: use `NativeInt` for number-typed arguments and an option to force the use `JSValue` for result values in callbacks.
❺ A button to show a preview of the code in a memo.
❻ Options to add the unit to the project, and copy the code to the clipboard. The latter is useful if you want to have multiple services in 1 unit: you can simply paste the declaration into an already existing unit.

All these options can be seen in figure 3 on page 7.
The class can be generated and renamed (`TUserService`) to match the class name used in the previous version of the project. In the data module, we define again a variable for the service, and add the login method with 2 events to signal success and failure, just as in the previous version of this program. The `OnCreate` event of the data module can be used to configure the connection at runtime:

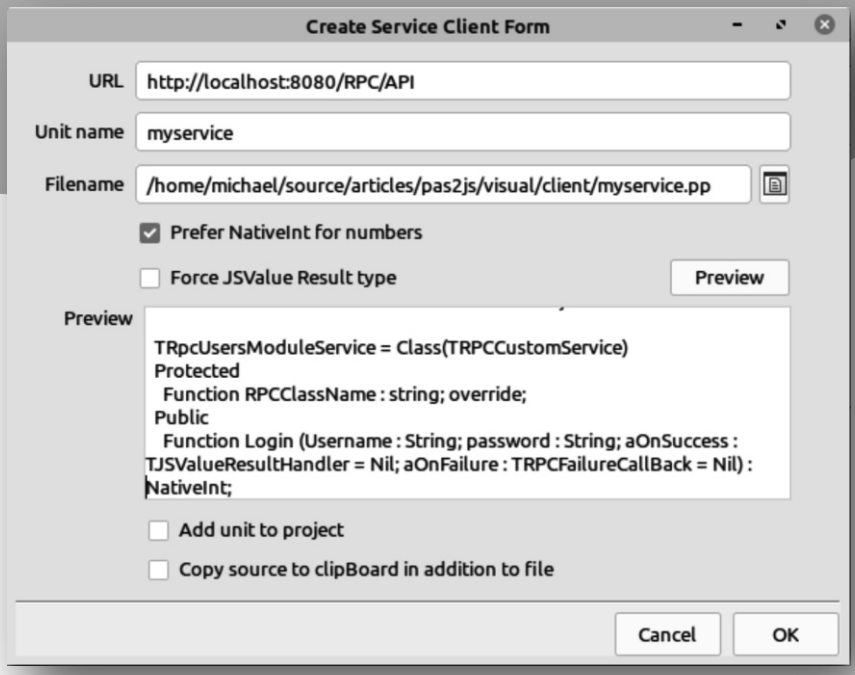


Figure 3: The dialog to generate a client class

```
Const
  ServerURL = 'http://localhost:8080/';

procedure TServerModule.DataModuleCreate(Sender: TObject);
begin
  FUserService:=TUserService.Create(Self);
  FUserService.RPCClient:=RPCClient;
  RPCClient.URL:=ServerURL+'RPC/';
  RestConnection.BaseURL:=ServerURL+'REST/';
end;
```

The login method remains exactly as it was in the first version of our application: we can simply copy and paste it: no code changes are necessary.

```
procedure TServerModule.DoLogin(const aUserName, aPassword: String);

    procedure DoOK(aResult: NativeInt);
    begin
       FUserInfo.Login:=aUserName;
       FUserInfo.ID:=aResult;
    if Assigned(FOnLogin) then
       FonLogin(Self);
    end;

    procedure DoFail(Sender: TObject; const aError: TRPCError);
    begin
       FUserInfo.Login:='';
       FUserInfo.ID:=-1;
       If Assigned(FOnInvalidLogin) then
       FOnInvalidLogin(Self);
    end;

begin
  UserService.Login(aUserName,aPassword,@DoOK,@DoFail);
end;
```

6 ACCESSING HTML ELEMENTS

In the previous articles we used the lazarus **IDE** wizard to generate a "form" class definition from a **HTML** page, where the form would have a field for every **HTML** element in the **HTML** page. We could do the same here and tell the wizard to use the `THTMLFragment` class as a parent. Doing so leads to a small problem: the wizard does not generate a form file, a problem which still needs to be solved. But there is another way: we can do this differently and in a manner which will allow us to work in visual (*point-and-click*) manner for creating the **HTML** element events. We will show this by starting with the main html file `(index.html)`. As a reminder: The index **HTML** file serves as the shell for the various forms in the application, and contains the application menu as well as the login menu and hamburger menu for responsive design. It also contains the parent **HTML** tag where the various pages of the Single-Page-Application are shown. These **HTML** tags are accessed from the login page to enable/diasble the menu and login menu items. So we make a 'form' for this page.

The 'File' - 'New' menu item **"Pas2JS HTML fragment module"** can be used to create a form-like enviromnent. The `THTMLFragment` class corresponds to a 'form': a fragment of **HTML** that will be downloaded and inserted in the main html page. This form (*we'll call it "form" for ease of use*) has several properties.

- **HTMLFileName**
  if set, this is the **HTML** file that will be downloaded by the templateloader.
  If not set, the name of the file is the `TemplateName`, in lowercase and with extension `.html`
- **ParentID**
  The **ID** of the parent **HTML** element for this form.
  This is the element below which the **HTML** for this form will be inserted.
- **TemplateName**
  if set, this is the name of a global template that will be used as the **HTML** form.
- **UseProjectHTMLFile**
  If set, no extra **HTML** will be loaded, and the main **HTML** file of the project will be used to find **ID**s for tags etc.

THERE ARE SOME EVENTS:

- **OnAllowUnrender**
  This will be called before unrendering (removing) the form's **HTML,** and can be used to disallow unrendering: in case there is data to be saved.
- **OnRendered**
  This event will be called when the form **HTML** is inserted in the parent **HTML** tag. At this moment, the **HTML** is inserted in the **DOM.**
- **OnHTMLoaded**
  This event will be called when the form **HTML** has been downloaded but is not yet inserted in the parent HTML tag.
- **OnUnRendered**
  This event will be called when the form HTML has been removed from the browser's DOM tree.

IT HAS FEW METHODS:

- **Render**
  Render the **HTML** by inserting it in the **DOM** tree. This method is **synchronous** and can only be called if the **HTML** for the form has already been downloaded.
- **UnRender**
  Remove the form's **HTML** by removing it from the **DOM** tree. This method is **synchronous.**
- **Show**
  Download and subsequently render the **HTML** by inserting it in the **DOM** tree.
  This method can be **asynchronous** and can be called even if the **HTML** for the form is not yet available: it will use the `HTMLFileName` and `TemplateName` to use the global templateloader (*presented in an earlier article in this series*) and download the necessary **HTML.**
- **Hide**
  Will check if the form is rendered and if so, unrenders it.

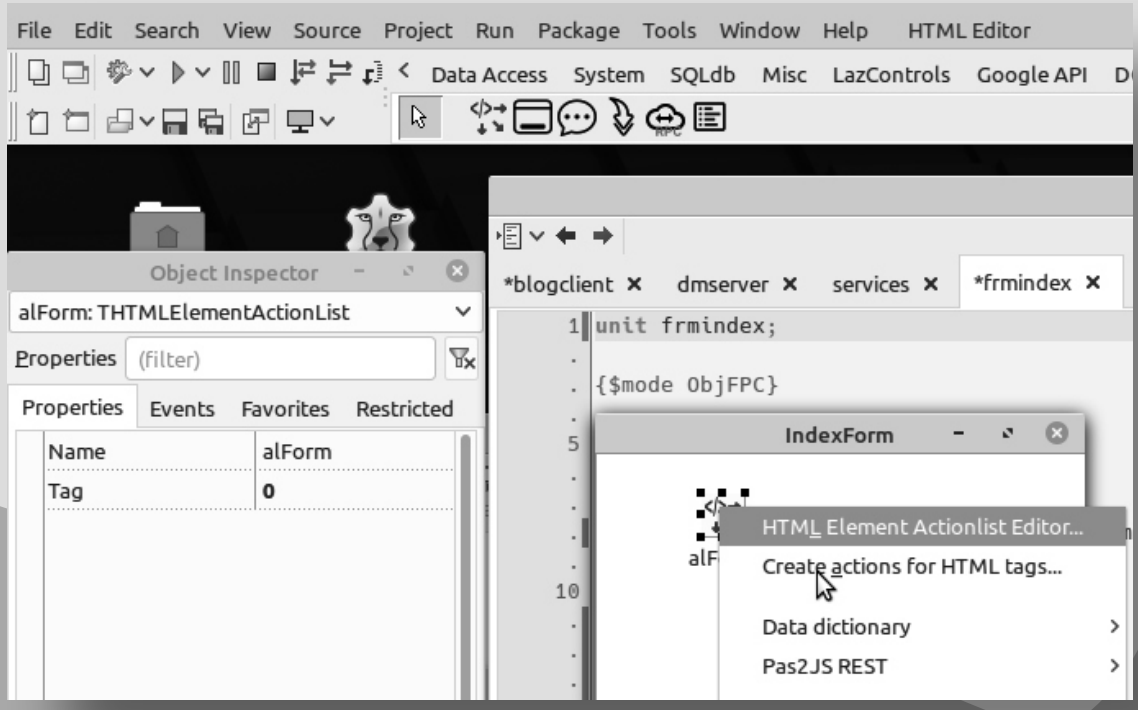


Figure 4: The THTMLElementActionList context menu.

The reason that there are so little properties and methods is simple:
There is not much else to do in the concept of a **HTML Fragment**. Designing the form is expected to happen in a **HTML** editor (*you can use the Lazarus IDE for this*).
For capturing events, we can use another component described below.

Once the `THTMLFragment` is created, we will name it `TIndexForm` and call the unit `frmIndex`. Then we can drop a `THTMLElementActionList` component on it from the **'PAS2JS'** tab of the component palette.

This component is modeled after the `TActionList` component: it has a component editor

- **HTML Element Actionlist Editor**
  this will show the element actionlist editor, a component editor that closely resembles the `TActionList` editor found in the **LCL.**

- **Create actions for HTML tags**
  this will scan the **HTML** file for **HTML** element tags that have their id attribute set, and will create an action for each of them.

There are various action classes possible: a `THTMLElementAction` is a `TComponent`, just like `TAction` in the **LCL.** As such, it will be inserted in the form and hence is available in code, you can do code completion on it etc.

When you select the `Create actions for HTML tags` menu item, a dialog will pop up that shows the ids of HTML tags for which no `THTMLElementAction` definition exists yet. The dialog is *shown in figure 5 on the next page: 11 / 18 of this article*

As can be seen in the editor, you can set an option to use Data-aware actions:
setting this option this will instruct the wizard to use data aware actions for all input elements and buttons.
You can select the action class to use for each element.
Once done, the necessary component definitions will be inserted in the form.
The wizard will choose default names for the various actions: 'act' appended with the id attribute of the tag (*suitably modified so it is a valid pascal identifier*).

The `THTMLELementAction` class has the following published properties:

- **Events**
  A set of events for which you want to create an event handler. One event handler is created that handles all events.
- **CustomEvents**
  The names of custom events, not present in the standard events list. These event names must be separated by commas or spaces.
- **ElementID**
  The 'id' attribute of the HTML tag to which to bind this action. This will result in a single HTML tag being bound.
- **CSSSelector**
  a CSS selector: this will be used using querySelectorAll to bind one or more HTML tags to this action.
- **PreventDefault**
  when set to True, the preventDefault method of the event object will be called after the OnExecute event was executed.
- **StopPropagation**
  when set to True, the stopPropagation method of the event object will be called after the OnExecute event was executed.
- **OnExecute**
  This event handler is executed for all selected events. You can see which event was executed by examining the event parameter.
- **BeforeBind**
  This event is executed just before the action is bound to the HTML tags.
- **AfterBind**
  This event is executed just after the action is bound to the HTML tags.

NOTE that only a single event handler is created for all possible HTML events. It is possible to create 2 actions for the same HTML tag, and handle different events in each action. Besides the published properties, there are also some public properties which can be used in code:

- **ActionList**
  The action list of which the action is part.
- **Element**
  In case an was specified, the **HTML** element (of type `TJSHTMLElement`) to which the action is bound.
- **Elements**
  An array of **HTML** Elements corresponding to the **CSS** Selector.
  In case an element **ID** was specified, the array will contain 1 element, corresponding to the **Element** property;
- **Index**
  The index of this action in the `ActionList`
- **Value** When the property is read, this returns the `value` of the first element in the `Elements` array. In case of writing to the property, the value will be set for all elements in the array.

- **Value** can be set on any **HTML** element: For `Input` and `Select` **HTML** tags, **Value** is the `value` property of the tag. For all other tags, this is the value of the `InnerHtml` property.
  It is possible to manipulate the **HTML** tag using the `Element` property.

For ease of use, there are some methods in the action:

- **Bind** Call this to bind the action again to the **HTML** tags: This is called automatically when the form is rendered, but you can call it again after the **HTML** inside the form has dynamically changed. For instance, it can be used to bind the onclick event to all rows of a table using the selector `'#mytable tr'`. If the table was filled with rows dynamically, you would call Bind to get the array of row elements.
- **ClearValue**
  Clear the value of the element.

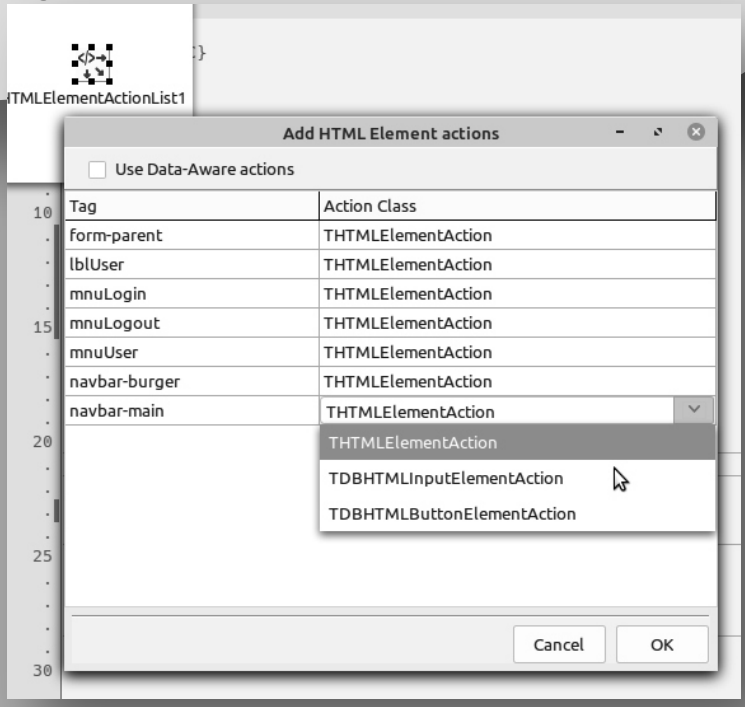


Figure 5: The 'Add HTML Element actions' dialog.

- **FocusControl**
  Focus the element.
- **BindEvents**
  Bind all events for a given element.
- **HandleEvent**
  Handle an event and call the event handlers
- **ForEach**
  execute a callback for each element in the array of elements for this action.
  Optionally a Data object can be specified which is passed to the callback.
- **AddClass**
  add a class to the list of **CSS** classnames for the **HTML** tag(s).
  The argument must be a single **CSS** class name.
- **RemoveClass**
  remove a class from the list of **CSS** classnames for the **HTML** tag(s).
  The argument must be a single **CSS** class name.
- **ToggleClass**
  toggle a class in the list of **CSS** classnames for the **HTML** tag(s).
  The argument must be a single **CSS** class name.

Descendents of `THTMLElementAction` may have more properties and methods.

These action classes can be used to interact with the **HTML;** just like for classical controls, the events can be created in a point-and-click manner. If the form **HTML** has a toplevel tag with an **ID** attribute, events can be handled for the whole form as well:
In that case you may need to set the `StopPropagation` to `True` on individual elements to prevent the event from being handled twice.

For the index form, the actions can be created, but no events are needed: the events will be handled in the login form. The menu is initially disabled, so we do add a method to show the menu when the user logs in. This method (`StartLogin`) demonstrates some of the methods and properties of the action class:

```
procedure TIndexForm.StartLogin(const aUserName: String);
begin
  mnuUser.RemoveClass('is-hidden');
  mnuLogout.removeClass('is-hidden');
  mnuLogin.AddClass('is-hidden');
  lblUser.Value:=aUserName;
end;
```

For the login form, we do the exact same as for the index form. But in this case, we fill in the `TemplateName` and `HTMLFileName`.
The context menu for the `THTMLElementActionList` and in particular the 'Create actions for **HTML** tags' menu item will know in what file to look for the **HTML** tags and their **ID** attributes: the wizard will create actions for the login button and the input tags for username and password.
Only the login button needs an 'onclick' event handler:

```
procedure TLoginForm.actbtnDoLoginExecute(Sender: TObject; Event: TJSEvent);
begin
  server.DoLogin(edtEmail.Value,edtPassword.value);
end;
```

The server module still has the OnLogin and OnInvalidLogin events, which we assign in the OnCreate event of the login form:

```
procedure TLoginForm.DataModuleCreate(Sender: TObject);
begin
  Server.OnInvalidLogin:=@DoLoginFailed;
  Server.OnLogin:=@DoLoginOK;
end;
```

7 TOAST COMPONENTS

In the `OnLogin` or `OnInvalidLogin` event handlers, a **Bulma** toast was shown. The **Pas2js** tab of the component palette contains a **TBootstrapToastWidget,** and the **Pas2js Bulma** tab contains a `TBulmaToastWidget`.
These components can be used to show a toast message.

THE BULMA TOAST HAS THE FOLLOWING PROPERTIES:

- **Header**
  A string with the HTML for the toast's header.
- **Body**
  A string with the HTML for the toast's body.
- **HeaderImage**
  A URL used to add an image to the header.
- **CloseButton**
  A boolean indicating whether to show a close button or not.
- **Contextual**
  Set the color scheme for the toast: The TContextual type is an enumerated with the primarys colors used in Bulma: link, warning, danger etc.
- **HideDelay**
  The delay - in milliseconds - to wait before hiding the toast.
  Only effective when AutoHide is True.
- **AutoHide**
  A boolean to indicate whether the toast must hide automatically after some delay.
- **Animate**
  A boolean to indicate whether to use an animation to show or hide the toast.
- **MinWidth**
  An integer indicating the minimal width of the toast.
- **Single**
  Single toast or multiple toasts ?
- **Position**
  The position of the toast on the browser form: An enumerated that can be set to top-left, bottom-right etc.
- **ParentID**
  The ID of the HTML Element where the toasts will be rendered. If none is set, the ParentID of the toast manager (toasts) will be used, if that is not set, the body element is used.

The Bootstrap toast component has the same properties.
So, to show a toast on succesful login, we drop 2 bulma toasts on the form, enter some text in the `Body` and `Header` properties and set the parameters for `Contextual, autohide`.
The resulting form in the designer looks as in figure 6 on page 14.

All that's left to do is to call **Refresh** in the appropriate places, which will actually render the toast:

```
procedure TLoginForm.DoLoginFailed(Sender: TObject);
begin
  FailedToast.Refresh;
end;

procedure TLoginForm.DoLoginOK(Sender: TObject);
begin
  OKToast.Body:='Hello, '+Server.UserInfo.Login;
  OKToast.Refresh;
  IndexForm.StartLogin(Server.UserInfo.Login);
  Router.RouteRequest('/articles',True);
end;
```

As you can see, at the end of the **DoLoginOK** we have the same code as in the previous version of the application: a call to route the brower to the articles form. The form manager class we introduced in the article on routing remains as it was, but there is a minor change: the **TBaseForm** form that underpins all forms is now a descendent of the **THTMLFragment** class.
We just need to make sure that our forms are descendents of **TBaseForm**. Ideally, we could create a **Lazarus IDE** wizard to create a descendent of **TBaseForm**, but a simpler method can also be used. We simply create an alias for the **THTMLFragment** class:

```
uses
  SysUtils, Rtl.HTMLActions, htmlfragment, Web,
  bulmawidgets, frmBase;
type
  THTMLFragment = TBaseForm;
  TLoginForm = class(THTMLFragment)
 // etc.
```

Using this method, all the code for displaying forms as it was presented in the article on routing remains functional, but behind the scenes the **THTMLFragment** class is used, which allows us to remove the code to actually download the **html** from the **TFormManager** class, as this part is now handled by the **THTMLFragment** class.

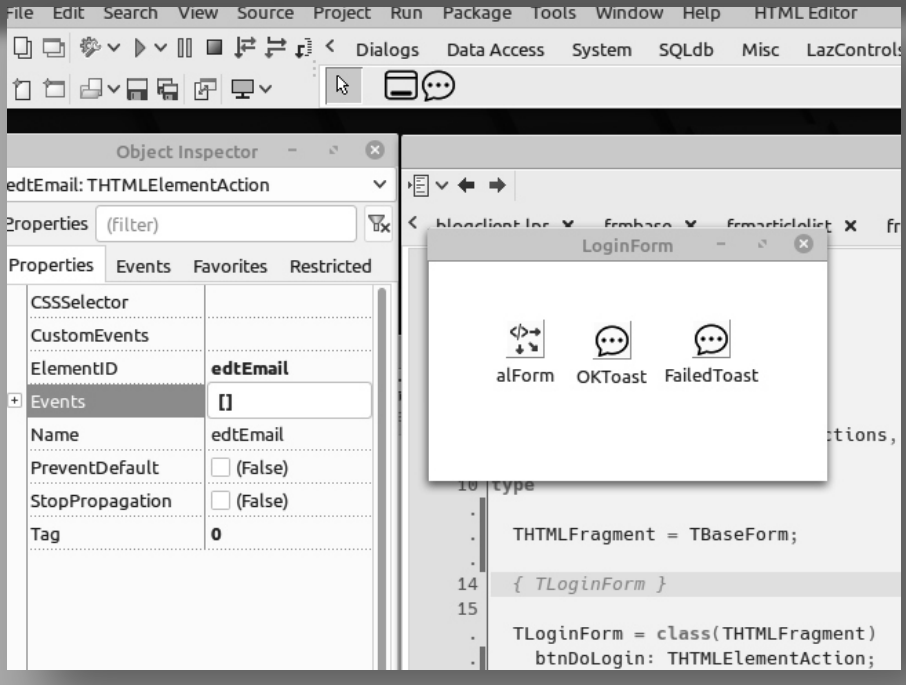


Figure 6: The login form in the designer

8 USING DATA-AWARE COMPONENTS

The article list form used a dataset to fetch a list of articles, which were subsequently shown with the `TDBLoopTemplateWidget` component class. Since the template widget is a component, it is available on the component palette. The same can be said for the `TDBSelectWidget` that is used to display a select element to filter the authors. Recreating the article list form with these components is easy to do:

- We create a **pas2js** `HTMLFragment` form and set the `HTMLFilename` property to `articles.html`.
- We create a `THTMLElementActionList` on the form and let it create actions for all elements.
- We drop 2 datasets (*for articles and authors*) on the form and connect them to the connection component on the server datamodule.
- We drop 2 datasources (`dsArticles` and `dsAuthors`) and connect them to the datasets.
- We drop a `TDBLoopTemplateWidget` component (`ltwArticlesList`) and connect it to the `dsArticles` datasource.
- We drop a `TDBSelectWidget` component (`dswAuthors`) and connect it to the `dsAuthors` datasource.

The `TSQLDBRestDataset` component has a context menu with a `Pas2JS REST` submenu in it (*figure 7 on page 16*). This submenu has 2 entries:

- **Create field defs**
  Using this menu you can create fielddefs for the REST resource indicated in `ResourceName`. After this, the fields editor can then also be used to create persistent fields for the dataset.
- **Show data** Using this menu you can actually see the data of the REST resource in a grid, as shown in figure 8 on page 16.

Note that the **SQLDBRest** server must be running for these menu items to work.

The various template properties of the `TDBSelectWidget` and `TDBLoopTemplateWidget` component can now be edited using the **Object Inspector**: a property editor that shows a **HTML** editor with syntax highlighting is available for editing the **HTML** snippets, *as shown in figure 9 on page 16.*

The author selection dropdown (`dswAuthors`) contains an `ItemField` property which determines the field to show in the dropdown. It is set to `u_author`.

This is a calculated field which we calculate in the `OnCalcFields` event of the `rdsAuthors` dataset:

```
procedure TArticleListForm.rdsAuthorsCalcFields(DataSet: TDataSet);
var aAuthor : String;
begin
  aAuthor:=rdsAuthorsu_firstname.AsString;
  aAuthor:=aAuthor+' '+rdsAuthorsu_lastname.AsString;
  if (rdsAuthorsu_id.AsLargeInt=Server.UserInfo.ID) then
    aAuthor:='You ('+aAuthor+')';
  rdsAuthorsu_author.AsString:=aAuthor;
end;
```

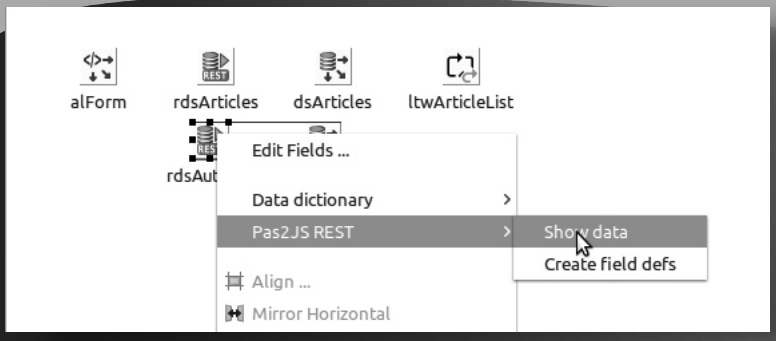


Figure 7: The TSQLDBRestDataset component context menu

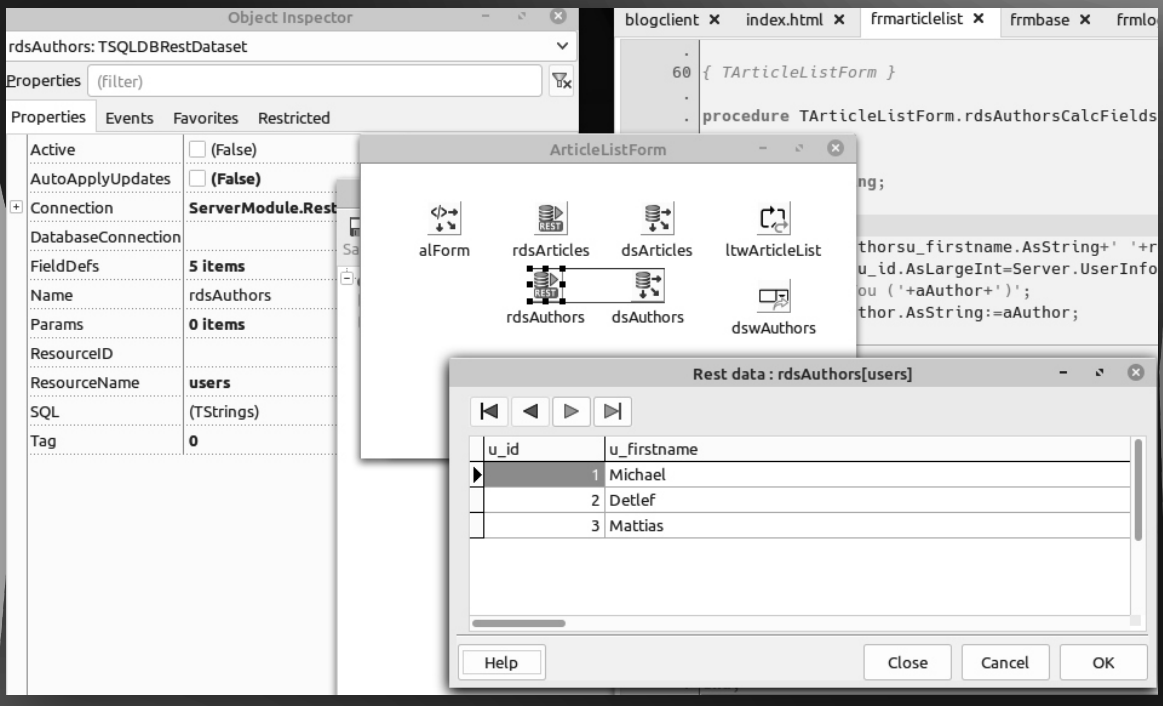


Figure 8: Showing the REST resource data

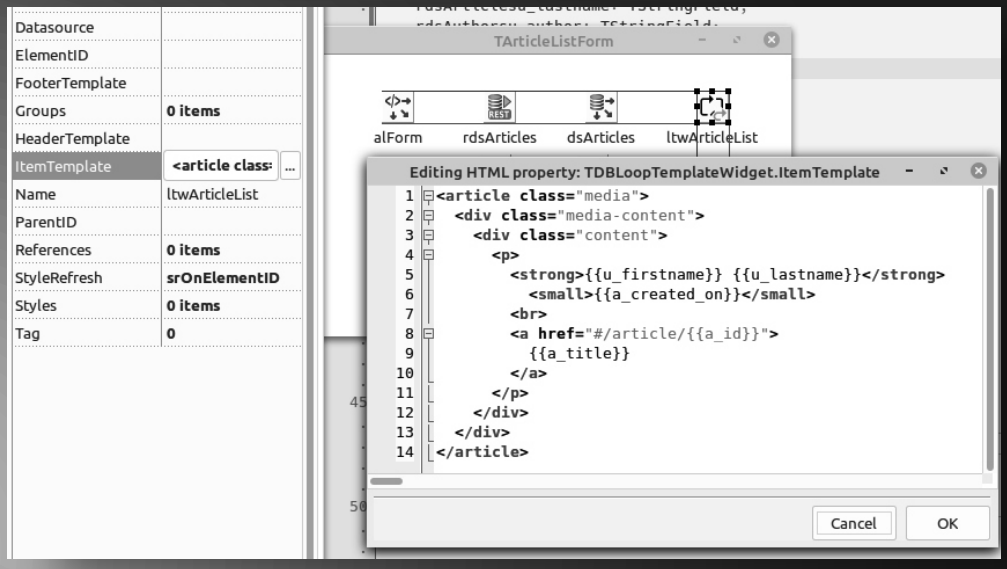


Figure 9: Editing the HTML templates using a property editor

Here we take advantage of the persistent fields: the compiler will check all code, and code completion helps us not to make any mistakes against the fieldnames. The `OnChange` event handler of the `dswAuthors` element can now be implemented much as it was in the previous version of our program:

```
procedure TArticleListForm.dswAuthorsChange(Sender: TObject; Event: TJSEvent);
Var aAuthorID : Integer;
begin
  aAuthorID:=StrToIntDef(selAuthor.value,-1);
  With rdsArticles.ParamByName('a_author_fk') do
   begin
     AsInteger:=aAuthorID;
     Enabled:=(aAuthorID<>-1);
   end;
  rdsArticles.Close;
  rdsArticles.Load;
end;
```

Note that we do not create the parameter `a_author_fk` in code, because we can use the object inspector to create it, *as seen in figure 10 on page 18.* The `AfterOpen` event handler of the dataset can now also be created using the **object inspector:**

```
procedure TArticleListForm.rdsArticlesAfterOpen(DataSet: TDataSet);
begin
  if Dataset.IsEmpty then
    divNoArticles.RemoveClass('is-hidden')
  else
    divNoArticles.AddClass('is-hidden');
end;
```

As you can see the `divNoArticles` component (of type `THTMLElementAction`) has a convenience method to add or remove **CSS** classes.
Last but not least, the `OnRendered` method of the form can be used to load the data:

```
procedure TArticleListForm.DataModuleRendered(Sender: TObject);
begin
 if Server.UserInfo.ID>0 then
   divNewArticle.removeClass('is-hidden');
   rdsArticles.Load;
   rdsAuthors.Load;
end;
```

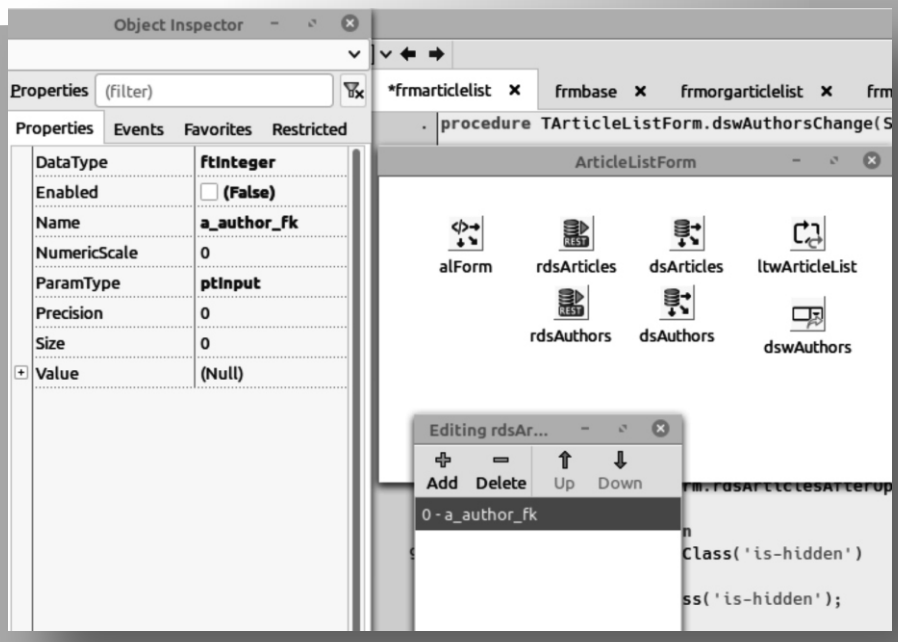


Figure 10: Creating filter parameters using the Object Inspector

A word of caution: **do not load the data** in the `OnCreate` method. After the create, the form **HTML** is not yet rendered, and the data-aware components will not be able to render the data if the data arrives before the **HTML** fragment. Similarly, the `divNewArticle` element action is not yet bound to the **HTML** tag, and the `removeClass` method will not work yet.

9 READY TO RUN

The last form to convert is the article form. In fact, it is completely similar to converting the article list, no new mechanisms are used, so we will not present it here.
The application class of our application can remain exactly the same as it was before, and with this the application is ready. Compile and running will however result in an error in the browser, as shown in figure 11 on page 19.
The reason for this is that the pas2js compiler allows 2 kinds of resources:

- Resources added as **Javascript** in the application `javascript` files.
- Resources added as link elements using a data url.

Since we didn't tell the compiler which resource mechanism to use, the compiler did not include our form files in the application. The solution is to add the `-JRjs` option in the custom compiler options, as can be seen *in figure 12 on page 18*. Once that is done, compiling and running the program will work as before. *See figure 13 on page 19.*

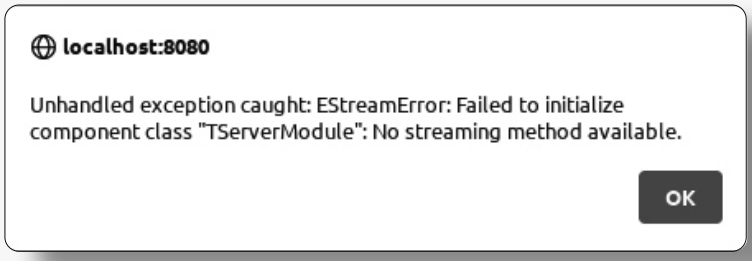


Figure 11: An error when running the application in the browser

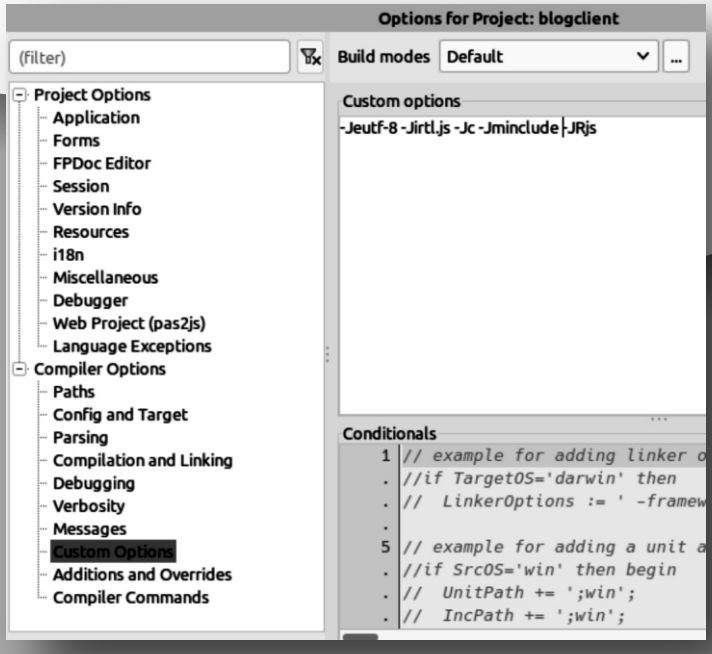


Figure 12: Adding the resource type to the compiler options

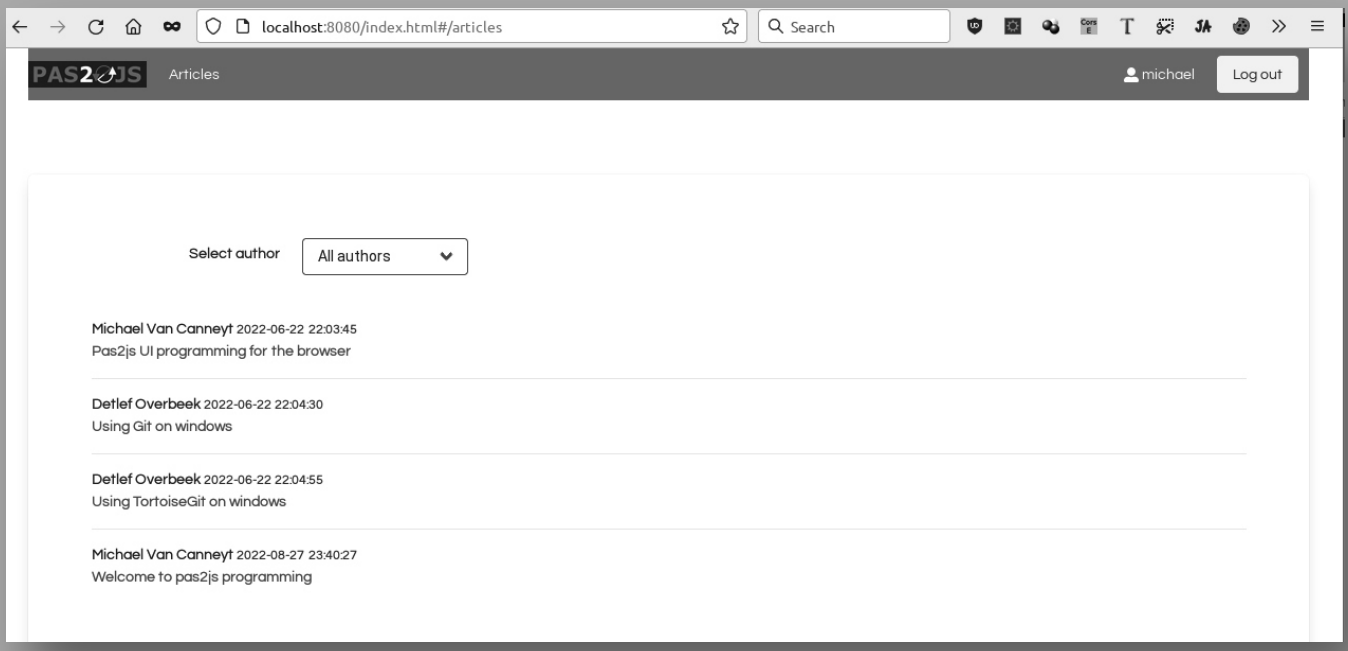


Figure 13: The reworked program in action

10 CONCLUSION

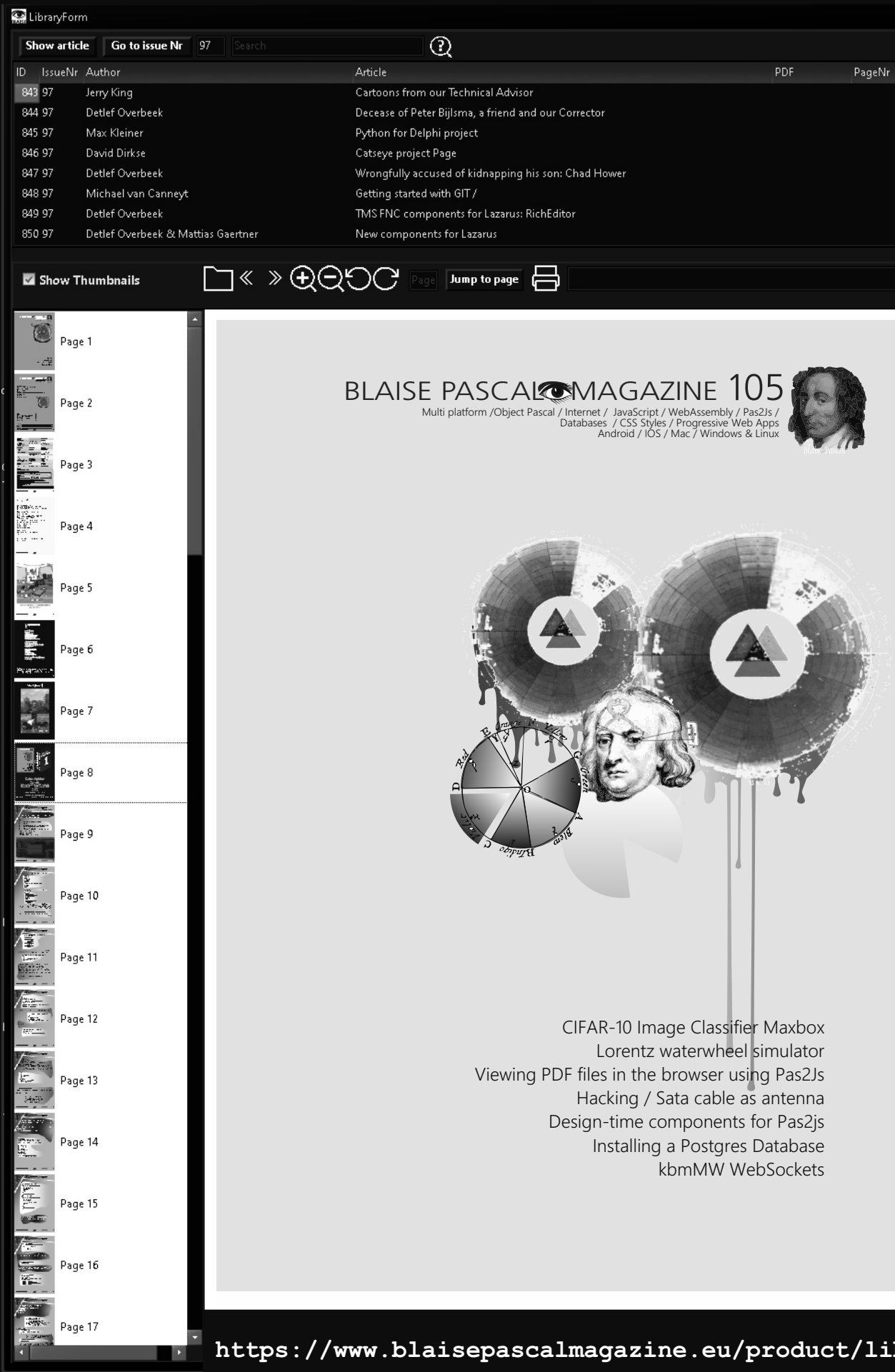
The support for visually designing **HTML** applications in the **Lazarus IDE** is a work in progress:
although the current state is not yet **WYSIWYG,** the use of 2-way code editing tools is already possible. It makes working on an **HTML** application easier, and reduces the need for writing code considerably. The same components will be usable once the actual **HTML** designer works, so the way of working presented here will not change once the **HTML** editing is actually available: at that point you will simply be able to see the **HTML** in the designer, instead of a white background.

LibraryForm

Show article | Go to issue Nr | 97 | Search

ID	IssueNr	Author	Article	PDF	PageNr
843	97	Jerry King	Cartoons from our Technical Advisor		5
844	97	Detlef Overbeek	Decease of Peter Bijlsma, a friend and our Corrector		6
845	97	Max Kleiner	Python for Delphi project		9
846	97	David Dirkse	Catseye project Page		24
847	97	Detlef Overbeek	Wrongfully accused of kidnapping his son: Chad Hower		29
848	97	Michael van Canneyt	Getting started with GIT /		34
849	97	Detlef Overbeek	TMS FNC components for Lazarus: RichEditor		49
850	97	Detlef Overbeek & Mattias Gaertner	New components for Lazarus		60

Show Thumbnails | Page | Jump to page



https://www.blaisepascalmagazine.eu/product/lib-stick/

starter expert

ABSTRACT

The installation example of the **Postgres Database** is necessary because of the use of this Database in the project of the article by Michael van Canneyt in this issue about **DESIGN TIME COMPONENTS FOR PAS2JS** (*Article starting at page ....1/ 19 of the issue*).
You can of course try to use your own: **Firebird, MySQL** etc.

We have chosen for this database because for the moment **Postgres** seem to be the best database available. The installation in itself is very simple but varies per **OS.** After the installation it would be wise to test the installation and therefore Lazarus has a beautiful little program which you always can use. This app needs some extra explanation and I will show how to use it.

INRODUCTION FOR THE POSTGRESQL DATABASE

PostgreSQL is a powerful, open source object-relational database system that uses and extends the **SQL** language combined with many features that safely store and scale the most complicated data workloads.
The origins of **PostgreSQL** date back to 1986 as part of the **POSTGRES** project at the **University of California** at Berkeley and has more than 30 years of active development on the core platform.
PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions. **PostgreSQL** runs on **all major operating systems,** has been **ACID***- compliant since 2001, and has powerful add-ons such as the popular **PostGIS*** geospatial database extender. It is no surprise that **PostgreSQL** has become the open source relational database of choice for many people and organisations.

Logo of Postgres at the top is made by Daniel Lundin https://wiki.postgresql.org

WHY USE POSTGRESQL?

PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, **PostgreSQL** is highly extensible. For example, you can define your own data types, build custom functions, even write code from different programming languages without recompiling your database!
PostgreSQL tries to conform with the **SQL** standard where such conformance does not contradict traditional features or could lead to poor architectural decisions. Many of the features required by the **SQL** standard are supported, though sometimes with slightly differing syntax or function. Further moves towards conformance can be expected over time. As of the version 14 release in September 2021, **PostgreSQL** conforms to at least 170 of the 179 mandatory features for **SQL** :2016 Core conformance. As of this writing, no relational database meets full conformance with this standard.

*1 **ACID** In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps.[1] In the context of databases, a sequence of database operations that satisfies the ACID properties (which can be perceived as a single logical operation on the data) is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.
In 1983,[2] Andreas Reuter and Theo Härder coined the acronym ACID, building on earlier work by Jim Gray[3] who named atomicity, consistency, and durability, but not isolation, when characterizing the transaction concept. These four properties are the major guarantees of the transaction paradigm, which has influenced many aspects of development in database systems.

*2 **PostGIS** provides spatial objects for the PostgreSQL database, allowing storage and query of information about location and mapping.

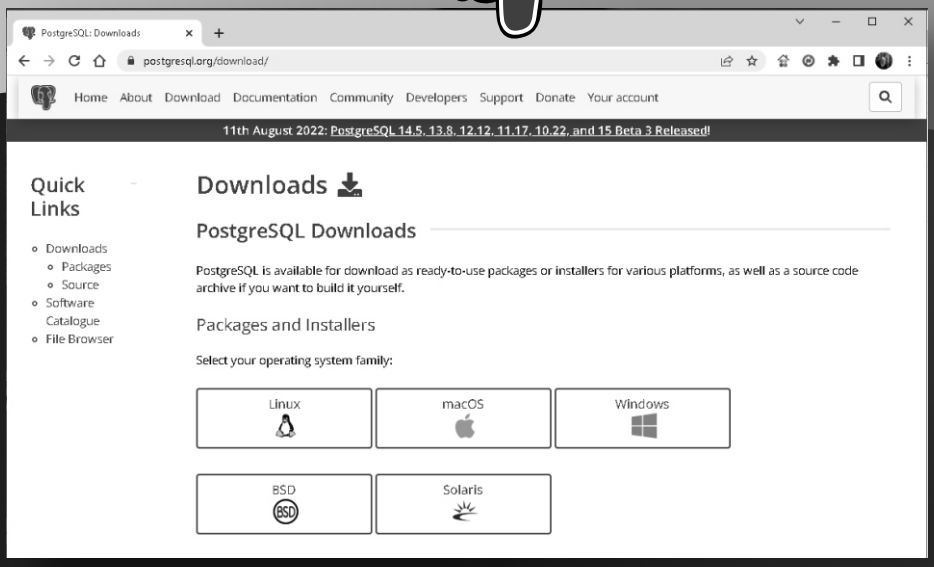
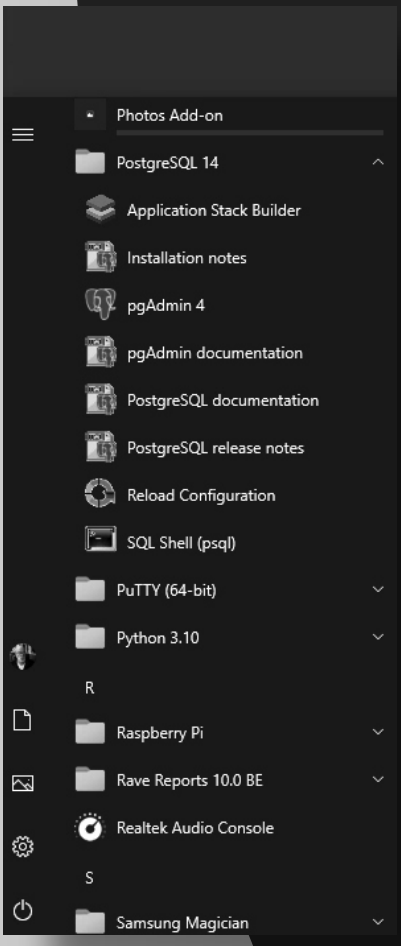


Figure 1: Make your choice: https://www.postgresql.org/



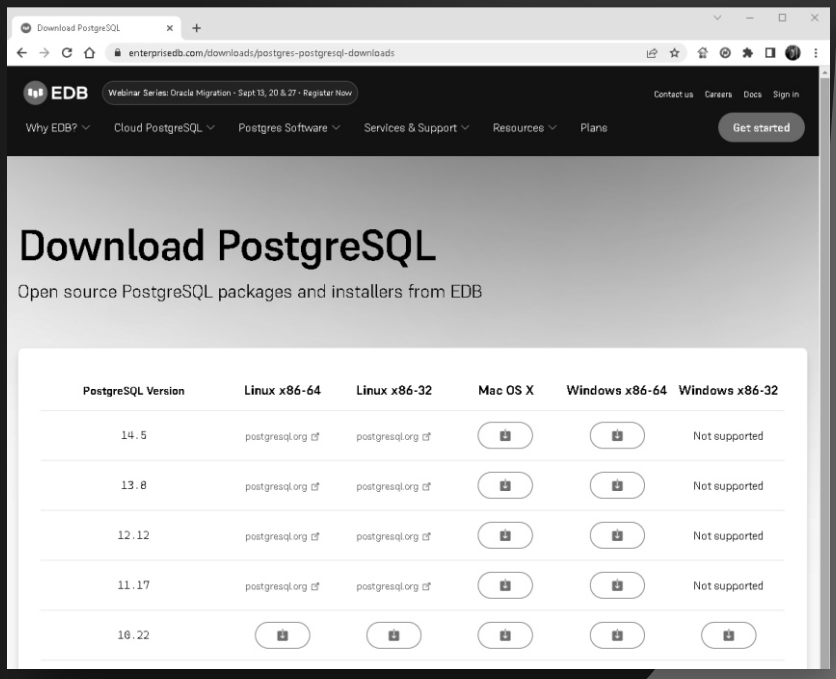


Figure 2: the versions.
We chose Win x86-64 , version 14.5

Figure 3: Fianlly in the list of win10

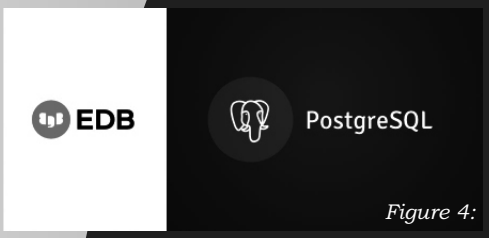


Figure 4:

Interactive installer by EDB
Download the installer certified by EDB for all supported PostgreSQL versions.

Note! This installer is hosted by EDB and not on the PostgreSQL community servers. If you have issues with the website it's hosted on, please contact webmaster@enterprisedb.com.

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

This installer can run in graphical or silent install modes.

Figure 5:

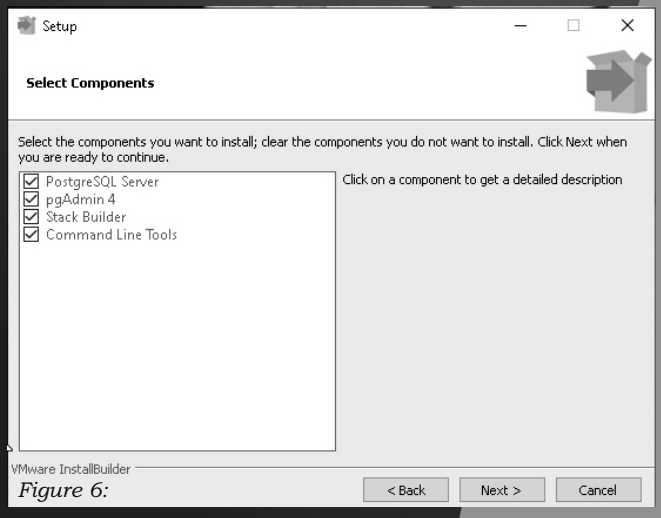


Figure 6:

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

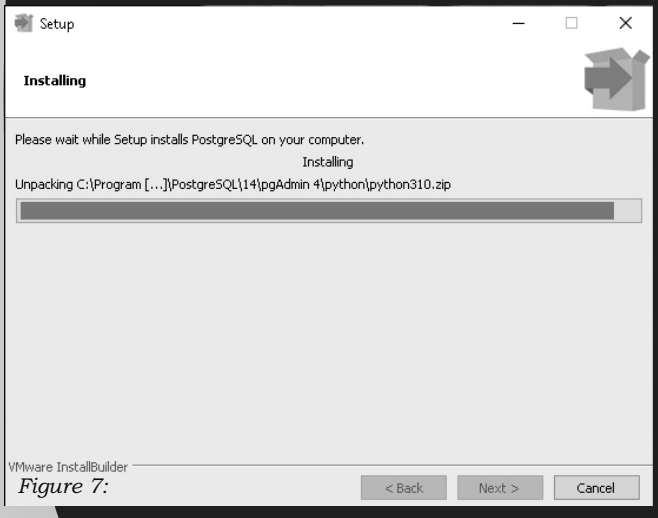


Figure 7:

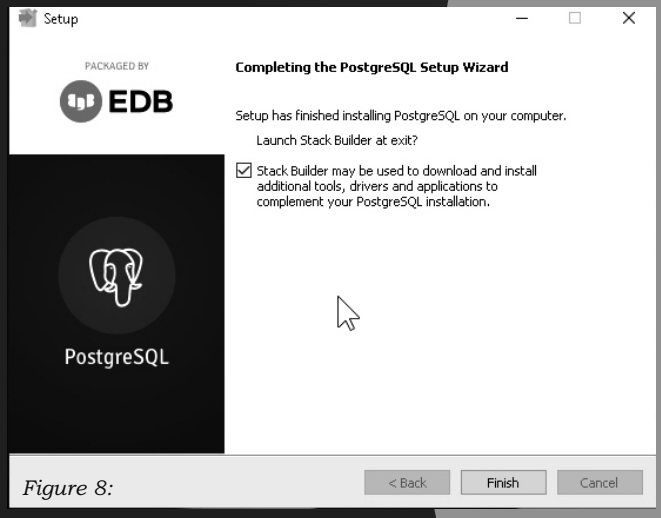


Figure 8:

Advanced users can also download a zip archive of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

PLATFORM SUPPORT

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions, for example, desktop releases of Windows:

PostgreSQL Version	64 Bit Windows Platforms	32 Bit Windows Platforms
14	2019, 2016	
13	2019, 2016	
12	2019, 2016, 2012 R2	
11	2019, 2016, 2012 R2	
10	2016, 2012 R2 & R1, 7, 8, 10	2008 R1, 7, 8, 10

Making the test project for all Data Bases for Lazarus

c:\lazarus\tools\lazdatadesktop\lazdatadesktop.lpi

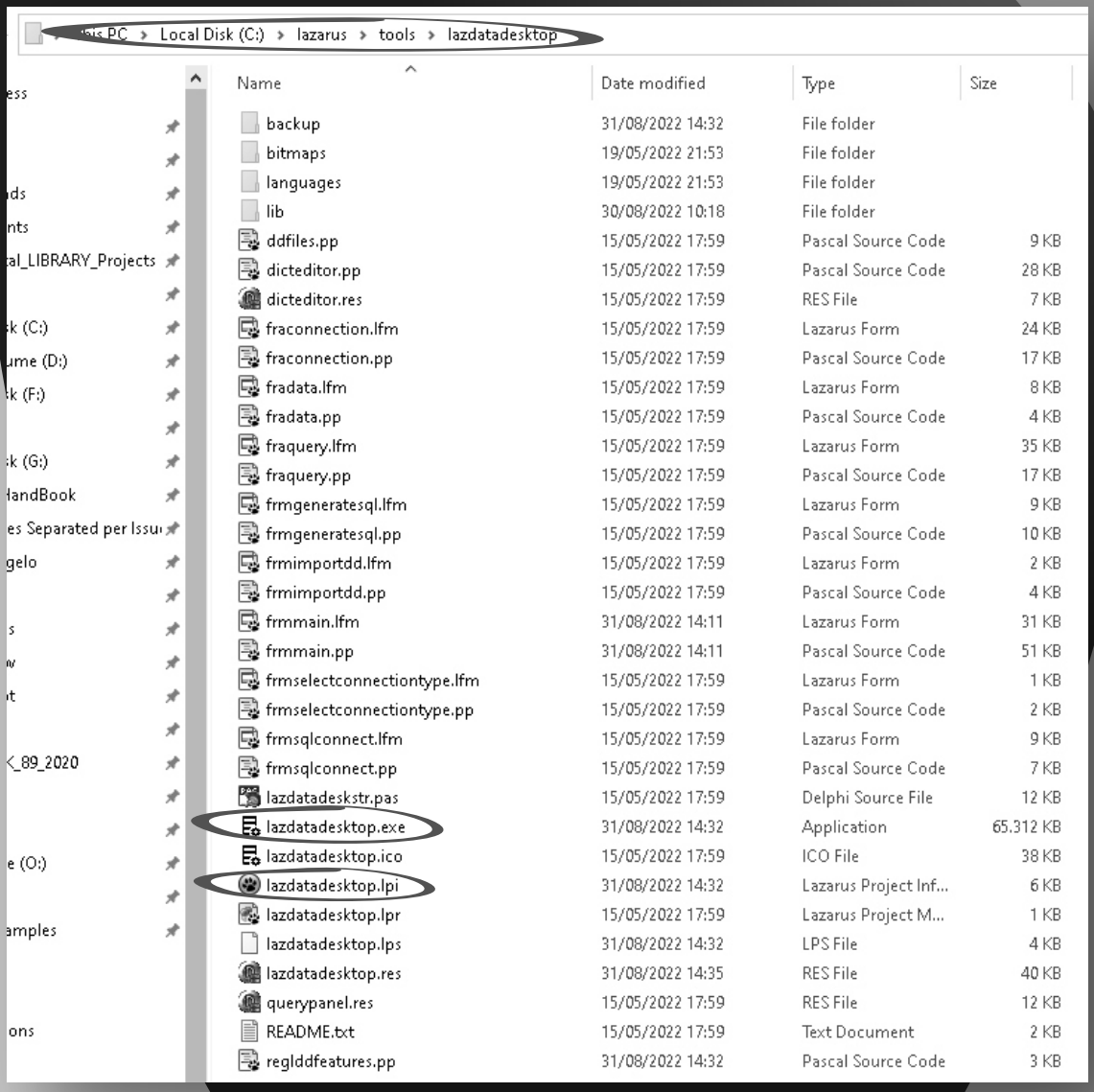


Figure 9: The location of the project under Lazarus here is the project file **lazdatadesktp.lpi** *After compiling you will find an* **.exe** *file*

Source Editor

frmmain *regldfeatures

```
{
*****************************************************************************
*                                                                           *
*   This source is free software; you can redistribute it and/or modify     *
*   it under the terms of the GNU General Public License as published by    *
*   the Free Software Foundation; either version 2 of the License, or       *
*   (at your option) any later version.                                     *
*                                                                           *
*   This code is distributed in the hope that it will be useful, but        *
*   WITHOUT ANY WARRANTY; without even the implied warranty of              *
*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU       *
*   General Public License for more details.                                *
*                                                                           *
*   A copy of the GNU General Public License is available on the World      *
*   Wide Web at <http://www.gnu.org/copyleft/gpl.html>. You can also        *
*   obtain it by writing to the Free Software Foundation,                   *
*   Inc., 51 Franklin Street - Fifth Floor, Boston, MA 02110-1335, USA.     *
*                                                                           *
*****************************************************************************
}


unit frmmain;

{$mode objfpc}{$H+}
```

Source Editor

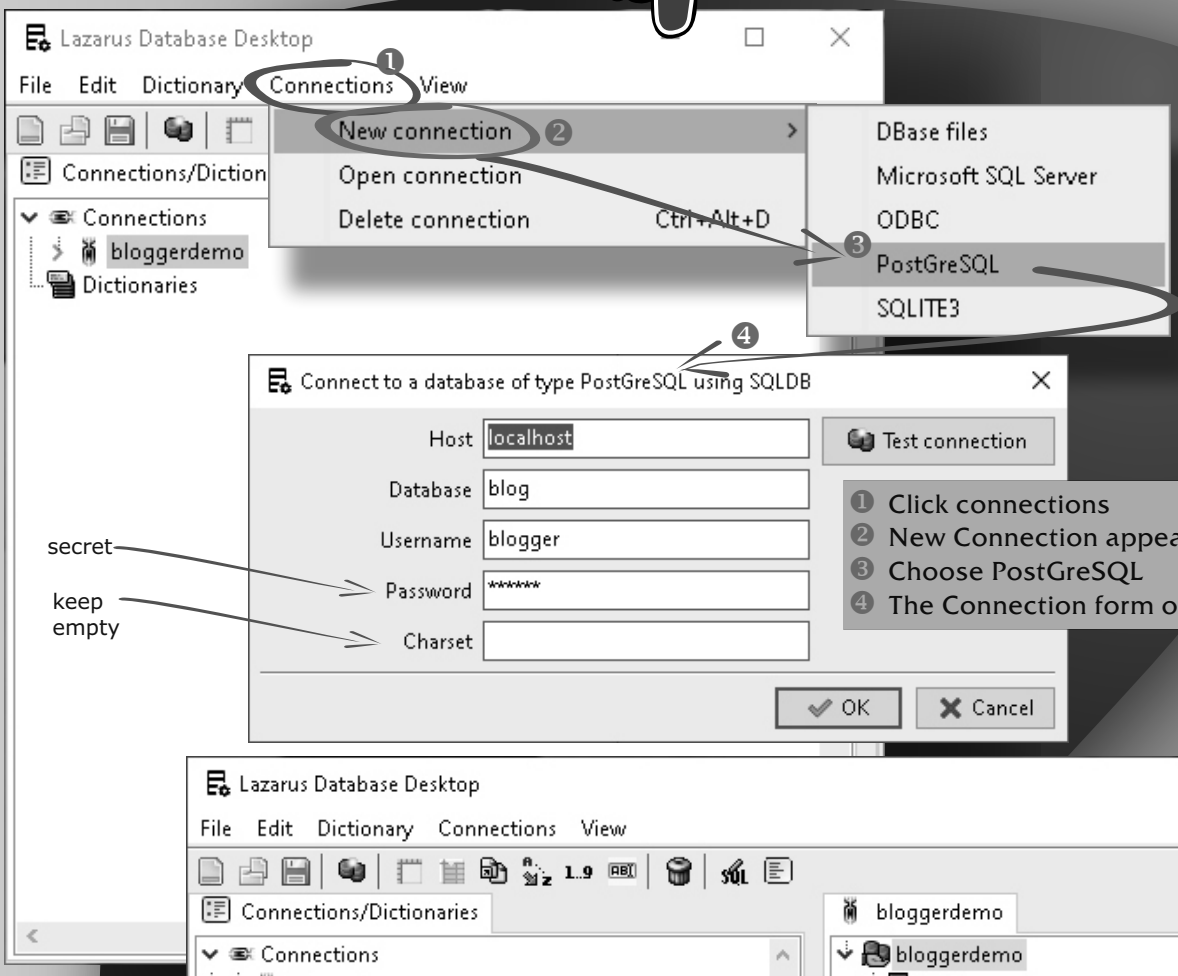
frmmain regldfeatures

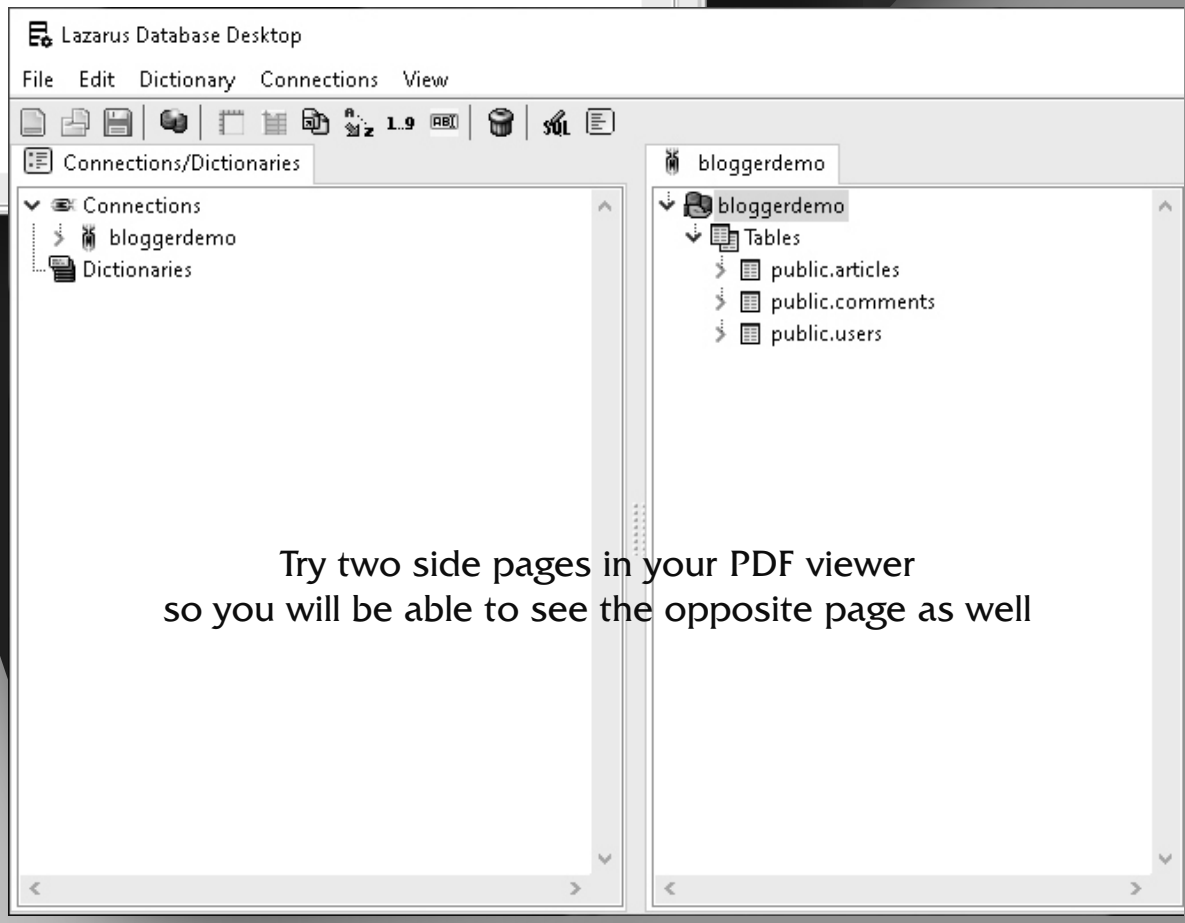
```
uses
  // Data dictionary support for datab
  fpdddbf,      // DBF
  {$ifndef win64}
  fpddfb,       // Firebird
  fpddmysql40, // MySQL 4.0
  fpddmysql41, // MySQL 4.1
  fpddmysql50, // MySQL 5.0
  fpddmysql51, // MySQL 5.1
  fpddmysql55, // MySQL 5.5
  {$ifdef HAVEMYSQL5657CONN}
  fpddmysql56, // MySQL 5.6
  fpddmysql57, // MySQL 5.7
  {$endif HAVEMYSQL5657CONN}
  fpddoracle,  // Oracle

  {$endif}
  fpddsqlite3, // SQLite 3
  fpddodbc,     // Any ODBC supported
  {$ifdef HAVEMSSQLCONN}
  fpddmssql,
  {$endif HAVEMSSQLCONN}
  // code generators
 {$IF FPC_FULLVERSION>=30200}
  fpcgfieldmap,
  fpcgtypesafedataset,
 {$ENDIF}
  fpddpq,       // PostgreSQL ----------
  fpcgSQLConst,
  fpcgdbcoll,
  fpcgCreateDBF,
  fpcgtiOPF,
  // data Export
  fpstdExports,
  fpxmlxsdexport,
  fpdbexport
  ;
```

```
implementation

procedure RegisterExportFormats;

begin
  RegisterXMLXSDExportFormat;
  RegisterStdFormats;
end;

procedure registerengines;

begin
 {$ifndef win64}
  RegisterFBDDEngine;
  RegisterMySQL40DDEngine;
  RegisterMySQL41DDEngine;
  RegisterMySQL50DDEngine;
  RegisterMySQL51DDEngine;
  RegisterMySQL55DDEngine;
 {$ifdef HAVEMYSQL5657CONN}
  RegisterMySQL56DDEngine;
  RegisterMySQL57DDEngine;
 {$endif}
  RegisterOracleDDEngine;

 {$endif}
  RegisterSQLite3DDEngine;
  RegisterODBCDDengine;
  RegisterPostgreSQLDDengine;    // PostgreSQL----
 {$IFDEF HAVEMSSQLCONN}
  RegisterMSSQLDDEngine;
 {$ENDIF}
end;
```

Figure 10: In the latest version of Lazarus 2.2.2., support for Postgres is not registered for the Win64 platform.
This can be fixed by moving the RegisterPostGreSQLDDEngine statement outside the IFDEF

Figure 11: In the latest version of Lazarus 2.2.2., the unit with support for Postgres is not included for the Win64 platform. This can be fixed by moving the fpddpq unit outside the IFDEF

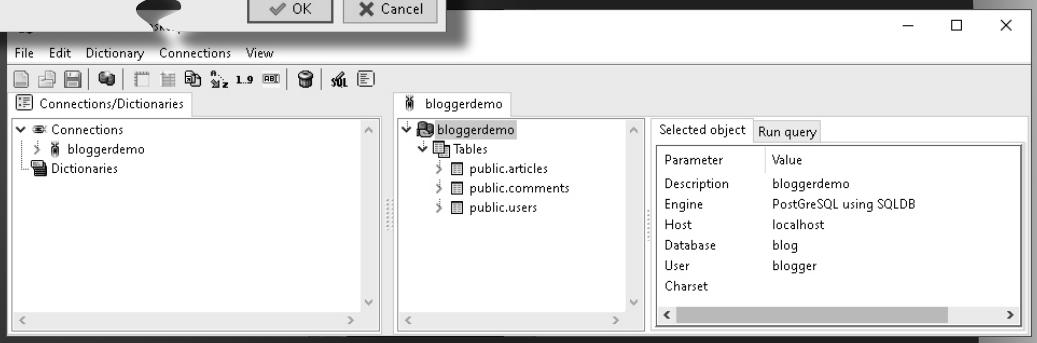


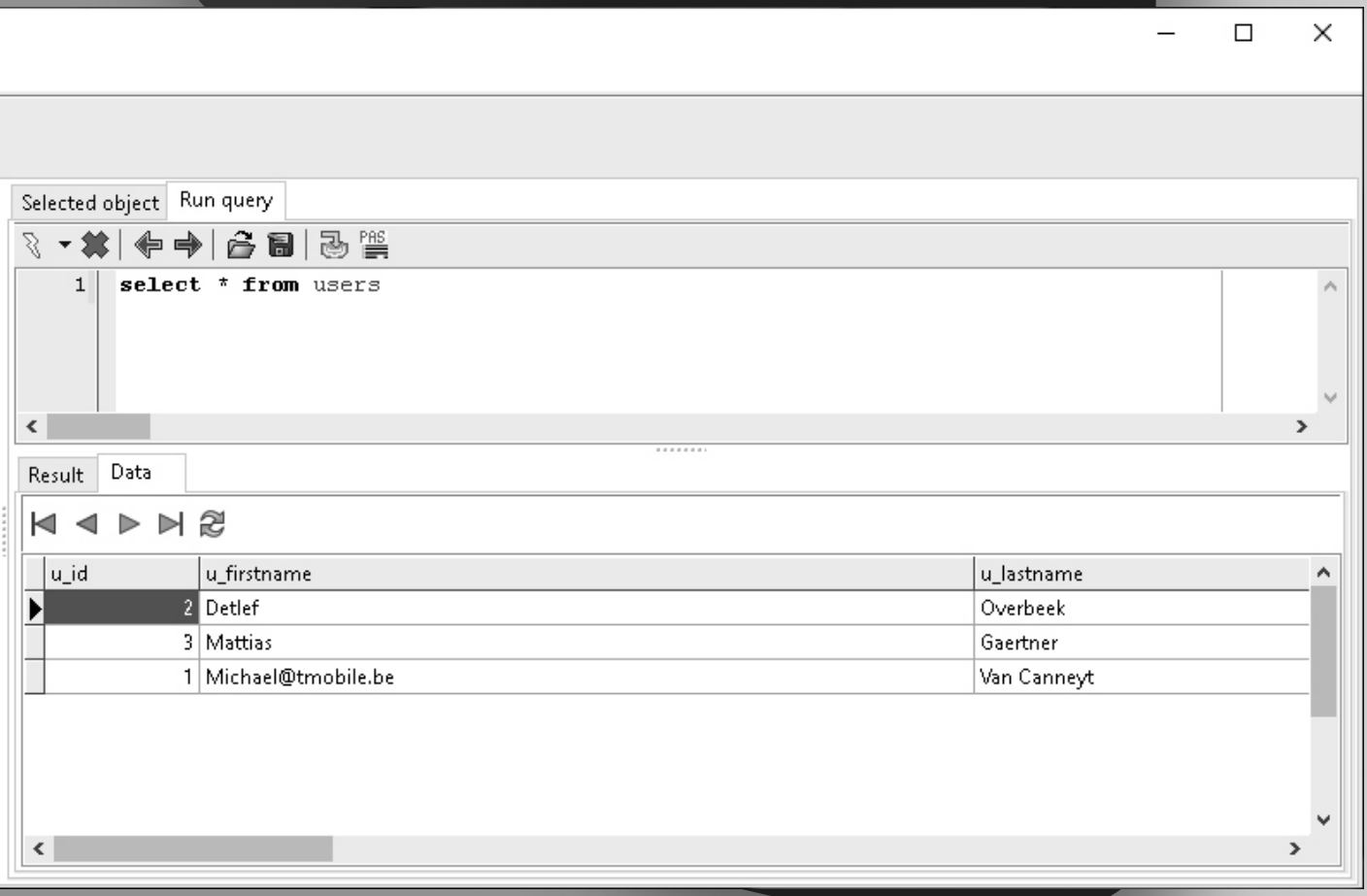


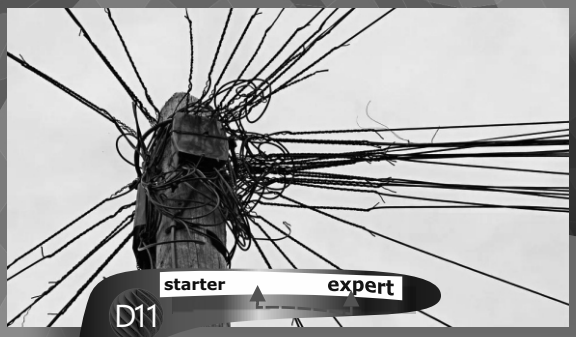
Try two side pages in your PDF viewer
so you will be able to see the opposite page as well

The **Lazarus Database Desktop** pops up.
Start with → **Connections** → **New connection** → **PostgreSQL**. You need to fill out the connection-form: in the first line enter localhost as the host. if you want to get into our predefined database, fill on the second line the database name: "blog" and "username" blogger. The password cannot be shown but fill in " secret".

Before saving the connection, you can can test the connection with the 'Test connection' button. The password cannot be shown but fill in secret - (*if you want to get into our predefined database*). **Character set** can be left empty. Now you'll see after clicking OK a new window where you can give in your **connection name.** The form opens, it can be smaller but you can widen it to show all items... Try the form options to get familiar with the possibilities. After this you can be sure your postgres database is working and will be available in the demo web application.







PROPHECY

The next release of **kbmMW Enterprise Edition** will include several new things and improvements. One of them is full WebSocket support.

WEBSOCKET EXPLAINED IN FEW WORDS

WebSocket is a "new" way (*standardized with an RFC in 2011*) to communicate between web browsers and application/web servers, which allows for a compact two way near real time communication, which is not based on a request/response scheme, but on that each side is listening for messages and pushing messages, in many ways similar to what the **WIB (***Wide Information Bus*) is doing.
Compared to the **WIB,** there is however no ability to subscribe for particular messages, so **WebSocket** is by default, not a **publish/subscribe** based duplex communication.

All major web browsers support **WebSocket** communication. To be able to use **WebSocket** on the client side, you will need to code in **JavaScript** or **WebAssembly.**
Fortunately it is pretty simple to code for **WebSocket** in **Javascript.**
WebSocket communication always starts out with regular HTTP communication.

A browser makes a regular **HTTP** request to a webserver, asking for the connection to be upgraded to **WebSocket** optionally with various additional wishes for the detailed communication.
The web server responds either by agreeing to one of the wishes requested by the browser, or by rejecting it all together, in which case no **WebSocket** communication is possible, at least not with the wishes requested by the browser client.

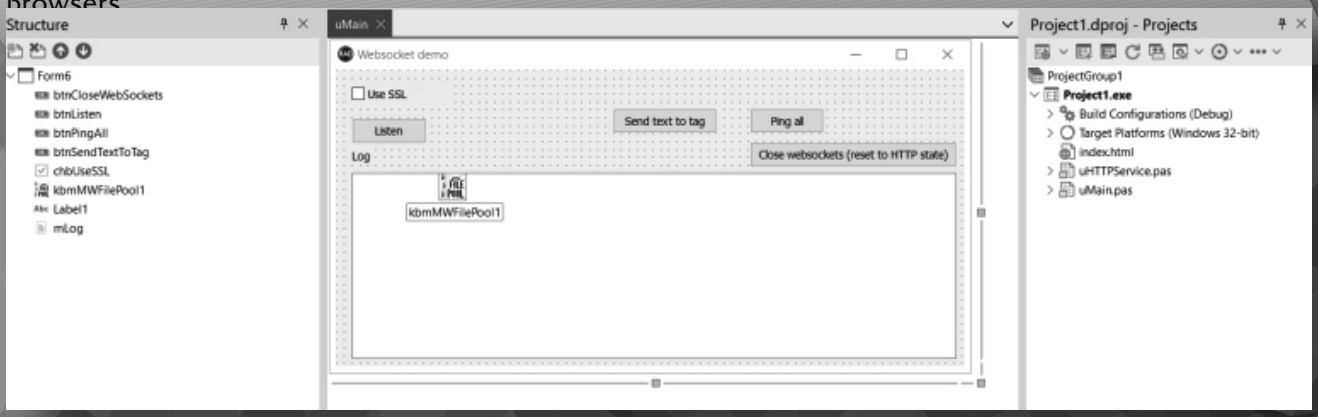
WEBSOCKETS IN KBMMW CONTEXT

kbmMW has for many years supported acting like a regular **WWW** server and as a **REST** endpoint. This is a requirement for being able to do **WebSockets.**
In addition **kbmMW** has usually been using the standard **Indy** components for transport, although other ways has been made possible over the years.

However kbmMW has for a couple of years included some extremely fast and efficient socket components, which moves **kbmMW** to the top of the performance board when doing **HTTP** or **REST** operations.
Obviously **WebSockets** continue to build on that technology, rather than on Indy which means you will get a very fast **WebSockets** able web server if you use **kbmMW.** Check this performance blog to notice that **kbmMW** actually beats all other competitors in most comparisons including all the popular variants used by non **Delphi** developers.

A SAMPLE KBMMW BASED WEBSOCKET SERVER

We make a simple **Delphi** project. It will contain a form, and an **HTTP** service unit along with some **HTML/Javascript** it can serve to a client. The server will provide a rudimentary chat client for browsers.



SAMPLE OF THE FORM

As you may notice, the sample includes the option to let the **Webserver** offer access via **SSL** depending on if the Use **SSL** checkbox is checked or not.

The Listen button contains this code:

```
procedure TForm6.btnListenClick(Sender: TObject);
begin
  if FServer.Active then
  begin
    FServer.Active:=false;
    btnListen.Caption:='Listen';
  end
  else
  begin
    FTransport.Host:='0.0.0.0';
    if chbUseSSL.Checked then
    begin
      FTransport.UseSSL:=true;
      FTransport.Port:=443;
      FTransport.SetSSLCertificateFromFile('.\domain.crt');
      FTransport.SetSSLPrivateKeyFromFile('.\domain.key');
    end
    else
    begin
      FTransport.UseSSL:=false;
      FTransport.Port:=80;
    end;
    FServer.Active:=true;
    btnListen.Caption:='Dont listen';
  end;
end;
```

Really simple code. Even the **SSL** part is extremely simple to setup in code.

Ok… there is a little bit more code to write. The following could have been set in properties on a **WebSocket** component put on the main form, but in this case I show it in code:

```
constructor TForm6.Create(Owner:TComponent);
begin
  inherited Create(Owner);
  FServer:=TkbmMWServer.Create(nil);
  FTransport:=TkbmMWWebSocketServerTransport.Create(nil);
  FTransport.Server:=FServer;
  FTransport.OnWebSocketData:=DoOnWebSocketData;
  FTransport.OnWebSocketOpen:=DoOnWebSocketOpen;
  FTransport.OnWebSocketClose:=DoOnWebSocketClose;
  FTransport.OnWebSocketPong:=DoOnWebSocketPong;
  FServer.AutoRegisterServices;
  Log.OutputToStrings(mLog.Lines);
end;
```

All that this code does is to create a **kbmMW** Server instance **(TkbmMWServer)** and a **TkbmMWWebSocketServerTransport** instance. Those are components and could have been put on the form instead of instantiated in code.
I add some event handlers for when the **WebSocket** protocol receives data, when a client connects and disconnects and when a ping response is returned by a client as a pong.

Further I link the transport to the server component, and ask **kbmMW** to do its magic about discovering services that can be made available.

Finally I instruct the **kbmMW** logger to output all its log live to the **TMemo**.

And similarly there needs to be a little bit clean up code in the forms destructor:

```
destructor TForm6.Destroy;
begin
  FServer.Active:=false;
  FTransport.Free;
  FServer.Free;
  inherited Destroy;
end;
```

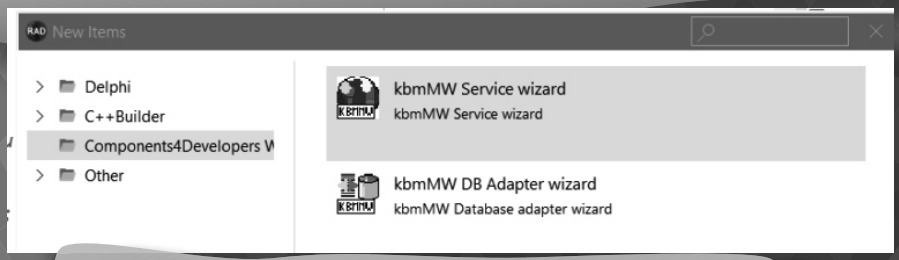
The following are event handlers for the **WebSocket** communication:

```
procedure TForm6.DoOnWebSocketData(const AConnection:IkbmMWWebSocketConnection; const
AData:TValue);
var
  s:string;
begin
  s:=AData.ToString;
  Log.Info('Got data:'+s);
  AConnection.BroadcastText(s);
end;
```

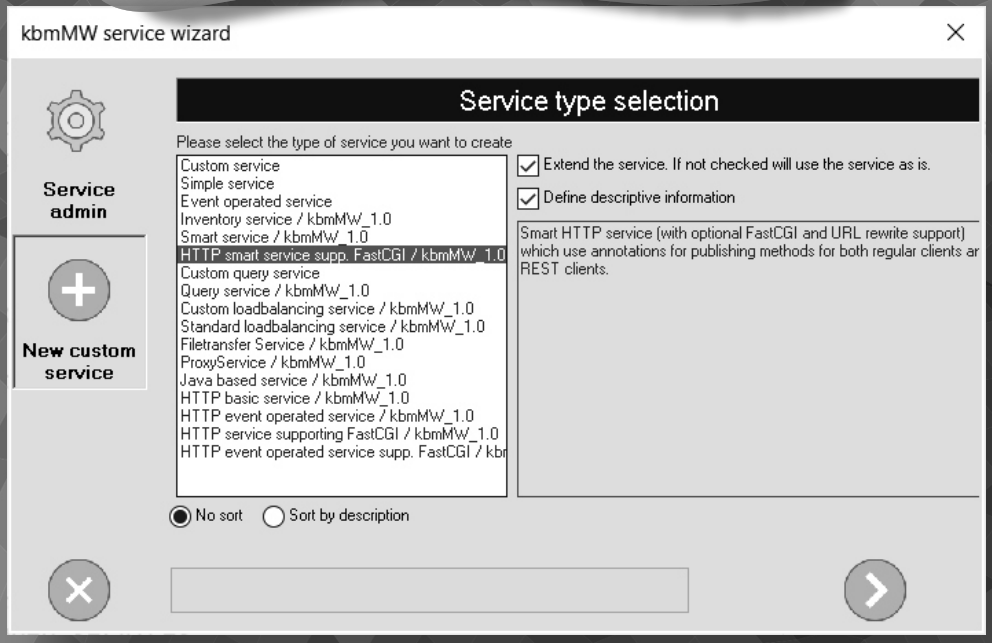
```
procedure TForm6.DoOnWebSocketPong(const AConnection:IkbmMWWebSocketConnection; const AData:TValue);
begin
  Log.Info('Got pong from:'+ AConnection.GetPeerAddr+':'+inttostr(AConnection.GetPeerPort)+
      ' : '+AData.ToString);
end;

procedure TForm6.DoOnWebSocketOpen(const AConnection:IkbmMWWebSocketConnection);
begin
  AConnection.TagString:='TAG1';
  Log.Info('Opening websocket connection: '+ AConnection.GetPeerAddr+':'+inttostr(AConnection.GetPeerPort)+
      ' now tagged as '+AConnection.TagString);
end;

procedure TForm6.DoOnWebSocketClose(const AConnection:IkbmMWWebSocketConnection; const ACode:word;
const AReason:string);
begin
  Log.Info('Closing websocket connection: '+ Connection.GetPeerAddr+':'+inttostr(AConnection.GetPeerPort)+
      ' Code:'+inttostr(ACode)+' Reason:'+AReason);
end;
```

The only one really required of these event handlers is the `OnWebSocketData`. In our case it simply broadcasts the data back to all **WebSocket** connected clients.
We still need a little bit more code.. namely something to verify and handle the request to upgrade from the **HTTP** protocol to the **WebSocket** protocol.

This is done in the **HTTP** service that is the 2nd **Delphi** unit added to the project.
The service was created by using the **kbmMW** service wizard:



The **kbmMW** Service wizard entry point in RAD Studio
After starting it, choose **HTTP Smart Service**



And then simply page right until you reach the end of the wizard and let the wizard generate the unit.
That results in this:

```
unit uHTTPService;

// ============================================================================
// kbmMW - An advanced and extendable middleware framework.
// by Components4Developers (http://www.components4developers.com)
//
// Service generated by kbmMW service wizard.
//
// INSTRUCTIONS FOR REGISTRATION/USAGE
// -----------------------------------
// Please update the uses clause of the datamodule/form the TkbmMWServer is placed on by adding
services unit name
// to it. Eg.
//
//    uses ...,kbmMWServer,YourServiceUnitName;
//
// Somewhere in your application, make sure to register the serviceclass to the TkbmMWServer instance.
// This can be done by registering the traditional way, or by using auto registration.
//
// Traditional registration
// ------------------------
// var
//   sd:TkbmMWCustomServiceDefinition;
// ..
//   sd:=kbmMWServer1.RegisterService(yourserviceclassname,false);
//
// Set the last parameter to true if this is the default service.
//
//
// Auto registration
// -----------------
// Make sure that your service class is tagged with the [kbmMW_Service] attribute.
// Then auto register all tagged services:
// ..
//   kbmMWServer1.AutoRegisterServices;
//
// ------------------------------------------------
//
// SPECIFIC HTTP SERVICE REGISTRATION INSTRUCTIONS
// ------------------------------------------------
// Cast the returned service definition object (sd) to a TkbmMWHTTPServiceDefinition. eg:
//
// var
//   httpsd:TkbmMWHTTPServiceDefinition;
// ..
//   httpsd:=TkbmMWHTTPServiceDefinition(sd)
//   httpsd.RootPath[mwhfcHTML]:='.';
//   httpsd.RootPath[mwhfcImage]:='.';
//   httpsd.RootPath[mwhfcJavascript]:='.';
//   httpsd.RootPath[mwhfcStyleSheet]:='.';
//   httpsd.RootPath[mwhfcOther]:='.';
// ------------------------------------------------
end.
```

```
{$I kbmMW.inc}

interface

uses  SysUtils,
{$ifdef LEVEL6}
 Variants,
{$else}
 Forms,
{$endif}
 Classes,
 kbmMWSecurity,
 kbmMWServer,
 kbmMWServiceUtils,
 kbmMWHTTPStdTransStream,
 kbmMWGlobal,
 kbmMWCustomHTTPSmartService
 ,kbmMWRTTI
 ,kbmMWSmartServiceUtils;

type

 [kbmMW_Service('flags:[listed]')]
 [kbmMW_Rest('path:/')]
 // Access to the service can be limited using the [kbmMW_Auth..] attribute.
 // [kbmMW_Auth('role:[SomeRole,SomeOtherRole], grant:true')]

 TWebSocketService = class(TkbmMWCustomHTTPSmartService)
  function kbmMWCustomHTTPSmartServiceUpgrade(Sender: TObject;
   const ARequestHelper, AResponseHelper: TkbmMWHTTPTransportStreamHelper;
   const ARequestedProtocols: string;
   var AAcceptedProtocol: string): Boolean;
 private  { Private declarations }
 protected  { Protected declarations }
 public
   { Public declarations }
   // HelloWorld function callable from both a regular client, due to the optional [kbmMW_Method] attribute,
   // and from a REST client due to the optional [kbmMW_Rest] attribute.
   // The access path to the function from a REST client (like a browser)+  is in this case relative to the services path.
   // In this example: http://...//helloworld
   // Access to the function can be limited using the [kbmMW_Auth..] attribute.
   // [kbmMW_Auth('role:[SomeRole,SomeOtherRole], grant:true')]
   [kbmMW_Rest('method:get, path:helloworld')]
   [kbmMW_Method]
   function HelloWorld:string;
 end;

implementation

uses kbmMWExceptions, uMain;

{$R *.dfm}


// Service definitions.
//--------------------

function TWebSocketService.HelloWorld:string;
begin
   Result:='Hello world';
end;

function TWebSocketService.kbmMWCustomHTTPSmartServiceUpgrade(Sender: TObject;
 const ARequestHelper, AResponseHelper: TkbmMWHTTPTransportStreamHelper;
 const ARequestedProtocols: string; var AAcceptedProtocol: string): Boolean;
begin
   AAcceptedProtocol:='websocket';
   Result:=true;
end;

initialization
 TkbmMWRTTI.EnableRTTI(TWebSocketService);
end.
```

It contains a lengthy comment at start, explaining a little bit about what the service is supposed to do and how to register it with **kbmMW.** But the interesting part is the function HelloWorld and the **OnUpgrade** event handler.
The function **HelloWorld** is just included as standard to show how to create a function that can be called from a browser, and which returns some data or **HTML** or other stuff to be presented in the client. It is possible to tell which mimetype is returned, so the browser will display the data correctly. In this case we simply return html/text since no specific mimetype is given.
You call this function from the browser by requesting an **URL** like this (*provided SSL is not enabled, otherwise replace http with https*):

http://localhost/helloworld
The browser will then display the text returned… Hello world. You could return files and many other things here, but you can also just leave that up to **kbmMW,** who will try to locate a file to send to the browser in case no services are implemented to handle the request (*URL path*).
The even handler **OnUpgrade** is called the moment a client asks for upgrading the connection to some other sort of connection. The client will provide a string over protocols that it will accept.
In this case you will receive this list as a comma separated string.

The event handlers job, is then to return back to the client, one of the protocols that the server will accept. In the case of **WebSockets,** it is the **websocket** protocol written exactly like that in lowercase. As long as the client requested the ability to upgrade to **WebSocket,** you will now have established a **WebSocket** connection between the client and the **kbmMW** server.
If the client asked for something else and did not include websocket as one of its protocol options, the client will disconnect with an error.
Now that is all there is needed as far as server side code. Let us take a look at the client side.

CLIENT SIDE CODE

The client side *(the browser)* will be fed with the needed **HTML** and **Javascript** from the server to allow implementation of a chat client that is able to open a **WebSocket** connection to the server.

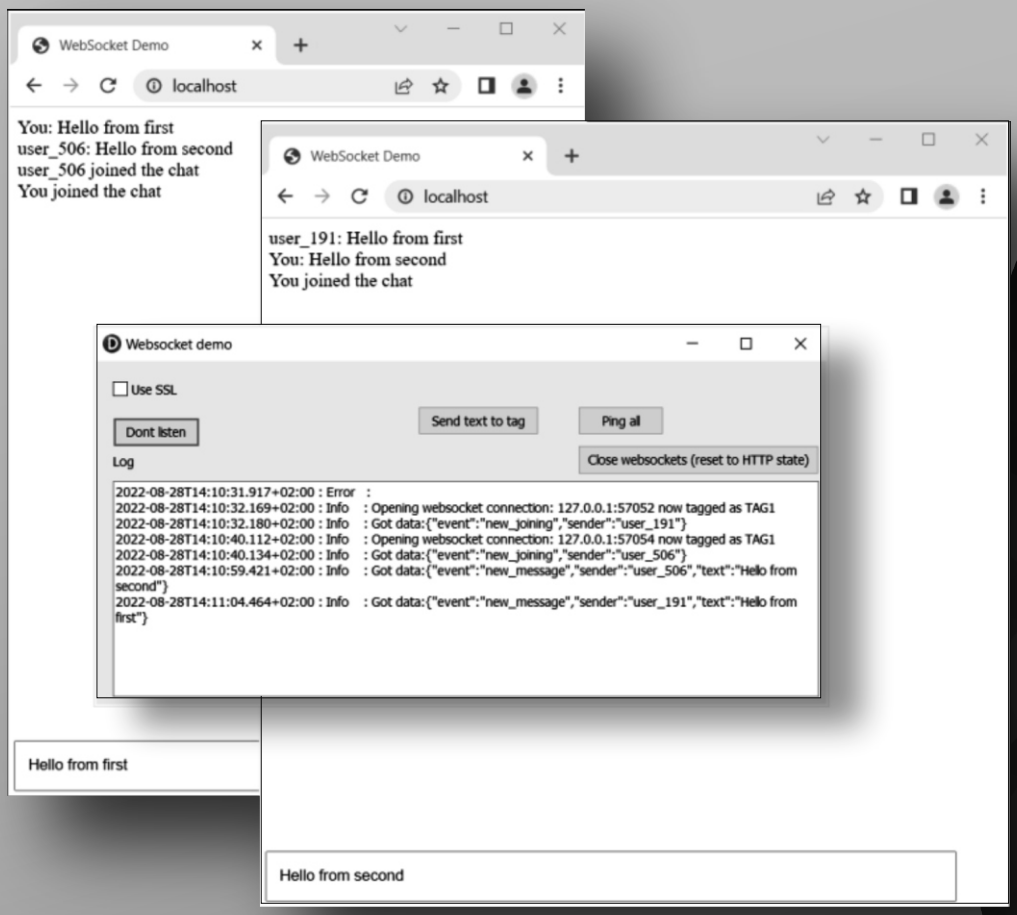
This is done, in this sample, by providing the index.html file to the client when it connects at first. The **index.html** file is relatively simple. It contains a little bit of **HTML,** primarily some divisions for text entry and the chat log, and then it contains a little bit of styling **(CSS)** and finally it contains a little bit of **Javascript** code to handle the **WebSocket** communication.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>WebSocket Demo</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        #chatFormContainer {
            text-align: center;
            position: fixed;
            bottom: 5px;
            left: 5px;
            right: 5px;
        }

        #chatMessage {
            display: inline-block;
            float: left;
            width: 90%;
            height: 20px;
            padding: 10px;
        }
    </style>
</head>
```

```
<body>
    <div id="chatLog">

    </div>
    <div id="chatFormContainer">
        <form id="chatForm">
            <input id="chatMessage" placeholder="Type  message and press enter..."
                   size="1">
        </form>
    </div>
    <script>
        var username = `user_${getRandomNumber()}`
        var channelId = 1;
        var socket = new WebSocket(location.origin.replace(/^http/, 'ws') +
                                   '/v3/${channelId}?notify_self');

        var chatLog = document.getElementById('chatLog');
        var chatForm = document.getElementById('chatForm');
        chatForm.addEventListener("submit", sendMessage);

        socket.onopen = function() {
            console.log(`Websocket connected`);
            socket.send(JSON.stringify({
                event: 'new_joining',
                sender: username,
            }));
        }
        socket.onmessage = function(message) {
            var payload = JSON.parse(message.data);
            console.log(payload);

            if (payload.sender == username) {
                payload.sender = "You";
            }

            if (payload.event == "new_message") {
                //Handle new message
                chatLog.insertAdjacentHTML('afterend',
                                `<div> ${payload.sender}: ${payload.text} <div>`);
            }
            else if (payload.event == 'new_joining') {
                //Handle new joining
                chatLog.insertAdjacentHTML('afterend',
                                `<div> ${payload.sender} joined the chat<div>`);
            }
        }

        function getRandomNumber() {
            return Math.floor(Math.random() * 1000);
        }

        function sendMessage(e) {
            e.preventDefault();

            var message = document.getElementById('chatMessage').value;

            socket.send(JSON.stringify({
                event: 'new_message',
                sender: username,
                text: message
            }));
        }
    </script>
</body>
</html>
```

First the **HTML** is rendered (*the two div's one of them holding a simple form*).
Secondly the script is run.
The script attempts to establish a **WebSocket** connection to the connected server, sets up some event handlers for data and states coming from the **WebSocket,** and sets up a couple of event handlers on the form's input component so it will send the input via the **WebSocket** connection the moment Enter is pressed (submit).

The result:



CHAT WITH TWO CLIENTS CONNECTED
We see a chat going on with two clients connected and the logs on the server.

kbmMW's WebSocket transport can thus be used as a replacement for any **HTTP** transport, while also being able to understand **WebSocket** communication.
I will show in a different blogpost, the various methods there exists for sending and receiving textual and binary data.
If you like what you have read, please make sure to share links to this with others on **Facebook** or other social medias!

DONATE FOR UKRAINE AND GET A FREE LICENSE AT:

https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

(Just click)

COMPONENTS 4 DEVELOPERS

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

DONATE FOR UKRAINE AND GET A FREE LICENSE AT:

https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

(Just click)

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.18.00 RELEASED!

- **RAD Studio XE5 to 11 Alexandria supported**
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support
- Native high performance 100% developer defined application server
- Full support for centralized and distributed load balancing and failover
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multithread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronouncable password generators.
- High performance LZ4 and Jpeg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, JSON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

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

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

- **NEW: FULL WEBSOCKET SUPPORT.**
  The next release of kbmMW Enterprise Edition will include several new things and improvements.
  One of them is full WebSocket support.
- New I18N context sensitive internationalization framework to make your applications multilingual.
- New ORM LINQ support for Delete and Update.
- Comments support in YAML.
- New StreamSec TLS v4 support (by StreamSec)
- Many other feature improvements and fixes.

Please visit
http://www.components4developers.com
for more information about kbmMW

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

COMPONENTS 4 DEVELOPERS

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