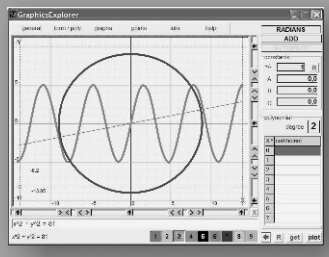
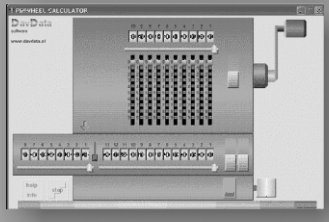
FOR DELPHI, LAZARUS, AND PASCAL
RELATED LANGUAGES / WEB APPS,
INTERNET, ANDROID, IOS, MAC,
WINDOWS & LINUX

BLAISE PASCAL MAGAZINE 72

DAVID DIRKSE





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

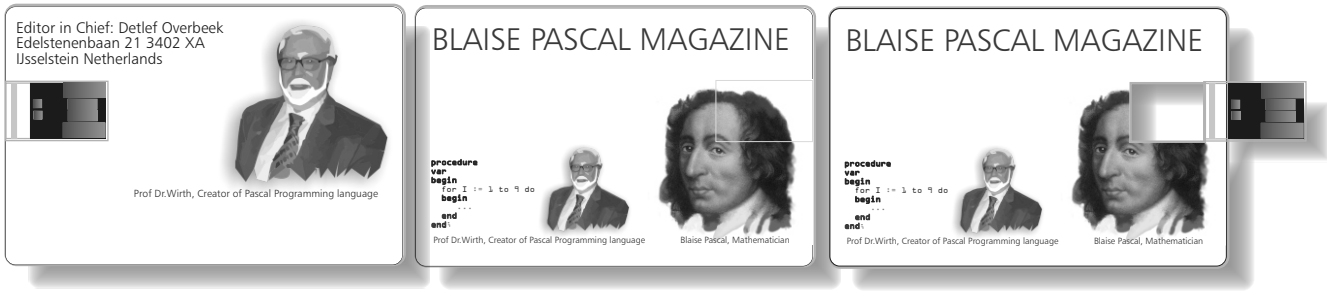
GRAPHICS COMPUTER MATH & GAMES IN PASCAL

POCKET EDITION

Printed in full color.
A fully indexed PDF book
is included + 52 projects

CREDITCARD LIBRARY STICK 16 GB

All issues on the USB stick
complete searchable 3600 pages
-fully indexed



COMBINATION: 3 FOR 1

BOOK INCLUDING THE LIBRARY STICK EXCL. SHIPPING
INCLUDING 1YEAR DOWNLOAD FOR FREE
GET THE BOOK INCLUDING THE NEWEST LIBRARY STICK
INCLUDING 1 YEAR DOWNLOAD OF BLAISE PASCAL MAGAZINE

€ 100

https://www.blaisepascal.eu/subscribers/UK/UK_CD_DVD_USB_Department.html

BLAISE PASCAL MAGAZINE 72
DELPHI, LAZARUS, SMART MOBILE STUDIO,
AND PASCAL RELATED LANGUAGES
FOR ANDROID, IOS, MAC, WINDOWS & LINUX


Blaise Pascal

CONTENT

Articles

ADVERTISERS

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

Left: Niklaus Wirth

Editor - in - chief
Detlef D. Overbeek, Netherlands Tel.: +31 (0)30 890.66.44 / Mobile: +31 (0)6 21.23.62.68
News and Press Releases email only to editor@blaisepascal.eu

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

Correctors
Howard Page-Clark, Peter Bijlsma

Subscriptions (2017 prices)

	Internat. excl. VAT	Internat. incl. VAT	Including Shipment
Printed Issue ±80 pages	€ 235	€ 266,50	€ 85,00
Electronic Download Issue 80 pages	€ 50	€ 60,50	———
Printed Issue Inside Holland(Netherlands) ±80 pages	———	€ 180	€ 30,00

Member and donator of WIKIPEDIA

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

From the editor

I'm always pleased when I can improve our magazine in response to reader feedback. I asked about colour usage, and people often tell me we should use less colour – not to go black and white, but to reduce the coloration. You can see in the present issue how I have tried to respond to this suggestion... Let me know what you think of it.

Another request from several readers is for us to include examples of applications that are easy for a beginner to understand, and simple to implement. This is more difficult than you might think, requiring extra effort to simplify ideas and techniques. Of course there are always topics we cover that are complex by their very nature, and not suited to a beginner. However we will endeavour to make articles as simple as possible (without dumbing down) and to include full explanations where required. You may recall my article about a client/server application where I made a start on this journey into simplicity, using various examples to aid understanding.

It is a long story and will need further pages in a sequel before it is fully clear to a beginner. The example of how to create a software clock is also a good one for beginners to start with.
Providing a few very simple applications designed to stretch beginners' understanding is how we shall proceed, apps that can be created using either Delphi or Lazarus. Not only creating the basic app, but later adding further features that enhance it, such as adding persistence so settings are remembered, or developing an installer for the app. It is good for me too, because I have to learn and learn again, revisiting topics I once knew but now only half-remember.

The cost of posting a printed magazine copy is an ever-rising proportion of the overall subscription cost (currently about €8.50 per issue to ship to any destination). To reduce the postage cost we have reduced the page size and weight. So this issue is smaller than the A4 format we used formerly, and uses a slightly thinner paper, bringing the weight down below the 180 gram postage band, into a somewhat lower band. It seems absurd that the cost of shipping an issue has crept up almost to match the cost of printing it! The postal company increase their prices whether or not it reduces their overall shipment statistics. They don't care that they will have a smaller throughput because their prices are so high.

This is the first issue in which I have tried to offer the sources for all apps in three versions: for Lazarus, for Delphi 7 and for Delphi Tokyo. That can be problematic (as you will realise), but in most cases I have been able to do this well. In fact it is often easier to write code in a way that minimises differences between the compilers. Naturally, now and then you encounter significant differences or omissions in the controls that come with those three IDEs. In the next issue I'll explain how to get around such difficulties.

We are starting to look ahead to September 2018 events, and there is one event of great interest: the Lazarus Professional Conference.
This will take place over three days in Cologne/Bonn from Wednesday 19th September and there are exciting plans for showcasing new things.

The first two days will be devoted to the commercial advantages of Lazarus Professional, when we will dive deep into the new world and capabilities of Lazarus Professional. We will show you the most amazing things about the program and its future. You can read the feature list on **https://www.blaisepascalmagazine.eu/events/** or page 40/41 in this issue.

The third day of the Conference is designed as a Community Day with a very special programme including the release of the new Lazarus Handbook, and some very good news for you as well: any Lazarus users who are not yet subscribers to Blaise Pascal Magazine will be offered a free six month subscription to discover whether our magazine is the one for them.
There will be a lottery with a top prize of a Lazarus Professional licence, and other prizes including copies of the Lazarus Handbook.

Full details of the event will be published as soon as there are new items - we are still developing the program. Consult our website for the latest information:
https://www.blaisepascalmagazine.eu/events/

INRODUCTION TO THE BEGINNINGS OF UNDERSTANDING AN ARTIFICIAL NEURAL NETWORK

This article is based on a first implementation of the famous **PERCEPTRON**, using a 20x20 grid (*just like the original Mark 1 Perceptron had*). A perceptron is like the mother of an **A**rtificial **N**eural **N**etwork **(ANN).** before going to work with the tutorial, I* will explain the meaning of this and show you eventually a working model that will allow you to understand what is happening in this algorithm.

WHAT IS A PERCEPTRON?

In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers (*functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not*).
It is a type of **linear classifier**, i.e. *a classification algorithm* that makes its predictions based on a linear predictor function, combining a set of weights with the feature **vector.** In computer science this means a one-dimensional array.

Arrays allow us to refer to a series of variables by the same name and to use a number (an index) to call out individual elements in that series. Arrays have both upper and lower bounds and the elements of the array are contiguous within those bounds. Elements of the array are values that are all of the same type (string, integer, record, custom object). In Delphi, there are two types of arrays:
a fixed-size array which always remains the same size - a static array - and
a dynamic array whose size can change at runtime.

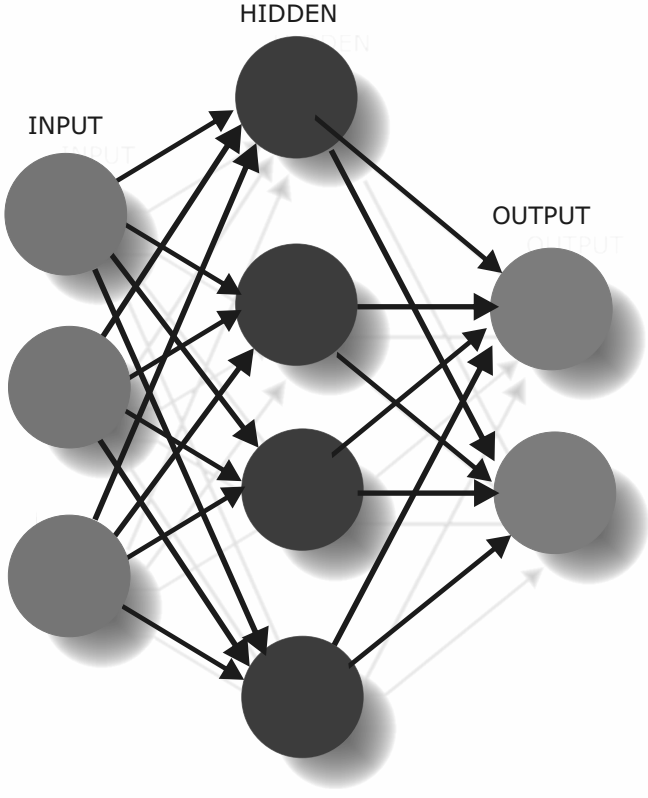
These modern language features were not yet invented at the time this all took place.

HISTORY OF THE PERCEPTRON

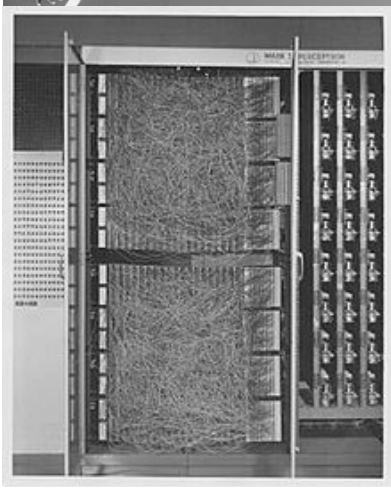
The perceptron algorithm was invented in 1957 at the **Cornell Aeronautical Laboratory** by **Frank Rosenblatt**, funded by the **United States Office of Naval Research**. The perceptron was intended to **be a machine**, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons".

Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

In a 1958 press conference organized by the **US Navy,** Rosenblatt made statements about the **perceptron** that caused a heated controversy among the fledgling AI community; based on Rosenblatt's statements, **The New York Times** reported the perceptron to be
"the embryo of an electronic computer that the Navy expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence".
Although the **perceptron** initially seemed promising, it was quickly proved that **perceptrons** could not be trained to recognise many classes of patterns.



This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with **two or more layers** (*also called a multilayer perceptron*) had far greater processing power than perceptrons **with one layer** (*also called a single layer perceptron*).
Single layer perceptrons are only capable of learning linearly separable patterns.



The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights.

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a **variable resistor** or **rheostat.** The word rheostat states (to set, to cause to stand") meaning "setter, regulating device", which is a two-terminal variable resistor. The term "rheostat" is becoming obsolete, with the general term "potentiometer" replacing it.

However, this is not true, as both **Minsky** and **Papert** already knew that **multi-layer perceptrons** were capable of producing an **`XOR function`**.

The Xor keyword is used in two different ways:
1. To perform a logical (**`or`** and **`Xor`**) or boolean 'Exclusive-or' of two logical values. If they are different, then the result is true.
2. To perform a mathematical '**`Exclusive-or`**' of two integers. The result is a bitwise 'Exclusive-or' of the two numbers. For example:
   `10110001 Xor 01100110 = 11010111`

Three years later **Stephen Grossberg** published a series of papers introducing networks capable of modelling differential, contrast-enhancing and **XOR functions.** Nevertheless, the often-miscited **Minsky/Papert** text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s.

The **perceptron** is a simplified model of a biological **neuron.** While the complexity of **biological neuron models** is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.

So far the history. It really took quite some time to come to a level of computing and to develop Pascal so we would be able to do this experiment again in a modern computer language.

MAX KLEINER HAS WRITTEN THE PROGRAM FOR PASCAL BY USING THE ROSETTA

There are of course scripts with larger data- and trainingsets and larger embedding space that could give additional accuracy points.
We will see that rate of improvement drops quite markedly as you increase the number of training runs from 1 to 14.
A **perceptron** is an algorithm used in machine-learning. It it the simplest of all neural networks, consisting of only one neuron, and is typically used for pattern or image recognition.

The script will give an insight in the following 5 steps:
1. Define the target function to predict
2. Set an activator (**`sigmoid, linear`**)
3. Declare the loss function (**`error delta`**) to train
4. Declare an optimizer to minimize the loss **`(error)`**
5. Test and Predict a subset

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. Often, sigmoid function refers to the special case of the logistic function shown in the first figure and defined by the formula.

Logistic activation function
In computational networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

Binary or binomial classification is the task of classifying the elements of a given set into two groups (*predicting which group each one belongs to*) on the basis of a classification rule.
Contexts requiring a decision as to whether or not an item has some qualitative property, some specified characteristic, or some typical binary classification include:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.
It infers a function from labeled training data consisting of a set of training examples.

In supervised learning, each example is a pair consisting of an input object (*typically a vector*) and a desired output value (*also called the supervisory signal*).

A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples.
An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.
This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (*see inductive bias*).

The parallel task in human and animal psychology is often referred to as concept learning.

For the purposes of this tutorial I use the simple function y = 2x + 1 to restrict the memory, disk and cpu use.
In machine learning, the Delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network. It is a special case of the more general backpropagation algorithm.


WIKIPEDIA

ARTIFICIAL NEURAL NETWORKS (ANNS)
or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. **They do this without any a priori knowledge about cats,** *e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.*

An **ANN** *is based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an animal brain).*
Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

In common **ANN** *implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs.*
The connections between artificial neurons are called 'edges'.
Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection.

Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers.
Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the **ANN** *approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology.*

ANNs *have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.*

An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each ircular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

First we start with the name of the program, which I hack in the editor of maXbox:

```
program Perceptron1_31;
(*  #sign:max: MAXBOX8: 30/05/2018 14:05:08
 * implements a version of the algorithm set out at
 * http://natureofcode.com/book/
   chapter-10-neural-networks/ ,
 * but without graphics - ;
   classes assumed to be 1, -1
 * http://www.rosettacode.org/wiki/Perceptron#Pascal
 * 1_31: test calls with asserts
 *)
```

The ANN has 3 inputs and one output so we need an array. The inputs are the function value of (2x + 1) < b and the bias. We use a classifier to assign each function value into a negative or positive class.
Here classes actually represent a scale and the underlying value (positive/negative) could be well mapped into a continuous range.
We could make use of this feature by computing a regression instead of a classification.
Then the slope 2 of 2x is a weight coefficient and 1 is just the bias as the interceptor.

```
function targetOutput( a, b: integer ): integer;
//split in two classes: neg & pos against value b
begin
  if a * 2 + 1 < b then
   result:= 1
  else result:= -1;
end;

procedure showTargetOutput(var refval: string);
var x, y : integer;
begin
  for y:= 10 downto -9 do begin
    for x:= -9 to 10 do
     if targetOutput( x,y )= 1 then begin
       write( '#' )
       refval:= refval +'#'
     end else begin
       write( 'O' );
       refval:= refval +'O'
     end;
    writeln('')
  end;
  writeln('')
end;
```

```
type TAofReal= array[0..2] of real;
function feedForward(ins: array of integer; ws: array
of real):
                        integer; forward;
//We can change LEARNRATE to see another range output
within an optimizer.

Const LEARNRATE = 0.01;
   BIAS = 1;
```

Since this rule is linear, each feature makes an independent contribution to the score of a class or instance. This contribution depends largely on the estimated weights and a bias. So next we see the output function which splits the input into positive or negative classes.
These conditions are called binary splits because they divide the instance space into two groups: those that satisfy the condition and those that dont. We can also split the instance space into more than two segments to create non-binary splits. For instance, where **f(X) = 0; 0 < F(X) < 20; F(X) > 20**, and so on.

```
procedure showOutput( ws: array of real; var resval:
string );
var inputs: array[0..2] of integer;
  x, y: integer;
  tmpstr: string;
begin
  inputs[2]:= BIAS; (* bias *)
  tmpstr:=''; resval:='';
  for y:=10 downto-9 do begin
    for x:=-9 to 10 do begin
      inputs[0]:= x;
      inputs[1]:= y;
      if feedForward(inputs, ws) = 1 then
       tmpstr:= tmpstr+'#'
        else tmpstr:= tmpstr +'O';
    end;
    writeln(tmpstr)
    resval:= resval + tmpstr;
    tmpstr:='';
  end;
  writeln('')
end;
```

So this perceptron attempts to separate input into a positive and a negative class with the aid of the linear function. The inputs are each multiplied by weights, random weights at first, and then summed up. Based on the sign of the sum a decision is made and return.
Next you see the initial <randomWeights> procedure and the following <feedForward> function with the sum iteration. The perceptron outputs 1 if the sum of its inputs multiplied by its input weights is positive, otherwise -1.

```
procedure randomWeights(var ws : TAofReal );
(* start with random weights -- pass by reference *)
var i: integer;
begin
  randomize; (* seed random-number generator *)
  for i:= 0 to 2 do
    ws[i]:= randomF * 2- 1;
end;
```

The objective is now to generate a feed forward code that allows to find the best parameters ws and b, that from input data, adjunct them to y data, in our case it will be a straight line defined by y_data = ws * x_data + b.

```
function feedForward(ins: array of integer; ws: array of real): integer;
var sum: real; i : integer;
begin
  sum:= 0;
  for i:= 0 to 2 do
    sum:= sum+ ins[i] * ws[i];
  if sum > 0 then
     result:= 1
  else result:= -1
end;
```

In order for the perceptron to make the right decision, it needs to train with input for which the correct outcome is known, so that the weights can slowly be adjusted until they start producing the desired results.
We can call this the backpropagation call.
The emergent approach of an ANN resolves the adaptability and learning issues by building massively parallel models, analogous to neural networks, where information flow is represented by a (back) propagation of signals from the input nodes. On the other hand, emergent architectures are easier to design first, but they must be trained in order to produce useful behavior.

Of course there are plenty of optimizer routines like cross entropy or a derivative of the cross-entropy cost function for the softmax function. Cross entropy can be used to define the loss function in machine learning and optimization. There are many situations where cross-entropy needs to be measured but the distribution is unknown. An example is language modeling, where a model is created based on a training set, and then its cross-entropy is measured on a test set to assess how accurate the model is in predicting the test data.

In our perceptron we use a simple and single called delta routine which defines the error as the difference between target output and the feed forward calculation of the weights. By the way a cost function or a loss function means the same, namely to reduce error difference and to gain accuracy.

```
Error:= targetOutput(x, y) -
feedForward(inputs, ws);
```

Entropy: This measure of disorder or impurity is based on the expected information content. Consider a message telling you about the class of a series of randomly drawn samples. The purer the set of samples, the more predictable this message becomes (low entropy),
and therefore the smaller the expected information. Entropy is 0 if n samples in the data are the same. Entropy is high if they are all different.

```
procedure train(var ws: TAofReal; runs: integer;
LEARNRATE: real);
(* pass the array of weights by reference so it can be
modified *)
var inputs: array[0..2] of integer;
  error: real;
  x,y, i,j : integer;
begin
  inputs[2]:= BIAS; (* bias *)
  for i:= 1 to runs do begin
    for y:= 10 downto -9 do begin
     for x:= -9 to 10 do begin
       inputs[0]:= x;
       inputs[1]:= y;
       Error:= targetOutput(x, y )
             - feedForward(inputs, ws);
      for j:= 0 to 2 do
       ws[j]:= ws[j]+ Error * inputs[j]* LEARNRATE;
     end;
    end;
  end;
end;
```

We can improve the accuracy by tuning the meta-parameters like the learning rate or the number of runs (steps), especially if we use a different module A validation set is very important if we want to get any reasonable results, because it is very easy to set up a model that learns to predict the training data without generalizing well to the test set.
Let us run a couple of trainings and testing evaluations to see how using a range can affect the accuracy and then predict some values. But beware of overfitting; it is to easy to set-up a model that learns to predict or memorize the training data (learning by heart) without generalizing well to the test set or a new productive set (evaluations).

```
procedure testAll(const ws : TAofReal; runs : integer );
var inputs: array[0..2] of integer;
  x,y, i,j: integer;
  outputs: integer;
begin
  inputs[2]:= BIAS; (* bias *)
  for i:= 1 to runs do begin
    for y:= 10 downto -9 do begin
     for x:= -9 to 10 do begin
       inputs[0]:= x;
       inputs[1]:= y;
      for j:= 0 to 2 do begin
       outputs:= outputs+ round(ws[j]* inputs[j]);
      end;
     //print(itoa(outputs)+', ')
      if outputs >= 1 then print('#') else write('O');
      outputs:= 0;
     end;
    writeln('')
   end;
  end;
end;
```

```
procedure Predict(const ws: TAofReal; a,b: integer);
var inputs: array[0..2] of integer;
  outputs, j: integer;
  cls: string;
begin
  inputs[2]:= BIAS; (* bias *)
  inputs[0]:= a;
  inputs[1]:= b;
  outputs:= 0;
  for j:= 0 to 2 do
   outputs:= outputs+ round(ws[j]* inputs[j]);
  if ((a*2)+1) < b then cls:=('pos') else cls:=('neg');
  println(itoa(outputs)+',  is class: '+cls)
   if cls = 'pos' then
     Assert((2*a)+1 < b,'Test Cond Error must <')
   else
    Assert((2*a)+1 >= b,'Test Cond Error must >=');
end;

procedure PredictFloat(const ws: TAofReal;
   a,b: integer);
var inputs: array[0..2] of integer;
  j: integer; outp: float;
  cls: string;
begin
  inputs[2]:= BIAS; (* bias *)
  inputs[0]:= a; inputs[1]:= b;
  outp:= 0;
  for j:= 0 to 2 do
   outp:= outp+ (ws[j]* inputs[j]);
  if ((a*2)+1) < b then cls:=('pos') else cls:=('neg');
  println(floattoStr(outp)+',  is class: '+cls)
   if cls = 'pos' then
     Assert((2*a)+1 < b,'Test Cond Error must <')
   else
    Assert((2*a)+1 >= b,'Test Cond Error must >=');
End;
```

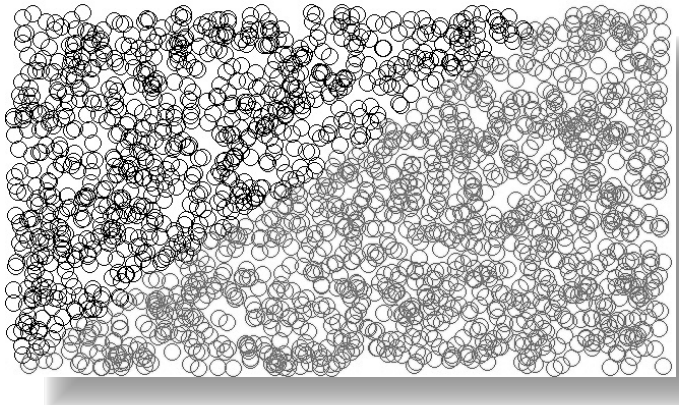
```
var weights: TAofReal;
  refval, resval: string; p,q: integer;

begin //@main
  writeln( 'Target output for the function f(x) = 2x + 1:' );
  showTargetOutput(refval);
  randomWeights(( weights) );
  writeln('Output from untrained perceptron:' );
  showOutput( weights, resval );
  for it:= 0 to 2 do print(floattostr(weights[it])+', ');
  writeln('--------------------------------------')
  train( weights, 1, LEARNRATE);
  writeln('Output from perceptron after 1 training run:' );
  showOutput( weights, resval );
  for it:= 0 to 2 do print(floattostr(weights[it])+', ');
  writeln('--------------------------------------')
  train( weights, 14, LEARNRATE);
  writeln('Output from perceptron after 14 training runs:' );
  showOutput( weights, resval )
  for it:= 0 to 2 do printF('weights %10.4f ',[weights[it]]);
  writeln('--------------------------------------')
  if strcompare(refval,resval)= 0 then
   writeln('accuracy ~100')
  else
    writeln('accuracy NOT 100');
  writeln('')
  writeln('Now testing values-----------')
  testAll( weights, 1 );
  println('')
  writeln('Predict values---------------')
  Predict( weights, 15,5);
  Writeln('')
  writeln('Predict line values----------')
  for p:= -9 to 10 do
   //for q:= -9 to 10 do
    Predict(weights, p, p);
  // TEst Float
  for p:= -9 to 10 do
   //for q:= -9 to 10 do
    PredictFloat(weights, p, p);
End.
```

```
ref: delta list, 0 is no error

err(0,2)/targ(-1,1)/feed(-1,1)
--------------------
 0  :  1  1
 0  : -1 -1
 2  :  1 -1
-2  : -1  1

optimal weights by 4000 runs
weights  -1.0114
weights   0.5006
weights  -0.5560
--------------------------------------
```



REF:

https://maxbox4.wordpress.com/code/
https://www.academia.edu/36608990/TensorFlow_AI_Demo
https://www.scribd.com/document/378905755/tensorflow-machinelearning-task9

```
F:\SPP\Blaise\Blaise_UK_72_2018\Authors\Max Kleiner\Lazarus\lib\i386-win32\Perceptron1_31.exe
Target output for the function f(x) = 2x + 1:
```

Perceptron1_31.lpr

```
    writeln('Output from untrained perceptron:' );
    showOutput( weights, resval );
    for it:= 0 to 2 do write(floattostr(weights[it])+', ');
    writeln('---------------------------------------');
    train( weights, 1, LEARNRATE);
    writeln('Output from perceptron after 1 training run:' );
    showOutput( weights, resval );
    for it:= 0 to 2 do write(floattostr(weights[it])+', ');
    writeln('---------------------------------------');
    train( weights, 14, LEARNRATE);
    writeln('Output from perceptron after 14 training runs:' );
    showOutput( weights, resval );
    for it:= 0 to 2 do writeln(format('weights %10.4f ',[weights[it]]));
    writeln('---------------------------------------');
    if compareStr(refval,resval)= 0 then
       writeln('accuracy ~100') else
         writeln('accuracy NOT 100');
    writeln('');
          writeln('Now testing values-----------');
    testAll( weights, 1 );
    writeln('');
    writeln('Predict values---------------');
    Predict( weights, 15,5);
    Writeln('');
    writeln('Predict line values----------');
    for p:= -9 to 10 do
      //for q:= -9 to 10 do
        Predict(weights, p, p);
    // TEst Float
    for p:= -9 to 10 do
      //for q:= -9 to 10 do
        PredictFloat(weights, p, p);
                sleep(1000);
                //halt(1);
                ReadLn();
          Readln();

End.


ref: delta list, 0 is no error

err(0,2)/targ(-1,1)/feed(-1,1)
-----------------------
   0   :    1    1
   0   :   -1   -1
   2   :    1   -1
  -2   :   -1    1

optimal weights by 4000 runs
weights    -1.0114
weights     0.5006
weights    -0.5560
---------------------------------------

ref:

https://maxbox4.wordpress.com/code/
https://www.academia.edu/36608990/TensorFlow_AI_Demo
https://www.scribd.com/document/378905755/tensorflow-machinelearning-task9

----File newtemplate.txt exists - cached - now saved!
```

44: 34 INS F:\SPP\Blaise\Blaise_UK_72_2018\Authors\Max Kleiner\Lazarus\Perceptron1_31.lpr

Messages | Watch List | Assembler | BreakPoints

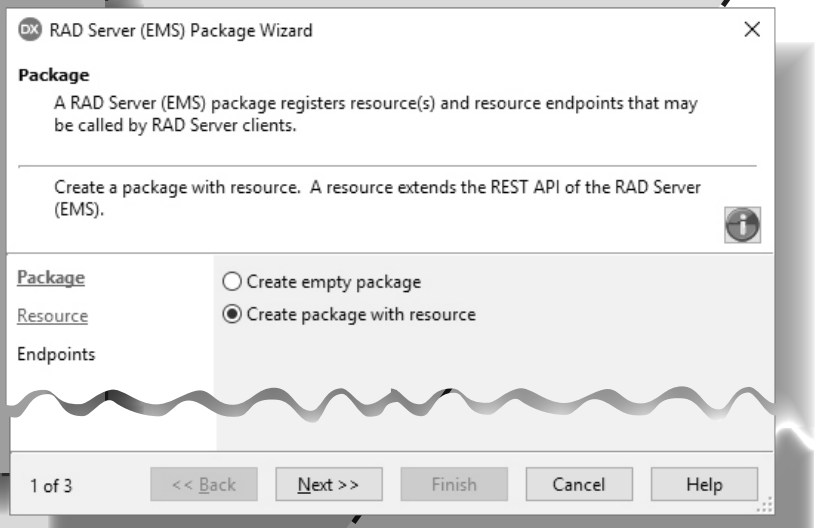
```
Compile Project, Target: lib\i386-win32\Perceptron1_31.exe: Success, Warnings: 1, Hints: 2
Perceptron1_31.lpr(185,27) Warning: Local variable "outputs" does not seem to be initialized
Perceptron1_31.lpr(240,30) Hint: Variable "weights" does not seem to be initialized
Perceptron1_31.lpr(234,31) Note: Local variable "q" not used
```

This output is created in Lazarus. The project works in **Lazarus** and is also available for **Delphi 7** and **Delphi Tokyo** where we needed to make some slight alterations
You can download all code from your Downloadpage
https://www.blaisepascalmagazine.eu/my-downloads/

BY BOB SWART

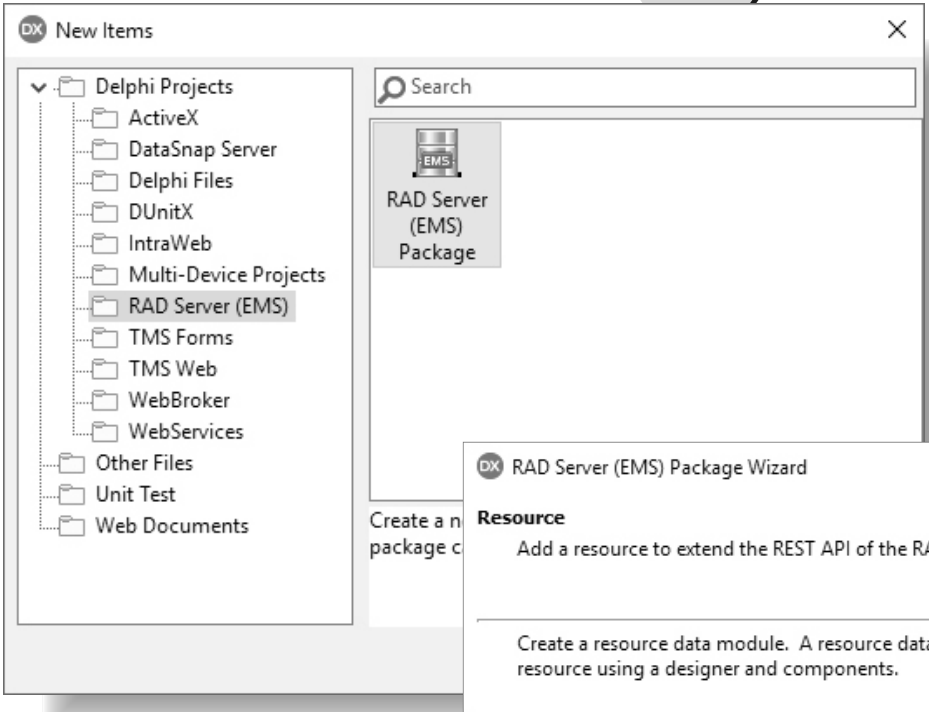
starter expert DX ENTERPRISE

Embarcadero RAD Server (EMS) is a technology to create and deploy REST (micro) Services, and TMS WEB Core is a technology to create and deploy Single Page Web Applications (SPA). In this article, I'll demonstrate how we can combine these two techniques for even more powerful applications.



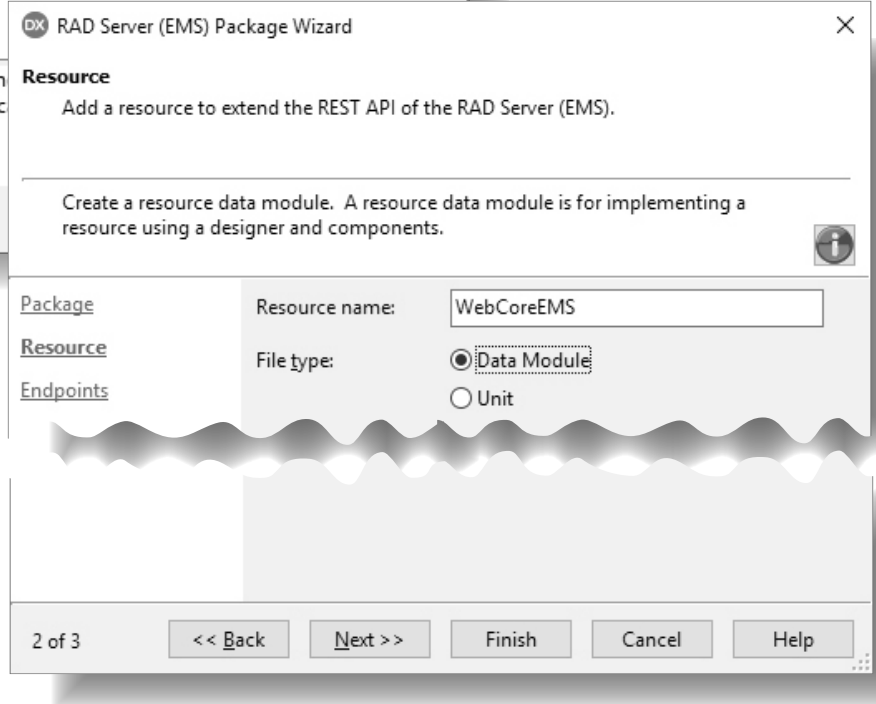
RAD SERVER (EMS)

Let's start by building a **RAD Server (EMS) Micro Service** using **Delphi Enterprise.** Do **File | New – Other**, and from the **RAD Server (EMS)** category double-click on the **RAD Server (EMS) Package** icon to create a new **RAD Server (EMS) package.**



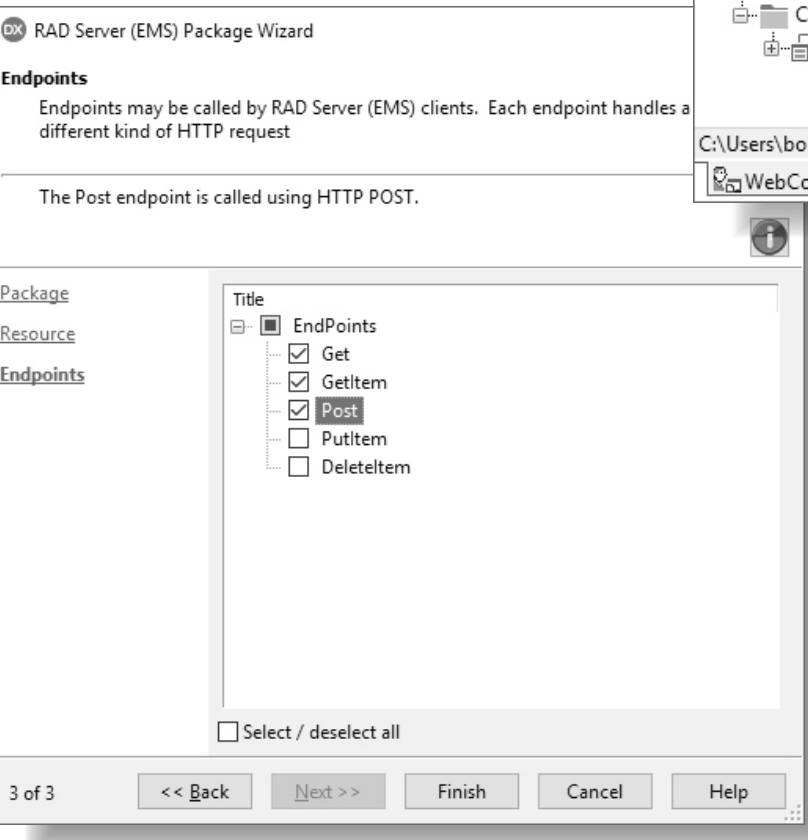
The wizard that follows gives us a choice of creating a new empty package, or a package with a resource. Although we can always add resources (*even to an empty package*), it's easiest to create a package with a resource right from the start, so select the option to "Create package with resource".

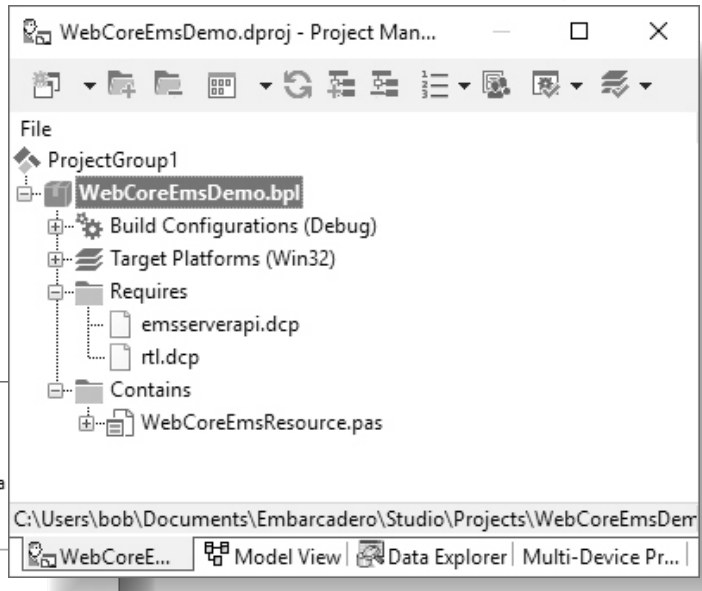
If we click on the **Next** button, we can specify the name of the resource (*which I've called WebCoreDemo for now*). We can also determine the file type of the resource: either a normal unit, or a data module. The advantage of a data module is that we can use it as a container for non-visual components (*for example for database access*) without having to create and configure all components at run-time. So, for our demo, I've specified **WebCoreEMS** as name of the resource, and selected the **Data Module** as file type:



The last page of the **RAD Server (EMS) Package Wizard** can be used to specify the endpoints that we want to expose. By default, the **`Get`** and **`GetItem`** endpoints are selected, corresponding to the **`GET HTTP`** verb. The Post endpoint corresponds to the **`POST HTTP`** verb, while PutItem and DeleteItem correspond to the **`PUT`** and **`DELETE HTTP`** verbs.

For our demo, I do not only want to use the **`GET HTTP`** verb, but also demonstrate the use of the **`POST HTTP`** verb, so make sure to select **`Post`** as well as the default choices **`Get`** and **`GetItem`.**





SQL

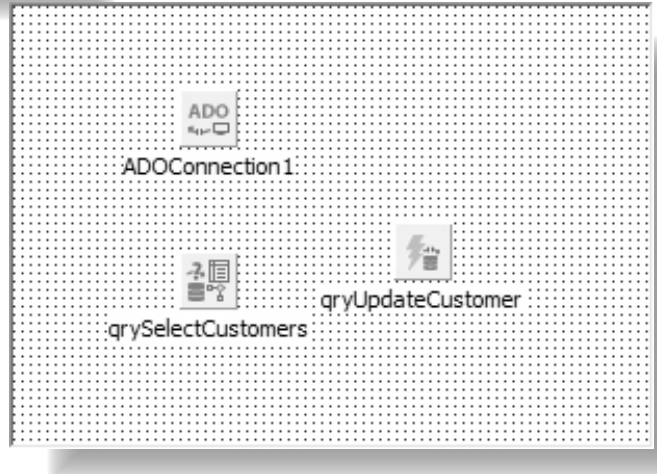
Place a **`TADOConnection`** component on the **`EMS data module`**, and configure the **`ConnectionString`** property to connect to the server machine with your database (*in my case, that's* **SQL Server Express** *with the* **Northwind example database**).

We also need a **`TADODataSet`** component called **`qrySelectCustomers`** to select all records from the **`Customers`** table, and a **`TADOCommand`** called **`qryUpdateCustomer`** component to update one customer record of the **`Customer table`**.

Clicking on **Finish** will generate the new **`EMS Package`** in an unnamed project with unnamed unit (*by default* **`Project1.bpl`** *with* **`Unit1.pas`**). I've renamed the project to **`WebCoreEmsDemo`** and the resource to **`WebCoreEmsResource.`**

The **`WebCoreEmsResource`** unit contains the empty **`data module`**. We can place a data-access component on it, to get our hands on data from a database. Although **RAD Server (EMS)** is deployed with an **InterBase** database, we are not required to use that database for our other tables or queries. For this demo, I want to use a **SQL Server database**, specifically the **SQL Server Northwind example database**, and work with the **`Customers`** table.

Delphi includes a number of data access technologies, the most recent and recommend one is **FireDAC.** However, when using **SQL Server** we can also use **dbGo** for **ADO,** a low-level way to work with the **SQL Server database**, without the need for additional deployment DLLs.



The **`CommandType`** property of the **`qrySelectCustomers`** is set to **`cmdText`**, and the **`CommandText property`** contains the **`SQL`** query **`SELECT * FROM Customers`**

The **qryUpdateCustomer** has the **CommandType** also set to cmdText and the **CommandText** assigned to the following **SQL**

```
UPDATE Customers
   SET CompanyName = :CompanyName,
       ContactName = :ContactName
 WHERE CustomerID  = :CustomerID
```

This specific update statement will change the **CompanyName** and **ContactName** from the **Customers** table, for a give **CustomerID.** Of course, we can modify other fields as well, but this is just an example for the demo of this paper. The query has three parameters that need to be filled before we can execute it to update the **Customer** record.

```
procedure TCustomer.Read(DS: TDataSet);
begin
 FCustomerID := DS.FieldByName('CustomerID').AsString;
 FCompanyName := DS.FieldByName('CompanyName').AsString;
 FContactName := DS.FieldByName('ContactName').AsString;
 FContactTitle := DS.FieldByName('ContactTitle').AsString;
 FAddress := DS.FieldByName('Address').AsString;
 FCity := DS.FieldByName('City').AsString;
 FRegion := DS.FieldByName('Region').AsString;
 FPostalCode := DS.FieldByName('PostalCode').AsString;
 FCountry := DS.FieldByName('Country').AsString;
 FPhone := DS.FieldByName('Phone').AsString;
 FFax := DS.FieldByName('Fax').AsString;
end;
```

CUSTOMER

Now it's time to implement the **Get, GetItem** and **Post** methods of the **WebCoreEMS resource.** For the **Get** and **GetItem,** I want to produce pure **JSON** output that others can consume easily, including the **TMS WEB Core** client components. For that purpose, I've written a special unit that defines a class **TCustomer** with private fields (*corresponding to the columns in the Customer table*) and a method **Read** that takes a **TDataSet** and reads the columns from the dataset into the class instance fields.

```
type
 TCustomer = class
 private
  FCustomerID: String;
  FCompanyName: String;
  FContactName: String;
  FContactTitle: String;
  FAddress: String;
  FCity: String;
  FRegion: String;
  FPostalCode: String;
  FCountry: String;
  FPhone: String;
  FFax: String;
 public
  procedure Read(DS: TDataSet);
 end;
```

This is just for one customer, so we also need a **TCustomers** class to maintain an open array of customers, with a procedure to read the whole dataset into the array, plus a function to return the collection as one big **JSON** string:

```
TCustomers = class
 public
  Customers: array of TCustomer;
 public
  destructor Destroy; override;
  procedure Read(DS: TDataSet);
  function ToJSON: String;
 end;
```

Reading the entire **TDataSet** is just a loop around the **Read** call:

```
procedure TCustomers.Read(DS: TDataSet);
var
 i: Integer;
begin
 SetLength(Customers, DS.RecordCount);
 DS.First;
 i := 0;
 while not DS.Eof do
 begin
  Customers[i] := TCustomer.Create;
  Customers[i].Read(DS);
  Inc(i);
  DS.Next;
 end;
end;
```

Producing the **JSON** string is the result of the **TJson.ObjectToJsonString** from the **REST.JSON** unit:

```
function TCustomers.ToJSON: String;
begin
 Result := TJson.ObjectToJsonString(Self)
end;
```

```
procedure TWebCoreEMSResource1.Get(const AContext: TEndpointContext;
 const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
 Custs: TCustomers;
begin
 qrySelectCustomers.Close;
 qrySelectCustomers.Open;
 Custs := TCustomers.Create;
 try
  Custs.Read(qrySelectCustomers);
  AResponse.StatusCode := 200;
  AResponse.Body.SetBytes(TEncoding.ASCII.GetBytes(Custs.ToJSON), 'application/json');
 finally
  Custs.Free
 end;
end;
```

The result of this method can be seen in a browser by calling the URL of the EMS Service, followed by the **/WebCoreEMS** resource specifier:
http://localhost:8080/WebCoreEMS

The *(formatted)* JSON output looks as follows:

```
{
  "customers": [
    {
      "customerID": "ALFKI",
      "companyName": "Alfreds Flipperkast",
      "contactName": "Maria abc 123123",
      "contactTitle": "Sales Representative",
      "address": "Obere Str. 57",
      "city": "Berlin",
   ,
   ...
}
```

This is a **JSON** array that can be consumed by any self-respecting **JSON** client, including **Delphi** clients and **TMS EMS Web Core** clients.

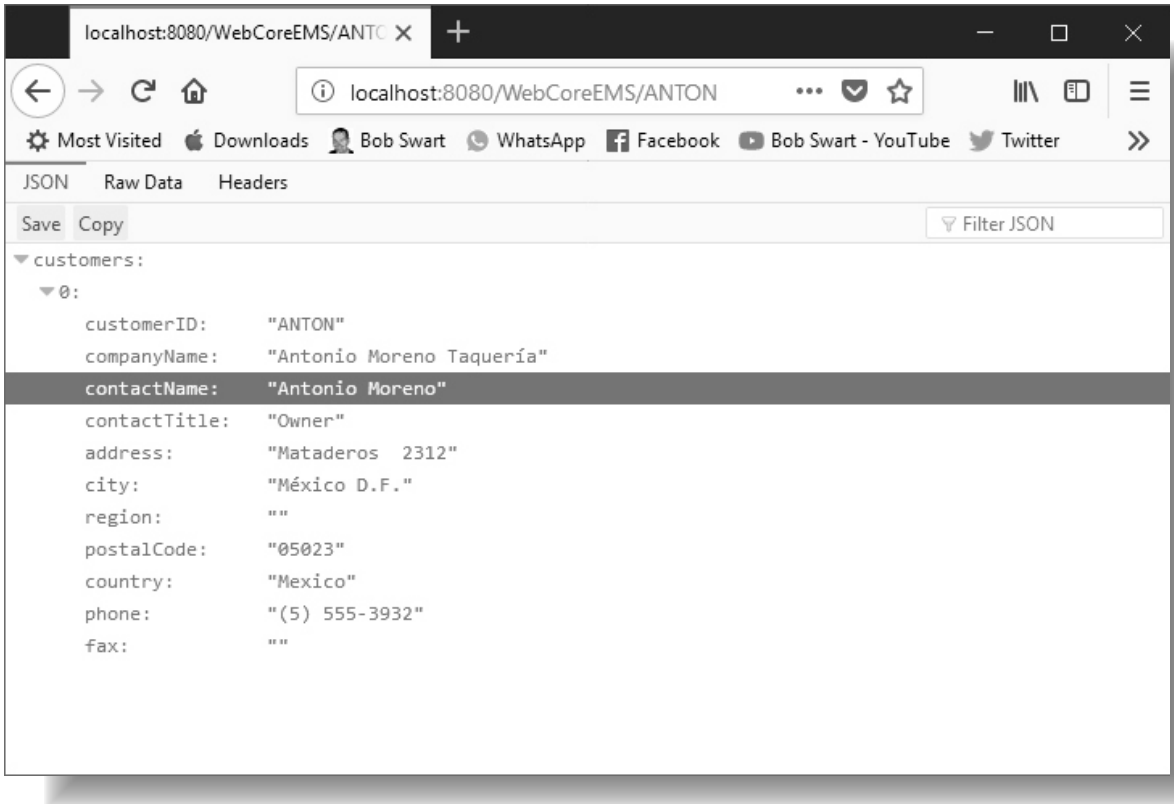
GETITEM

The **GetItem** method gets an Item parameter, to look at a specific Customer record. This item is usually the primary key or another unique identifier. In our case, it can be the **CustomerID** (*which consists of a 5-character identifier for the customer record*).
The code to return the single customer can be as follows, using a simple filter to limit the dataset to just a single record:

```
procedure TWebCoreEMSResource1.GetItem(const AContext: TEndpointContext;
 const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
 Custs: TCustomers;
 CustomerID: string;
begin
 CustomerID := ARequest.Params.Values['item'];
 qrySelectCustomers.Close;
 qrySelectCustomers.Open;
 qrySelectCustomers.Filter := 'CustomerID = ''' + CustomerID + '''';
 qrySelectCustomers.Filtered := True;
 Custs := TCustomers.Create;
 try
  Custs.Read(qrySelectCustomers);
  AResponse.StatusCode := 200;
  AResponse.Body.SetBytes(TEncoding.ASCII.GetBytes(Custs.ToJSON),
   'application/json');
 finally
  qrySelectCustomers.Filtered := False;
  Custs.Free
 end;
end;
```

The result of the **GetItem** method can be seen by adding a **CustomerID** to the previous URL, for example as follows:

http://localhost:8080/WebCoreEMS/ANTON



Similar output compared to the **Get,** but this time only the result of a single customer.

POST

The update of a single **Customer** record is done using the query with 3 parameters. This means we have to modify the method definition to bind the names of the parameters to the vaules. The definition of the **GetItem** method was as follows:

Based on this, we can define the **Post** method as follows:

```
[ResourceSuffix('UpdateCustomer/{CustomerID}/{CompanyName}/{ContactName}')]
  procedure GetUpdateCustomer(const AContext: TEndpointContext;
   const ARequest: TEndpointRequest;
   const AResponse: TEndpointResponse);
```

The implementation is straightforward:

```
procedure TWebCoreEMSResource1.Post(const AContext: TEndpointContext;
 const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
begin
 qryUpdateCustomer.Parameters.ParamByName('CustomerID').Value :=
       ARequest.Params.Values['CustomerID'];
 qryUpdateCustomer.Parameters.ParamByName('CompanyName').Value :=
       ARequest.Params.Values['CompanyName'];
 qryUpdateCustomer.Parameters.ParamByName('ContactName').Value :=
       ARequest.Params.Values['ContactName'];
 try
  qryUpdateCustomer.Execute;
  AResponse.StatusCode := 200;
  AResponse.Body.SetBytes(BytesOf('OK'), 'text/plain');
 except
  on E: Exception do
   AResponse.Body.SetBytes(BytesOf(E.Message), 'text/plain');
 end;
end;
```

We can test the **WebCoreEmsDemo** using the **EMSDevServer.exe** for development, but for real deployment, we need to use the **EMSServer.dll ISAPI DLL** and deploy the **WebCoreEmsDemo.bpl** on a web server (using IIS on Windows for example).

DEPLOYMENT

Although you can use the **EMSDevServer.exe** as development and test environment, you really need to deploy the **WebCoreEmsDemo.bpl** using the **ISAPI DLL** version of the **EMSServer.dll**. In fact, I had to deploy a number of files in order to make the **WebCoreEmsDemo.dpl** work:

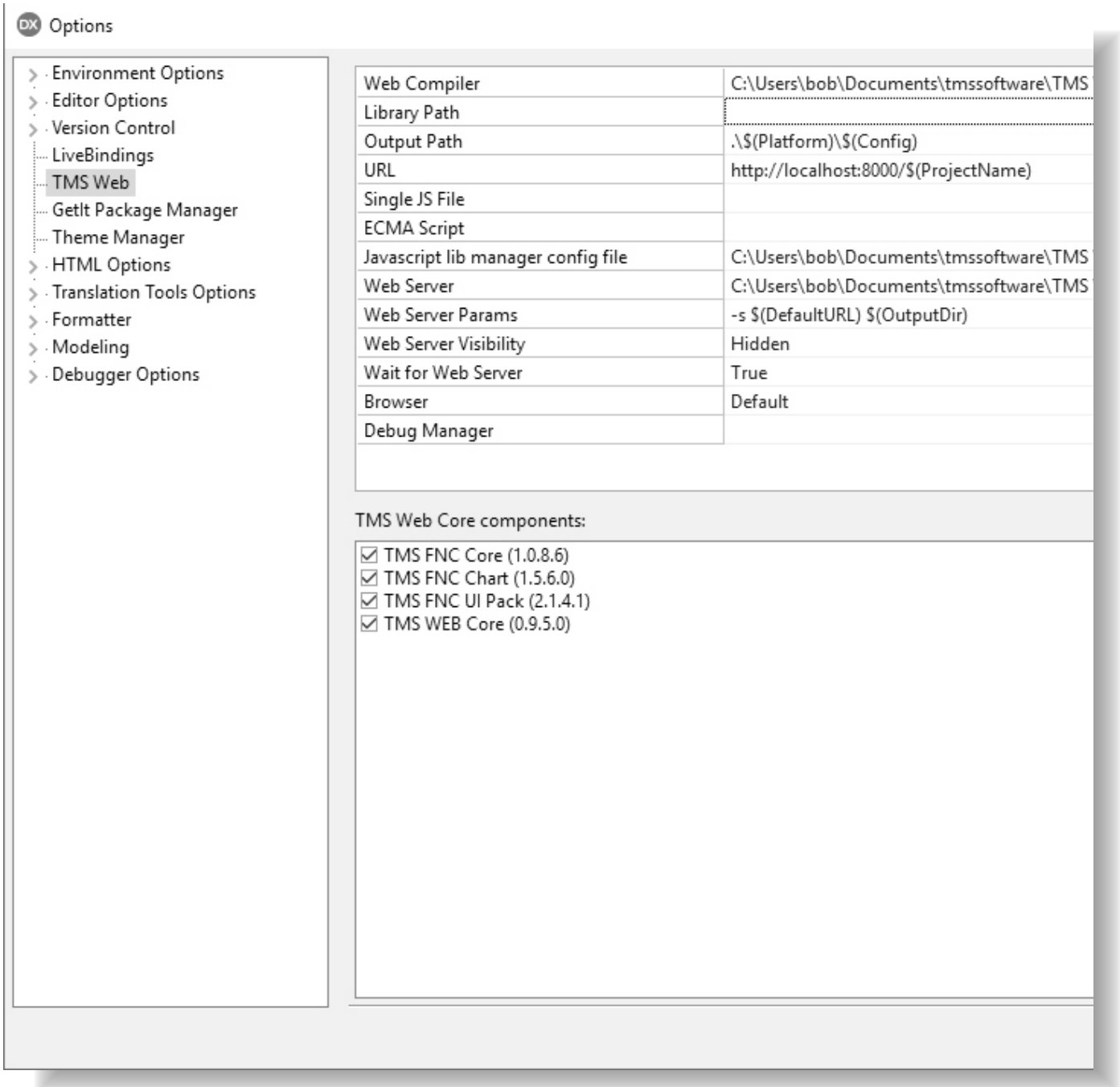
```
adortl250.bpl
bindcomp250.bpl
bindengine250.bpl
borlndmm.dll
CustomIPTransport250.bpl
dbrtl250.bpl
emsclient250.bpl
EMSConsole.dll
emsedge250.bpl
emshosting250.bpl
EMSServer.dll
emsserver.ini
emsserverapi250.bpl
FireDAC250.bpl
FireDACCommon250.bpl
FireDACCommonDriver250.bpl
FireDACIBDriver250.bpl
FireDACSqliteDriver250.bpl
inet250.bpl
RESTComponents250.bpl
rtl250.bpl
soaprtl250.bpl
vcl250.bpl
vcldb250.bpl
vclFireDAC250.bpl
vclx250.bpl
WebCoreEmsDemo.dpl
xmlrtl250.bpl
```

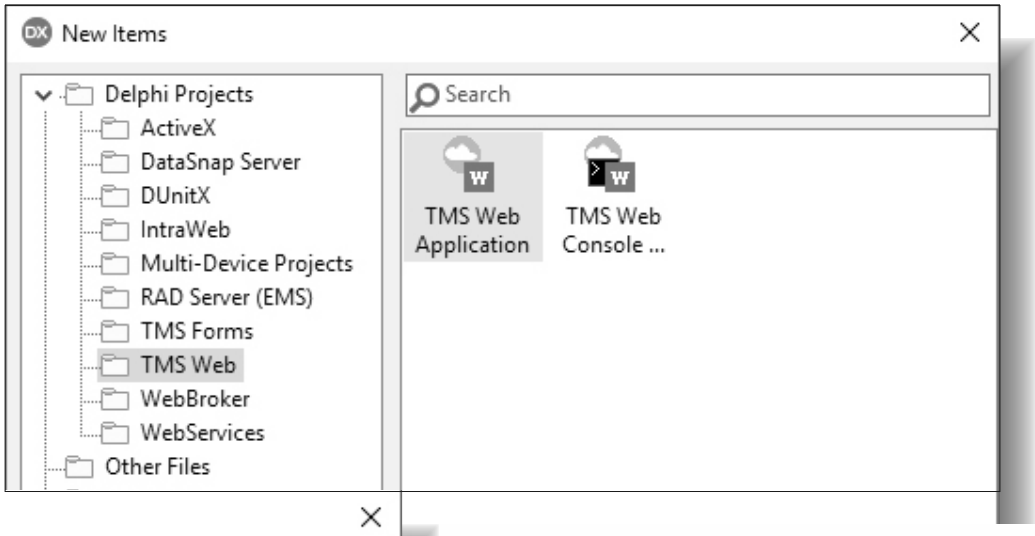
We also need to modify the **emsserver.ini** to include the location of our **EMS web service package**, in the **Server.Packages** section:

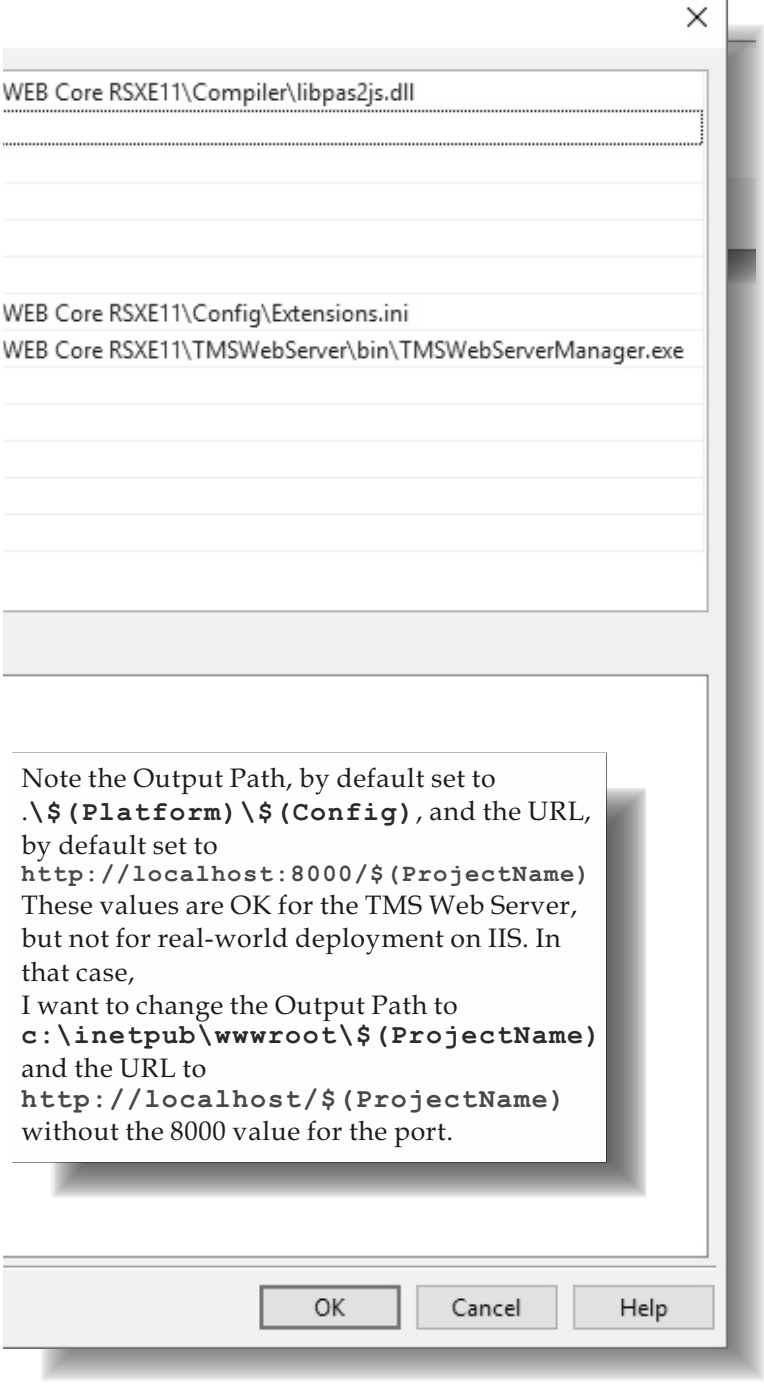
```
[Server.Packages]
;# This section is for extension packages.
;# Extension packages are used to register custom resource endpoints
;c:\mypackages\basicextensions.bpl=mypackage description
c:\inetpub\wwwroot\Deployment\EMS\WebCoreEmsDemo.bpl=WebCoreEMS
```

TMS WEB CORE
`https://www.tmssoftware.com/site/tmswebcore.asp`
Once the **WebCoreEmsDemo** is deployed, we can write the **TMS WEB Core client** to connect to it and work with the data from the **EMS micro service**. Feel free to use the deployed version of the `WebCoreEmsDemo.bpl` as deployed on my web server with my deployment license of **RAD Server (EMS)**.

Before we add the **TMS WEB Core** project, we may first want to configure the **TMS WEB** options. Go to **Tools | Options** and select the **TMS WEB** options.







Note the Output Path, by default set to `.\$(Platform)\$(Config)`, and the URL, by default set to `http://localhost:8000/$(ProjectName)`
These values are OK for the TMS Web Server, but not for real-world deployment on IIS. In that case,
I want to change the Output Path to `c:\inetpub\wwwroot\$(ProjectName)` and the URL to `http://localhost/$(ProjectName)` without the 8000 value for the port.

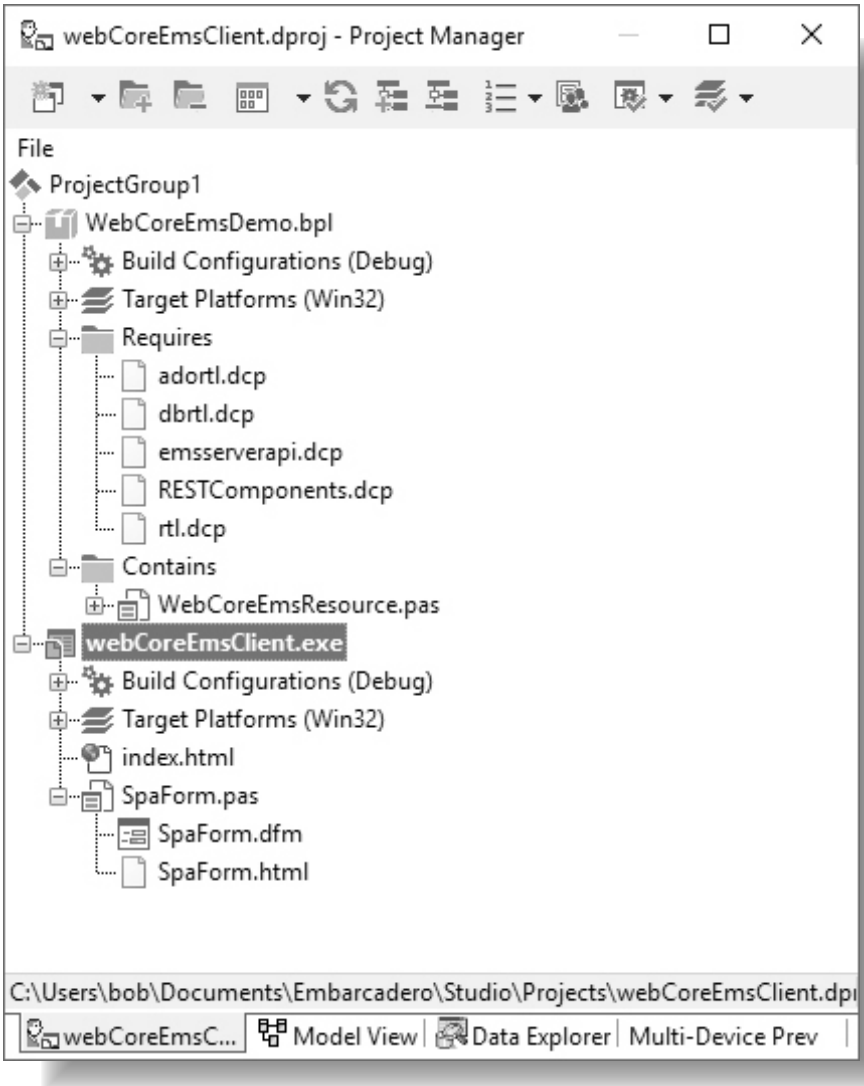
TMS WEB CORE APPLICATION

To add a **TMS WEB Core** project to the project group, do **File | New – Other** again, this time moving to the **TMS Web category.** Two wizards are available here: **TMS Web Application** and **TMS Web Console Application.**

This will produce a new project with a html project file as well as a .pas file with associating .dfm and .html files. I've saved the project in WebCoreEmsClient.dpr, the project HTML file in index.html, and the page in SpaForm.pas (with corresponding .dfm and .html files).

WEB CLIENT DATA

In order to connect to a REST server, and get our hands on JSON data to be able to process and display inside the WEB Core Form, we need to place three non-visual components from the TMS Web DB category of the Tool Palette.
First, a WebClientConnection component, with the URI pointing to your RAD Server (EMS) URI, or if you want to connect to my ready-to-user server, point it to `https://www.bobswart.nl/ems/emsserver.dll/WebCoreEMS` and set the DataNode property to customers to get the records from that array node.
Second, we need a TWebClientDataSet component, with the Connection property pointing to the WebClientConnection1 component.
Third, we need a TWebClientDataSource component, with the DataSet property pointing to the TWebClientDataSet1 component.

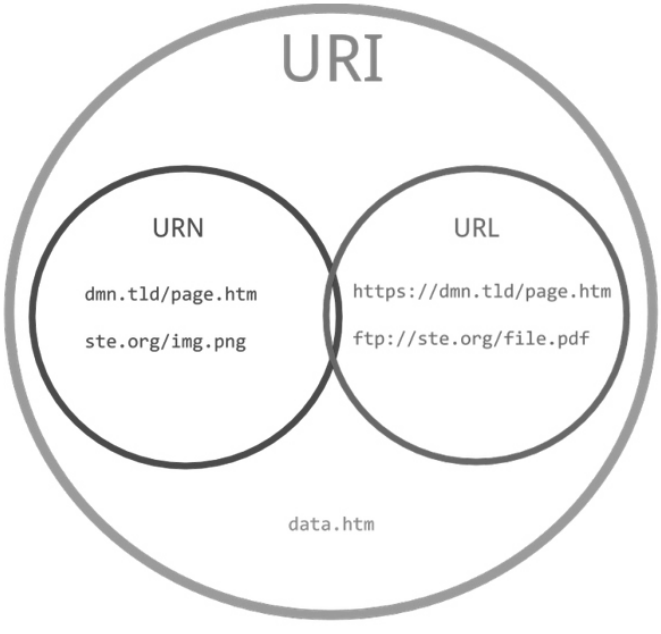


A **URI** can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location")

This is just an "empty" project for now, but it will generate the **JavaScript** file as soon as we compile the project.
Pressing F9 will produce the following files in the specified location in
`c:\inetpub\wwwroot\webCoreEmsClient`

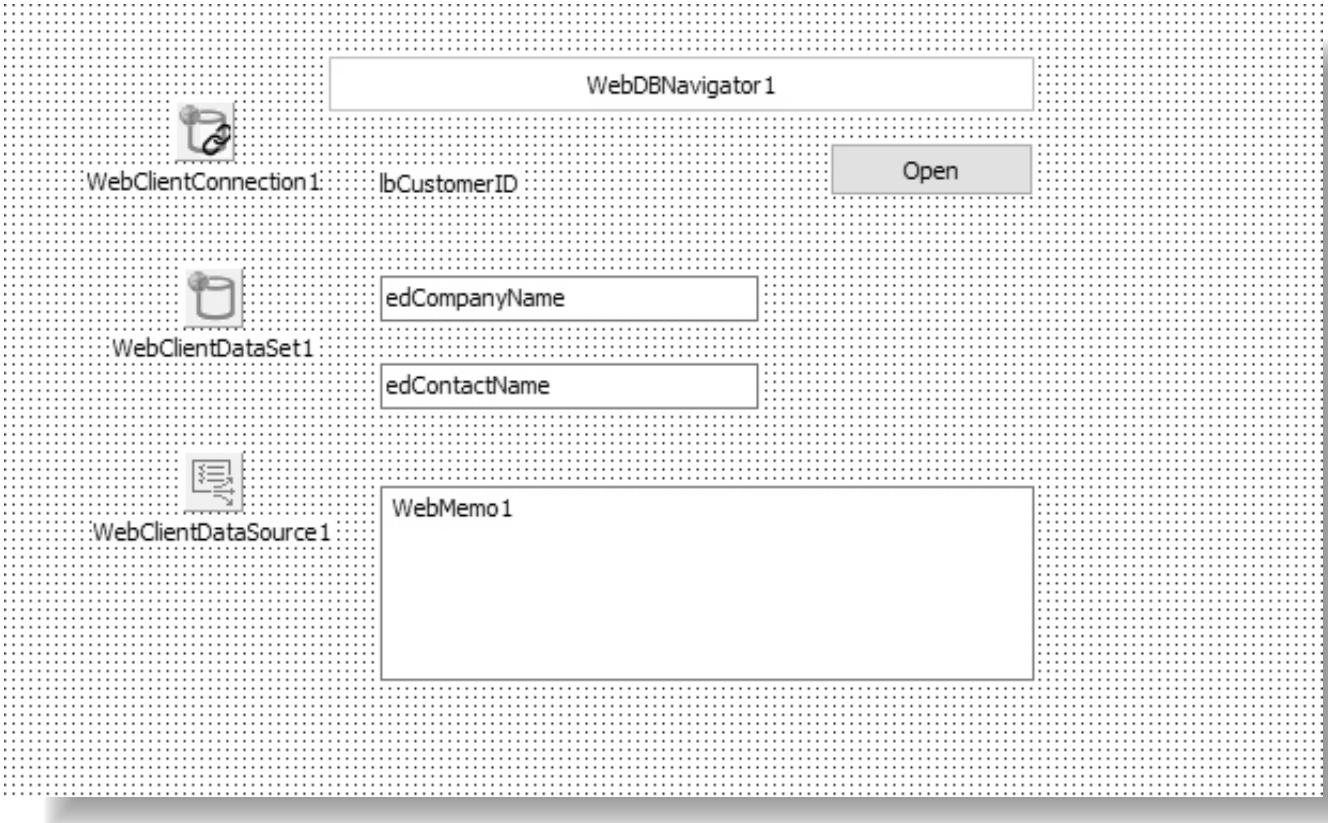
`index.html`
`SpaForm.html`
`WebCoreEmsClient.js`
`WebCoreEmsClient.js.map`

The file index.html is the starting point, and will show up if we open a browser at
`http://localhost/webCoreEmsClient`
(*right now showing an empty page, but we will fill it up with data from the EMS micro service shortly*).

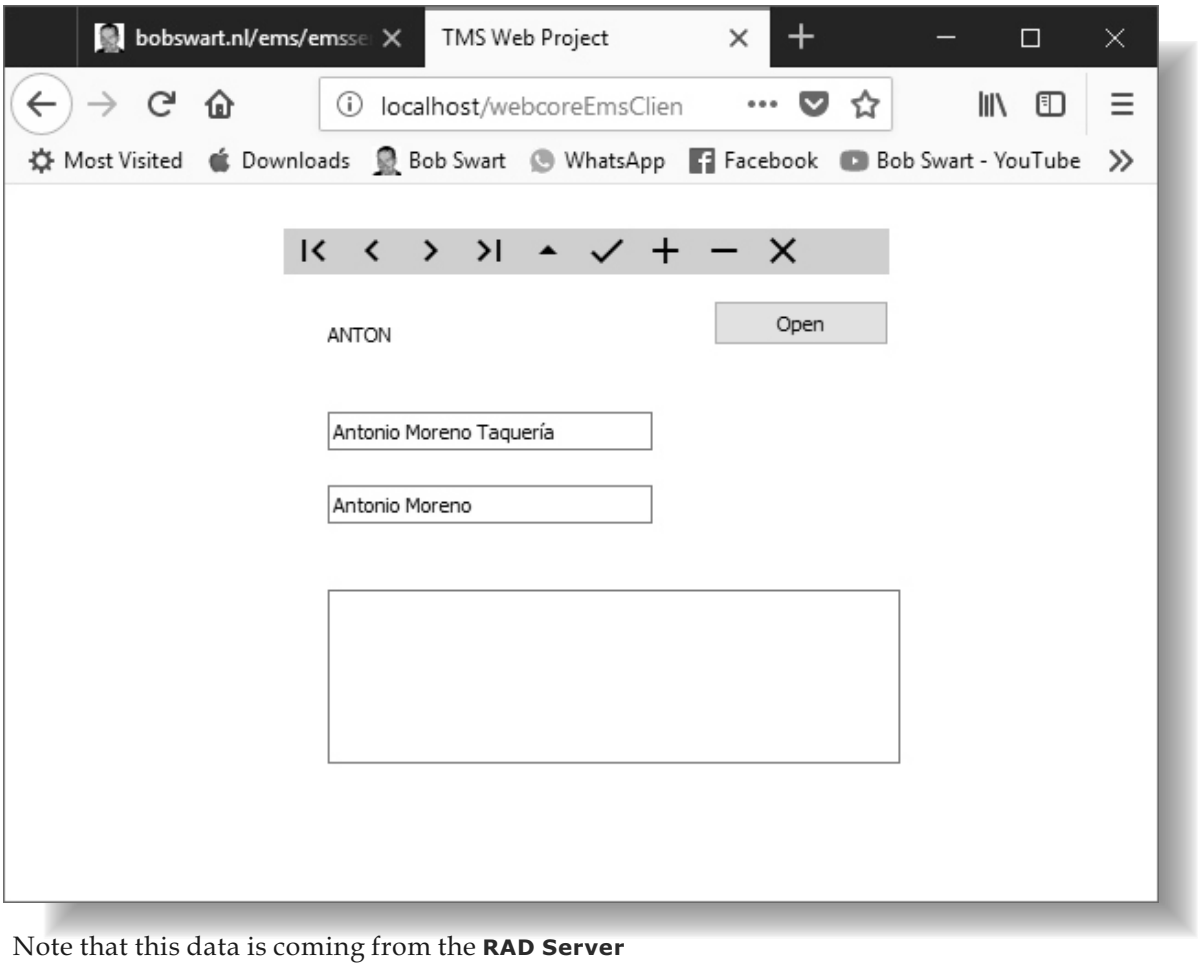


```
procedure TForm1.WebButton1Click(Sender: TObject);
begin
 WebMemo1.Lines.Clear;
 try
  WebClientConnection1.Active := False;
  WebClientDataSet1.Close;
  WebClientConnection1.URI :=
    'https://www.bobswart.nl/ems/emsserver.dll/WebCoreEMS';
  WebClientConnection1.DataNode := 'customers';
  WebClientDataSet1.FieldDefs.Clear;
  WebClientDataSet1.FieldDefs.Add('customerID',ftstring,5); // case sensitive
  WebClientDataSet1.FieldDefs.Add('companyName',ftstring,40);
  WebClientDataSet1.FieldDefs.Add('contactName',ftstring,30);
  lbCustomerID.DataField := 'CustomerID';
  edCompanyName.DataField := 'CompanyName';
  edContactName.DataField := 'ContactName';
  WebClientConnection1.Active := True;
 except
  on E: Exception do
   WebMemo1.Lines.Add(E.Message);
 end;
end;
```

A **TWebMemo** component is added to display some trace and error messages that we may encounter along the way:



Running this application, and clicking on the Open button, will indeed show the data in the label and two edit controls:



Note that this data is coming from the **RAD Server (EMS) micro service** deployed on my server at this time. And you may notice some edits in the data, made by other people who also connected to the **WebCoreEmsDemo** package. We will also be able to make modifications to the data, but let's first examine the events that can happen in the **TMS WEB Core** data access components.

EVENT HANDLERS

The **TWebClientConnection** component has three event handlers that we can hook into: **`BeforeConnect`**, **`AfterConnect`**, and **`OnConnectError`**. I find it instructive to log these events in the memo control, as follows:

```
procedure TFormEmsDemo.WebClientConnection1BeforeConnect(Sender: TObject);
begin
  WebMemo1.Lines.Add('BeforeConnect');
end;

procedure TFormEmsDemo.WebClientConnection1AfterConnect(Sender: TObject);
begin
  WebMemo1.Lines.Add('AfterConnect');
end;

procedure TFormEmsDemo.WebClientConnection1ConnectError(Sender: TObject;
  ErrorCode: Integer);
begin
  WebMemo1.Lines.Add('ConnectError ' + IntToStr(ErrorCode));
end;
```

WEB CLIENT DATA

In order to connect to a **REST server,** and get our hands on **JSON** data to be able to process and display inside the **WEB Core Form,** we need to place three non-visual components from the **TMS Web DB** category of the **Tool Palette**.

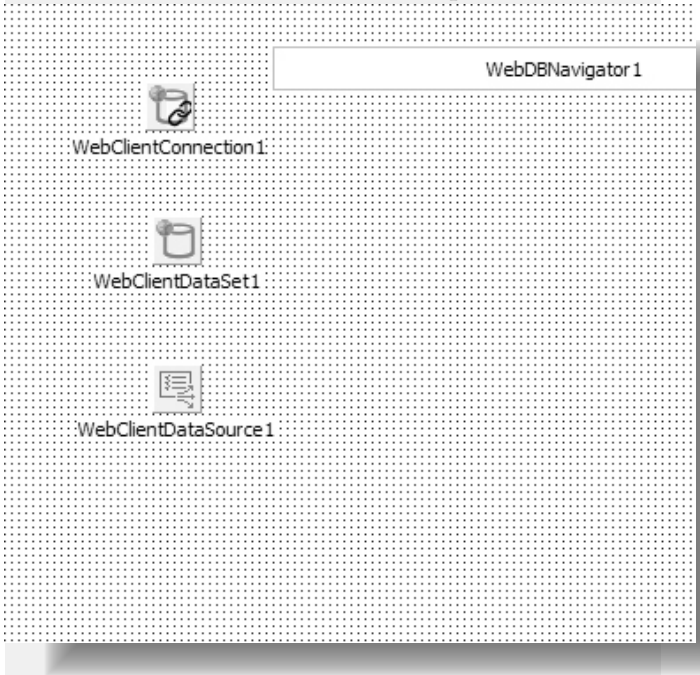
First, a `WebClientConnection` component, with the URI pointing to your **RAD Server (EMS) URI**, or if you want to connect to my ready-to-user server, point it to `https://www.bobswart.nl/ems/emsserver.dll/WebCoreEMS` and set the DataNode property to customers to get the records from that array node.

Second, we need a `TWebClientDataSet` component, with the Connection property pointing to the `WebClientConnection1` component.

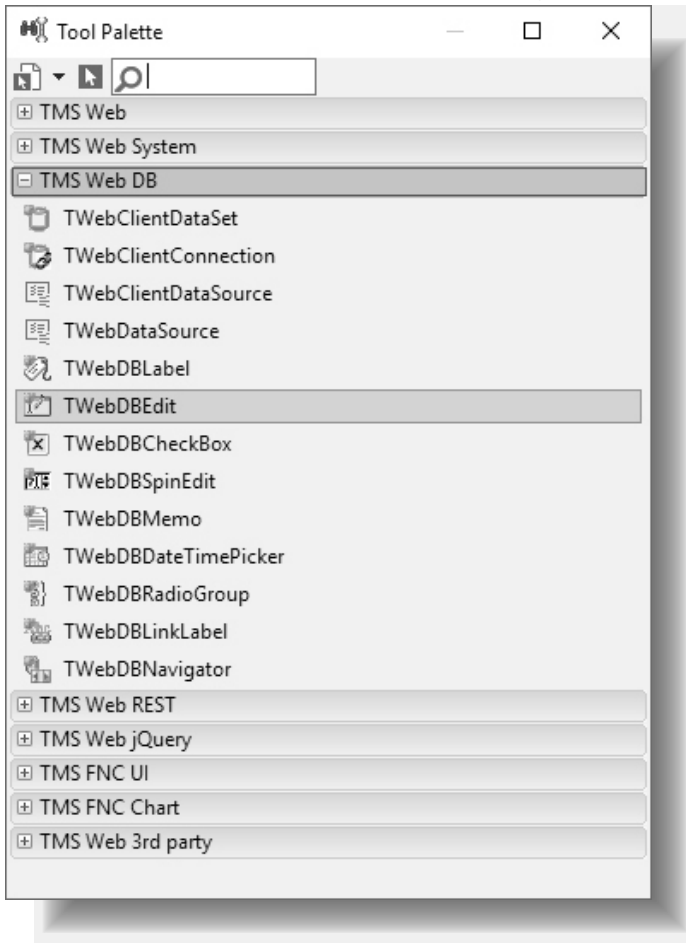
Third, we need a `TWebClientDataSource` component, with the `DataSet` property pointing to the `TWebClientDataSet1` component.

This is all very similar to regular **VCL** data-access and data-aware components, so should not be that difficult to understand.

The first visual component we can now place, is a `TWebDBNavigator` component, with the DataSource property obviously pointing to the `TWebClientDataSource1` component.



Apart from the `TWebDBNavigator`, there are a dozen more **TMS Web DB components** that we can use to build our web user interface, including the useful `TWebDBEdit`.



For my demo application, I have placed one `TWebDBLabel` and two `TWebDBEdit` components on the form, called them resp. `lbCustomerID` , `edCompanyName`, `edContactName`, and connected their `DataSource` property to the `WebClientDataSource1` component.

It is tempting to open up the DataField property as well, to connect these data-aware **TWeb** components to the actual data, but that's not possible at design-time. Not even if we have set the Active property of the `WebClientConnection1` component to `True`. I'm afraid we have to set the DataField properties at runtime.

```
lbCustomerID.DataField:=
'CustomerID';
 edCompanyName.DataField :=
'CompanyName';
 edContactName.DataField  :=
'ContactName';
```

At the same time, we should add persistent fields to the `WebClientDataSet`, and in fact in order to demonstrate how we set the URI of the `WebClientConnection`, and do everything that needs to be done, I've written the following code in the `OnClick` event handler of a `TWebButton` component:

The **TWebClientDataSet** has no event handlers, but there are three event handlers in the **TWebClientDataSource** that we can use: **OnDataChange, OnStateChange** and **OnUpdateData.**
I have logged these in the same way:

```
procedure TFormEmsDemo.WebClientDataSource1DataChange(Sender: TObject;  Field: TField);
begin
 WebMemo1.Lines.Add('DataChange');
end;

procedure TFormEmsDemo.WebClientDataSource1StateChange(Sender: TObject);
begin
 WebMemo1.Lines.Add('StateChange');
end;

procedure TFormEmsDemo.WebClientDataSource1UpdateData(Sender: TObject);
begin
  WebMemo1.Lines.Add('UpdateData');
end;
```

It is interesting to see when these events are fires when we navigate through the data using the **TWebDBNavigator.** If we open the dataset (*click on Open*), we get the **BeforeConnect** and **AfterConnect,** plus the **StateChange** and **DataChange** of the **WebClientDataSource.**

BeforeConnect
AfterConnect
StateChange
DataChange

Now, if I enter the **edContactName** control and start to type, we get another **StateChange** event (*dataset goes in Edit mode*) as well as a DataChange event because I modified the data in the **TWebDBEdit** control.

StateChange
DataChange

If I now click on the Post button of the **TWebDBNavigator,** I get an UpdateData, a **StateChange** and a DataChange event.

UpdateData
StateChange
DataChange

That's ideal, because this means we can use the **UpdateData** event handler to send the update – which is in our local **TWebClientDataSet** only – back to the **RAD Server (EMS) micro service**, so the update ends up in the actual database.

However.. if I do not click on the **Post** button of the **TWebDBNavigator,** but just click on the **Next** button to move to the next record, we get a different set of events. In that case, we get two **UpdateData** events, followed by **StateChange** (*back to browse mode*), **DataChange** and another **DataChange:**

```
UpdateData
UpdateData
StateChange
DataChange
DataChange
```

The second call to **UpdateData** may be too much, and we can take care of that by adding a protected **Boolean** field called **TableUpdating** to the web form, set to **True** when we enter the **UpdateData** method, and ensureing we do not re-enter the same method.

```
protected
  TableUpdating: Boolean;
```

We need to set if to False when the form is created:

```
procedure TForm1.WebFormCreate(Sender: TObject);
begin
 TableUpdating := False;
end;
```

The implementation of **UpdateData** now becomes as follows:

```
procedure TFormEmsDemo.WebClientDataSource1UpdateData(Sender: TObject);
begin
 if not TableUpdating then
 try
  TableUpdating := True;
  WebMemo1.Lines.Add('UpdateData');
  if WebClientDataSet1.State in dsEditModes then
  begin
   WebClientDataSet1.Post;
   WebMemo1.Lines.Add('Post');
  end;

 finally
  TableUpdating := False;
 end;
end;
```

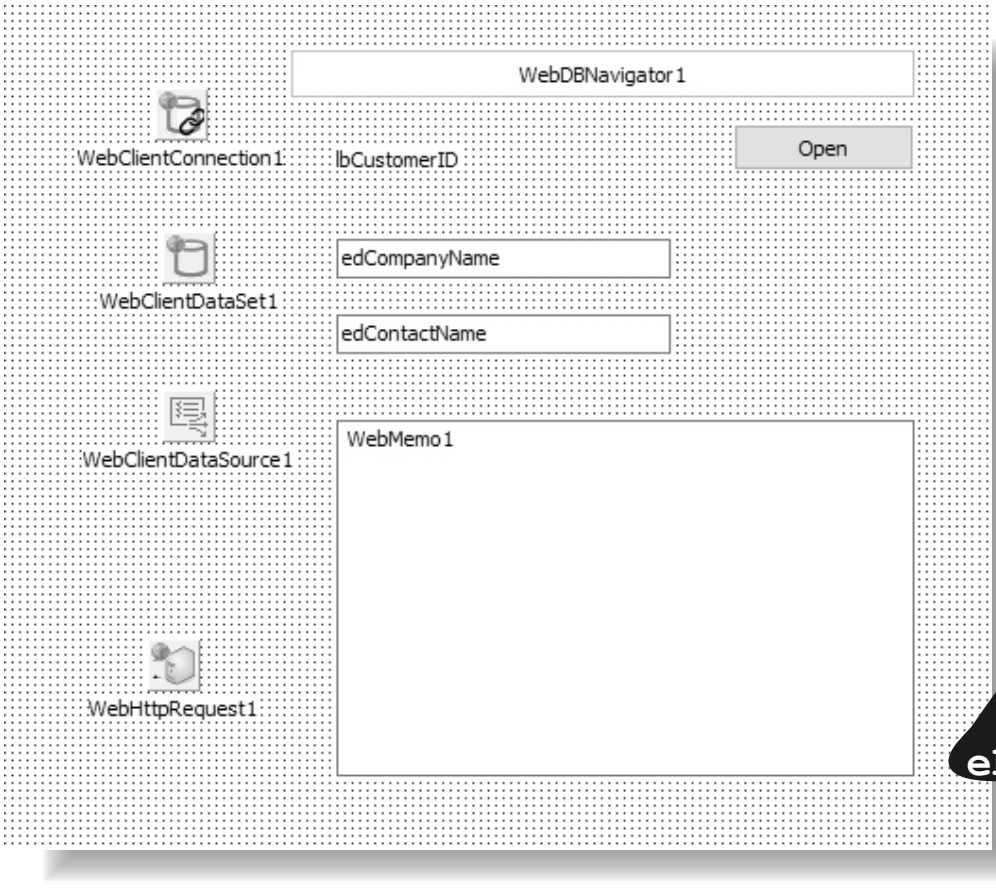
This will ensure that the update is written to the dataset, although not to the **REST Server,** yet.

```
URL := 'https://www.bobswart.nl/ems/emsserver.dll/WebCoreEMS/UpdateCustomer' +
  '/' + lbCustomerID.Caption +
  '/' + edCompanyName.Text +
  '/' + edContactName.Text;
```

TMS WEB REST

In order to send the update of the customer to the **RAD Server (EMS) micro service,** we must call the **WebCoreEmsDemo microservice** with the **POST** protocol, passing the **CustomerID, CompanyName** and **ContactName** as arguments.
An example URL would be as follows:

NOTE that we cannot test this **URL** in the browser, because we need to **POST** it, and inside the browser we can only send **GET** requests.
In order to send **Web REST** requests from within a **TMS WEB Core** application, we can use the **TWebHTTPRequest** component, so place one on the Web Form.



First, we need to ensure that we **POST** the **HTTP REST** request, so make sure to set the Command property to **httpPOST.**
Then, we can set the **URL** property to the one we defined on the previous page, and call the Execute method to send the **HTTP REST** request to the **RAD Server (EMS) micro service**. If we do this right after the **Post** of the **WebClientDataSet1,** then the entire **UpdateData** event handler becomes as follows: The example source code contains both the links to the **localhost:8000** version of the **RAD Server (EMS) micro service,** and the deployed version on my web server.
Feel free to make your own modifications, and let me know if you have any questions or comments.
For more information or support regarding Delphi, RAD Server (EMS) or TMS WEB Core, feel free to contact the author: **Bob Swart** from **Bob Swart Training & Consultancy (eBob42)** at **Bob@eBob42.com** or via **https://www.linkedin.com/in/drbob42**

```
procedure TForm1.WebClientDataSource1UpdateData(Sender: TObject);
begin
 if not TableUpdating then
 try
  TableUpdating := True;
  WebMemo1.Lines.Add('UpdateData');
  if WebClientDataSet1.State in dsEditModes then
  begin
   WebClientDataSet1.Post;
   WebMemo1.Lines.Add('Post');
  end;
  WebHttpRequest1.URL :=
   'https://www.bobswart.nl/ems/emsserver.dll/WebCoreEMS/UpdateCustomer' +
   '/' + lbCustomerID.Caption +
   '/' + edCompanyName.Text +
   '/' + edContactName.Text;
  WebMemo1.Lines.Add(WebHttpRequest1.URL);
  WebHttpRequest1.Execute;
 finally
  TableUpdating := False;
 end;
end;
```

DELPHI CONFERENCE 18 SEPTEMBER 2018

08:30-09:30 Welcome and registration with coffee and tea

09:30-10:30 **KEYNOTE: MARCO CANTÚ - PRODUCT MANAGER DELPHI - "DELPHI 10 FOR WINDOWS 10 AND BEYOND"**

In this technical keynote, Marco will cover the status of Delphi 10 and what's coming, with a particular focus on Windows 10 support, VCL development for Windows 10, but also covering Delphi mobile and server solutions and the overall industry trends the product is part of

10:30-11:30 **BRIAN LONG - CREATIVE DELPHI DEBUGGING TECHNIQUES**

Debugging represents a big part of development, perhaps one of the biggest. We all know about breakpoints, single-stepping and watches, but what else can we do to help work through bug scenarios and resolve problems?

This session looks at a number of techniques, tricks, and utilities to help make the chore of debugging a bit more productive. Warning: this session may contain the CPU window!

11:30-11:50 **Coffee Break / Go to Breakout Sessions**

11:50-12:40 **Brian Long**

HOW TO ACCESS THE ANDROID API

The ability to build Android applications is a great aspect of recent versions of Delphi, which gets more capable and functional with every release.However exploring outside the "FMX envelope" is still an onerous task to all but the most propeller-headed of Delphi developers.

We'll look at how to pull in various "not-in-the-box" features into an Android application using the latest version of Delphi and hopefully take away the mystery associated with it.

11:50-12:40 **Bruno Fierens**

A RADICALLY NEW WAY TO DEVELOP MODERN WEB APPLICATIONS

The all new TMS WEB Core product brings exciting new ways to create modern, fast and responsive web applications using the SPA model. This enables Delphi devs to use the familiar Delphi language and RAD development techniques to create web apps directly from the IDE. While TMS WEB Core facilitates creating the UI logic completely with Delphi using a Pascal to JavaScript compiler, the framework is extensible to consume popular JavaScript libraries and frameworks such as Bootstrap, jQuery, etc... TMS WEB Core also empowers Delphi developers to leverage the TMS FNC UI Controls framework as UI controls for web applications, reusing the VCL or FMX UI logic.

12:40-13:30 **Lunch - Go to Break Out Sessions**

13:30-14:20 **Roald van Doorn**

CONTINUOUS DELIVERY WITH EXISTING VCL APPLICATIONS

A case study of how we applied CD principles to an older VCL application. We will take a look at the challenges we faced and the solutions we chose, the frameworks we use, release procedures and feedback loops. We will demonstrate how we safely build and deploy the Windows software for Albelli en Vistaprint and the benefits this brings to our team and organization. Outline: automating your builds using TeamCity - Unit testing using DUnitX - Automated UI tests using Ranorex and Specflow - Deploy to different environments with ProGet and Octopus Deploy - Increase speed of value to customer (reduced stock) - Increased feedback to developers.

13:30-14:20 **Daan van der Werff**

DELPHI OP DE WERKVLOER "GROOTHANDEL & MAGAZIJN"

Tijdens deze sessie krijgt u een kijkje onder de motorkap van een groothandel waar kritische processen gemaakt zijn in Delphi. Deze zijn verantwoordelijk voor een omzet van ca 31 miljoen! Van data connectoren tot orders, microservices, mobile en cross platform ontwikkelingen voor warehouse management systemen en meer!

https://www.barnsten.com/default/events/details?events_id=327

DEVELOP YOUR FUTURE

Delphi Conference 2018
Jaarbeurs Utrecht
Netherlands

14:20-14:30 Go to next Break Out Sessions

14:30-15:20 Danny Wind

MICRO SERVICES AND PROGRESSIVE WEB APPS (PWA) DELPHI

In this session we'll showcase a lightweight REST microservice and a (progressive) web app, as well as an Android/iOS App and a desktop application all crated in Delphi. With the techniques in this session you'll be able to leverage these new technologies in your own projects. Just re-use the sources and you're ready to go.

14:30-15:20 Bob Swart

DELPHI EN FIREDAC ENTERPRISE CONNECTORS

De FireDAC Enterprise Connectors stellen Delphi ontwikkelaars in staat om externe data bronnen beschikbaar te maken als (FireDAC) tables en queries, voor gebruik en verwerking met FireDAC data-access componenten. In deze sessie zal Bob de algemene werking van de FireDAC Enterprise Connectors laten zien, met veel code voorbeelden, en daarbij een aantal specifieke toepassingen demonstreren met externe bronnen zoals bijvoorbeeld Facebook, Twitter, LinkedIn maar ook Gmail, Google Drive, Google Analytics en een generieke REST en JSON connectie.

eBob42.com

15:20-15:40 Break

15:40-16:30 André Mussche

DE OPKOMST VAN SPRAAKHERKENNING

André werkt momenteel met het nieuwe realtime en streaming protocol gRPC dat vrij recent door Google is ontwikkeld. gRPC wordt bijvoorbeeld gebruikt bij Blockchain implementaties zoals hyperledger, maar is ook uitermate geschikt voor de toepassing in projecten met spraakherkenning. Het gebruik van spraakherkenning in applicaties wordt steeds meer toegepast en wordt bijvoorbeeld in ziekenhuis applicaties veel gebruikt. Maar ook in ERP systemen wordt dit steeds vaker toegepast. In deze sessie krijgt u te zien hoe u met dit communicatieprotocol een extra dimensie kunt toevoegen aan uw applicatie met het door André ontwikkelde protocol voor Delphi toepassingen dat inmiddels ook als open source beschikbaar is.

15:40-16:30 Marco Cantú

RAD SERVER IN DEPTH

This session offers a deeper look into the development of REST + JSON web services with RAD Server, going beyond the basic marketing information and introductory demos, and highlighting some advanced features like dynamic resources and custom login modules, recent web and JavaScript support additions, touching on ExtJS clients, and providing indication of new coming features.

16:30-17:00 Closing - Q & A - Prize Draw

https://www.barnsten.com/default/events/details?events_id=327

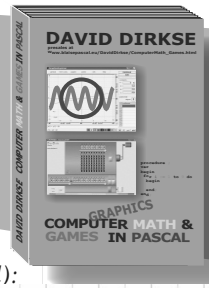
DEVELOP YOUR FUTURE

Delphi Conference 2018
Jaarbeurs Utrecht
Netherlands

EXAMPLES OF RECURSION
BY DAVID DIRKSE

All programcode available in **Delphi 7** and **Delphi Tokyo 2.3**

starter — expert



There are several ways to repeat code in computer programs. One way is code repetion as done by for, while or repeat..until loops. Another way are functions and procedures. Extra for functions and procedures is the possibility to call themselve, this is named recursion.

Recursion in computer science is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem (as opposed to iteration). The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

„The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions."

Most computer programming languages support recursion by allowing a function to call itself from within its own code. Some functional programming languages do not define any looping constructs but rely solely on recursion to repeatedly call code. Computability theory proves that these recursive-only languages are Turing complete; they are as computationally powerful as Turing complete imperative languages, meaning they can solve the same kinds of problems as imperative languages even without iterative control structures such as "while" and "for".

A recursive process looks like this *(blue boxes are identical)*:

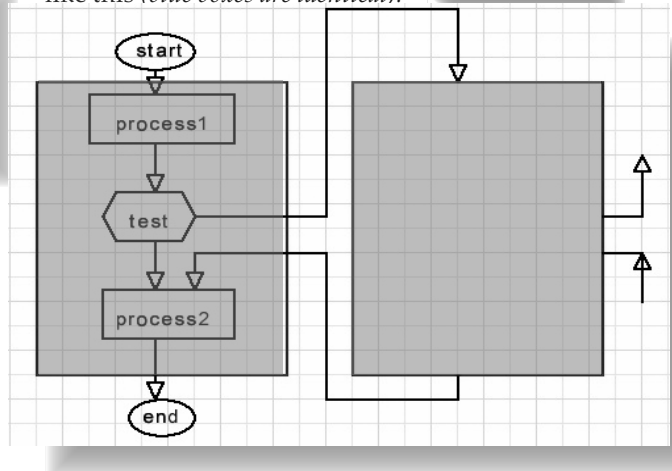
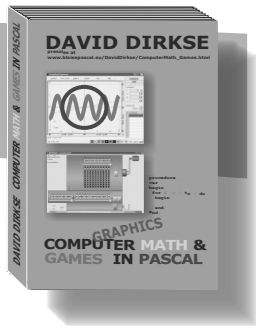


Figure 1:Diagram 1

After some **process 1** a test is made to call the same procedure or function again or to continue with **process 2**. After finishing a number of processes1 the same number of processes2 takes place and the procedure ends.
Recursive procedures are very powerful but hard to understand. For clarity it helps to consider a recursive procedure as many different procedures performing the same thing.
That's what the diagram1 above illustrates

Figure2:
below: A nice example is the old Droste effect.

Designer: Jan Misset Uploader: Alf van Beem



Recursion only works for parameters and locally declared variables because they are placed on the stack and are unique for each call.

EXAMPLE 1

The calculation of N!

(N faculty = N(N-1)(N-2)....3.2.1

```
function NfacR(n : byte) : dword;
//calculate n! in recursive way
begin
 if n = 1 then result := 1
  else result := NfacR(n-1);
 result := result * n;
end;
```

Process 1 sets the result to **1 if n=1** else calls itself. **Process 2** multiplies by **n..**

EXAMPLE2

Here I program **the so called Pythagoras tree,** which is constructed of squares and triangles.

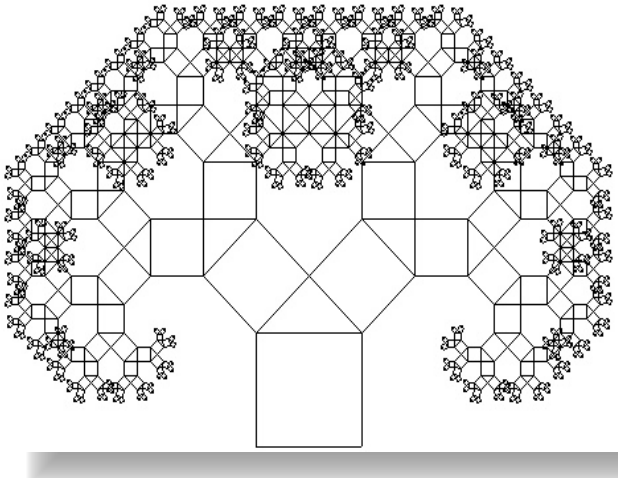


Figure 2 This is done by example app bp-pythtree_Delphi 7 or bp-pythtree_Tokyo_2_3 or bp-pythtree_LAZARUS. You can see this in slow motion

THEORY

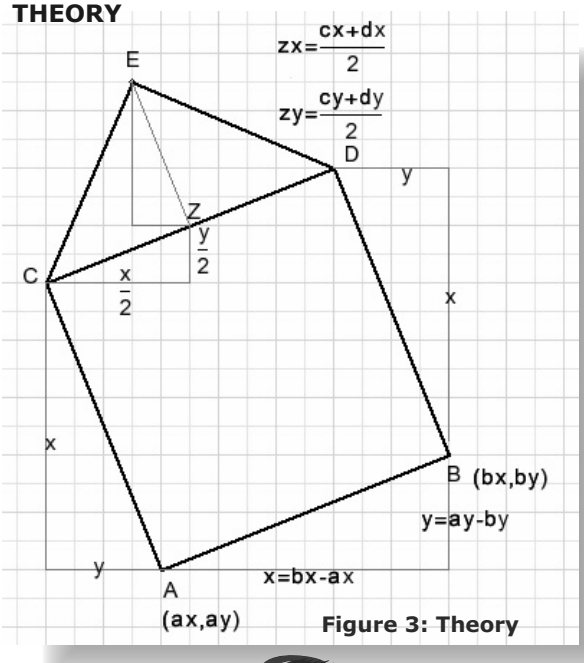


Figure 3: Theory

Start is by line **AB** and the coordinates of **A** and **B**. Then points **C,D,E** are calculated and lines **AB,BD,CD,CE,DE** drawn. Then the procedure calls itself twice: first with **CE,** then with **DE. A** depth counter controls the number of calls. **This is the procedure:**

```
procedure makeTree(ax,ay,bx,by : smallInt; depth : byte);
//paint part of tree
var cx,cy,dx,dy,ex,ey : smallInt;
  x,y,zx,zy : smallInt;
begin
 x := bx-ax;
 y := ay-by;
 cx := ax-y;
 cy := ay-x;
 dx := bx-y;
 dy := by-x;
 zx := (cx + dx) div 2;
 zy := (cy + dy) div 2;
 ex := zx - (y div 2);
 ey := zy - (x div 2);
 with form1.PaintBox1.Canvas do
 begin
  pen.color := $000000;
  pen.width := 1;
  moveto(ax,ay);
  lineto(cx,cy);
  lineto(dx,dy);
  lineto(bx,by);
  moveto(cx,cy);
  lineto(ex,ey);
  lineto(dx,dy);
  if depth > 1 then
  begin
   maketree(cx,cy,ex,ey,depth-1);
   maketree(ex,ey,dx,dy,depth-1);
  end;
 end;
end;
```

NOTE: ax,ay are the coordinates of point **A**, etc. We see that now process1 does all the work.

EXAMPLE 3

Square root calculation.

Here we apply the properties of chained fractions which have the general form:

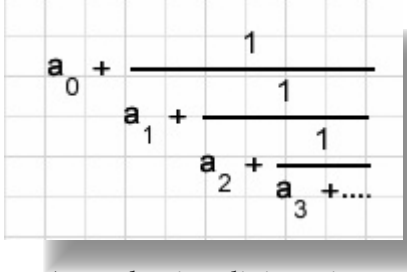
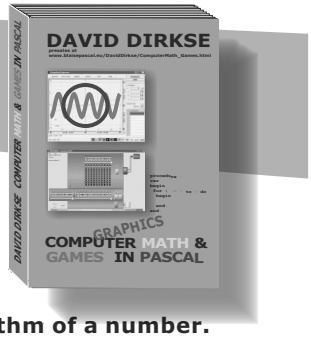


Figure 4: Example

A number is split in an integer part **(a)** and **a fraction < 1.**

Then the **reciproke** of the **fraction** is taken which is **> 1** and again the integer part is stripped off.



THEORY:

$$N - a^2 = (\sqrt{N} - a)(\sqrt{N} + a)$$

$$\sqrt{N} - a = \frac{N - a^2}{a + \sqrt{N}} \qquad \text{als } N - a^2 = r \ldots$$

$$\sqrt{N} = a + \frac{r}{a + \sqrt{N}} = a + \frac{r}{a + a + \frac{r}{a + \sqrt{N}}} \ldots\ldots$$

Figure 5: Recursion

The recursion is clearly visible.
When root **N** descents in the denominators, it's influence decreases and the answer becomes more accurate.

Before the recursive process starts, a and r have to be calculated which are constant during root calculation. A is the number where the square just fits **N** so **a2 < N<(a+1)2**

```
var A,N,R : dword;
.....
procedure TForm1.Button2Click(Sender: TObject);
// calculate root recursive way
// N : number  declared outside
// R : N-A*A   ....
var i,c : byte;   root : double;
begin
 try
  N := strtoint(edit1.text);
 except
  N := 1;
  edit1.Text := '1';
 end;
 c := 0;
 if N > 0 then begin
  for i := 0 to 31 do   // find largest 1 bit square
   if ((1 shl i) and N) <> 0 then c := i;
  c := c shr 1; // 2 bit multiple
  A := 1 shl c; // largest single bit root
  R := N - (A shl c);
  root := RecROOT(10); //depth = 10
 end
 else root := 0;
 statictext2.Caption :=
FormatFloat('#0.0#####',root);
end;
```

This is the recursive square root function:

```
function RecROOT(i : byte) : double;
// recursive root claculation
begin
 if i >1 then result := A + R/(A + RecROOT(i-1))
  else result := A + R/(2*A);
end;
```

NOTE that the only parameter is the depth control. The function result is the root value.

EXAMPLE 4.
Calculation of the logarithm of a number.
To refesh your knowledge:

$${}^g\log(a) = x \quad \Leftrightarrow \quad g^x = a$$

$${}^g\log(a^n) = n.{}^g\log(a)$$

$${}^g\log(ab) = {}^g\log(a) + {}^g\log(b)$$

$${}^a\log(b) = \frac{{}^g\log(b)}{{}^g\log(a)} \qquad \text{...so also:}$$

$${}^a\log(b) = \frac{1}{{}^b\log(a)}$$

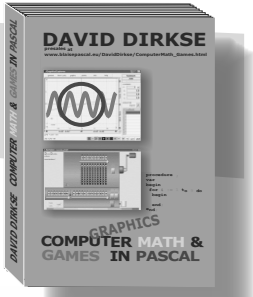
Figure 6: Calculation of the logarithm of a number.

In mathematics, the **logarithm** is the **inverse** function to **exponentiation.** That means the logarithm of a given number **x** is the exponent to which another fixed number, the base **x,** must be raised, to produce that number **x.**
In the simplest case the logarithm counts repeated multiplication of the same factor;
e.g., since $1000 = 10 \times 10 \times 10 = 10^3$, the "logarithm to base 10" of 1000 is 3.
The logarithm of **x** to base b is denoted as ${}^b\log(x)$ (*or, without parentheses, as* ${}^b\log x$, *or even without explicit base as* $\log x$, *when no confusion is possible*).

More generally, exponentiation allows any positive real number to be raised to any real power, always producing a positive result, so the logarithm for any two positive real numbers **b** and **x** where **b** is not equal to 1, **is always a unique real number y**.
More explicitly, the defining relation between exponentiation and logarithm is:

${}^b\log x = y$ exactly if $b^y = x$.

For example, ${}^2\log 64 = 6$, as $64 = 2^6$.
The logarithm to **base 10 (that is b = 10)** is called the common logarithm and has many applications in science and engineering. The natural logarithm has the number e (that is $b \approx 2.718$) as its base; its use is widespread in mathematics and physics, because of its simpler derivative. The binary logarithm uses **base 2 (that is b = 2)** and is commonly used in computer science.



WIKIPEDIA

Logarithms were introduced by **John Napier** in the early 17th century as a means to simplify calculations. They were rapidly adopted by navigators, scientists, engineers, and others to perform computations more easily, using slide rules and logarithm tables. Tedious multi-digit multiplication steps can be replaced by table look-ups and simpler addition because of the Fact —important in its own right—that the logarithm of a product is the sum of the logarithms of the factors:
$^{b}\log(xy) = {}^{b}\log x + {}^{b}\log y$,
$^{2}\log(16*4) = {}^{2}\log 16 + {}^{2}\log 4 = 4+2$,
provided that b, x and y are all positive and $b \neq 1$.
The present-day notion of logarithms comes from Leonhard Euler, who connected them to the exponential function in the 18th century.

Taking the logarithm of a number is imagining that number as a base number powered to an exponent.

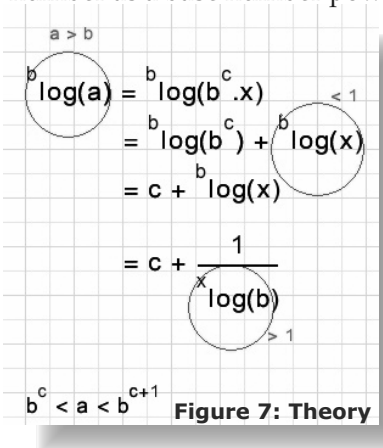


Figure 7: Theory

Then the base is removed, leaving the exponent. Again we use chained fractions.
Theory:

c is the highest exponent of b which fits a. The c terms make the chained fraction.

At the start we suppose a > b and in the calculations the biggest powers of b are stripped off from a. Pressing a GO button starts the process. First the variables a and b must be prepaired:

```
procedure TForm1.goBtnClick(Sender: TObject);
// calculate logarithm
// base: edit1;    number : edit2
var a,b,x : double;   OK : boolean;
begin
   activecontrol := edit2;
   a := 0; b := 0;
   if length(edit1.Text) = 0 then edit1.Text := '10';
   if length(edit2.Text) = 0 then edit2.Text := '1';
   try
      b := strtofloat(edit1.Text);
      a := strtofloat(edit2.Text);
      OK := (b > 1) and (a >= 1);
   except
      OK := false;
   end;
      if OK then  begin
         x := RecLog(b,a); // recursive log process
         statictext1.Caption :=
                  formatfloat('#0.########',x);
      end else resetBtnClick(self);
end;
```

The recursive log procedure:

```
function recLog(b,a : double) : double;
//recursive logarithm calculation
//b : base; a : number
var c : byte;  b2,bt : double; GO : boolean;
begin
   c := 0;
   b2 := 1;
 repeat
   bt := b2 * b;
   GO := a >= bt;
   if GO then begin
      inc(c);
      b2 := b2 * b;
   end;
   until GO = false;
   a := a/b2;
   if a > 1.000001 then result := c + 1/reclog(a,b)
 else result := c;
end;
```

In the process variable a becomes closer to 1 and this is the test to end the recursion. To end this article please look at a result:
On the next pages you will find the Images of the various pascal versions: Delphi Tokyo and Lazarus

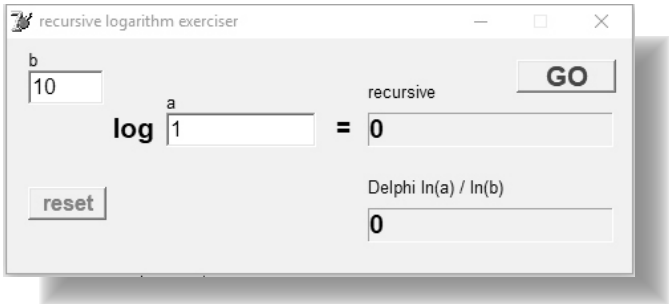


Figure 8: This is the original Delphi 7 projectfile bp-logs-D7.zip.
Other available projects:
bp-logs-Tokyo_2_3
bp-logs-LAZARUS

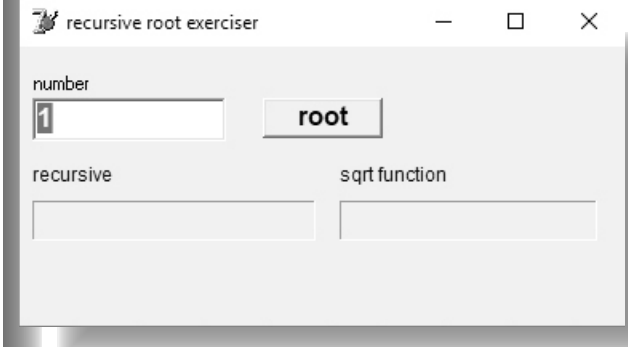
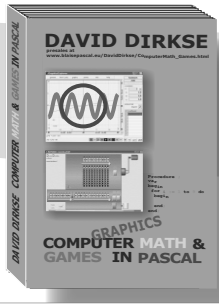


Figure 9: This is the original Delphi 7 projectfile bp-roots-D7.zip.
Other available projects:
bp-roots-Tokyo_2_3
bp-roots-LAZARUS



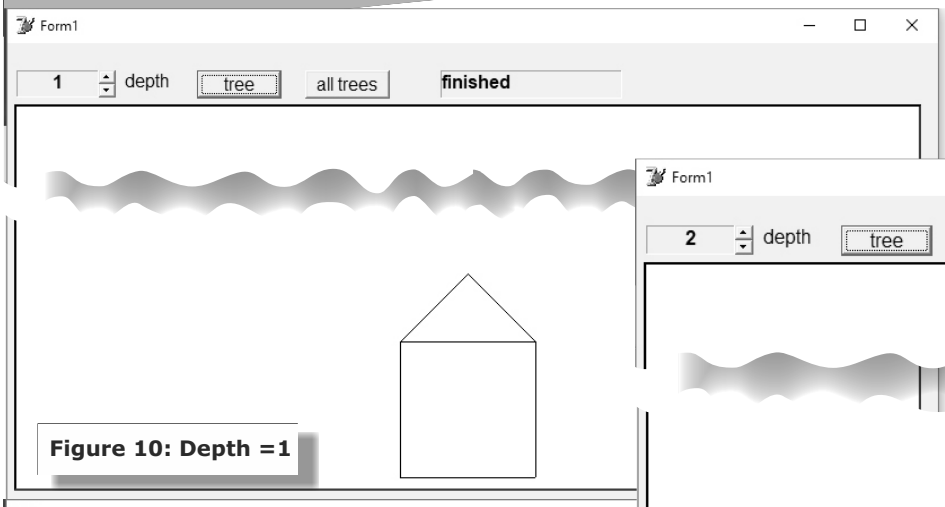


Figure 10: Depth =1

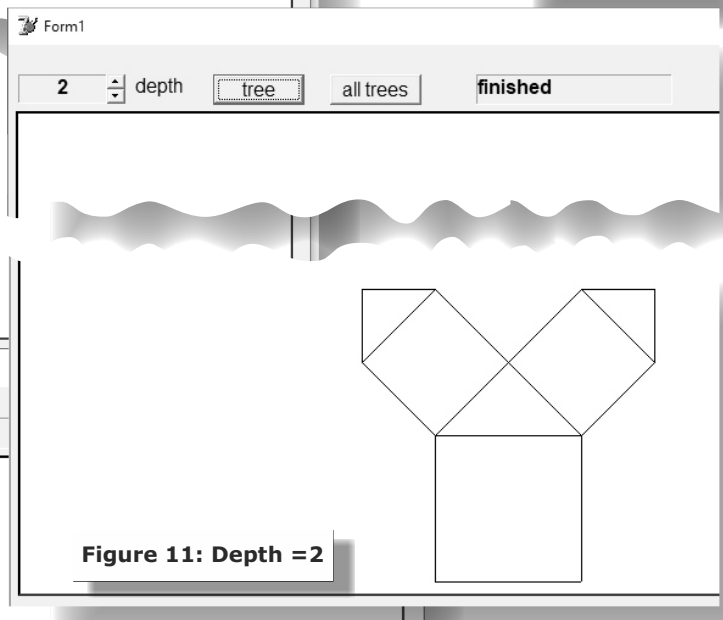


Figure 11: Depth =2

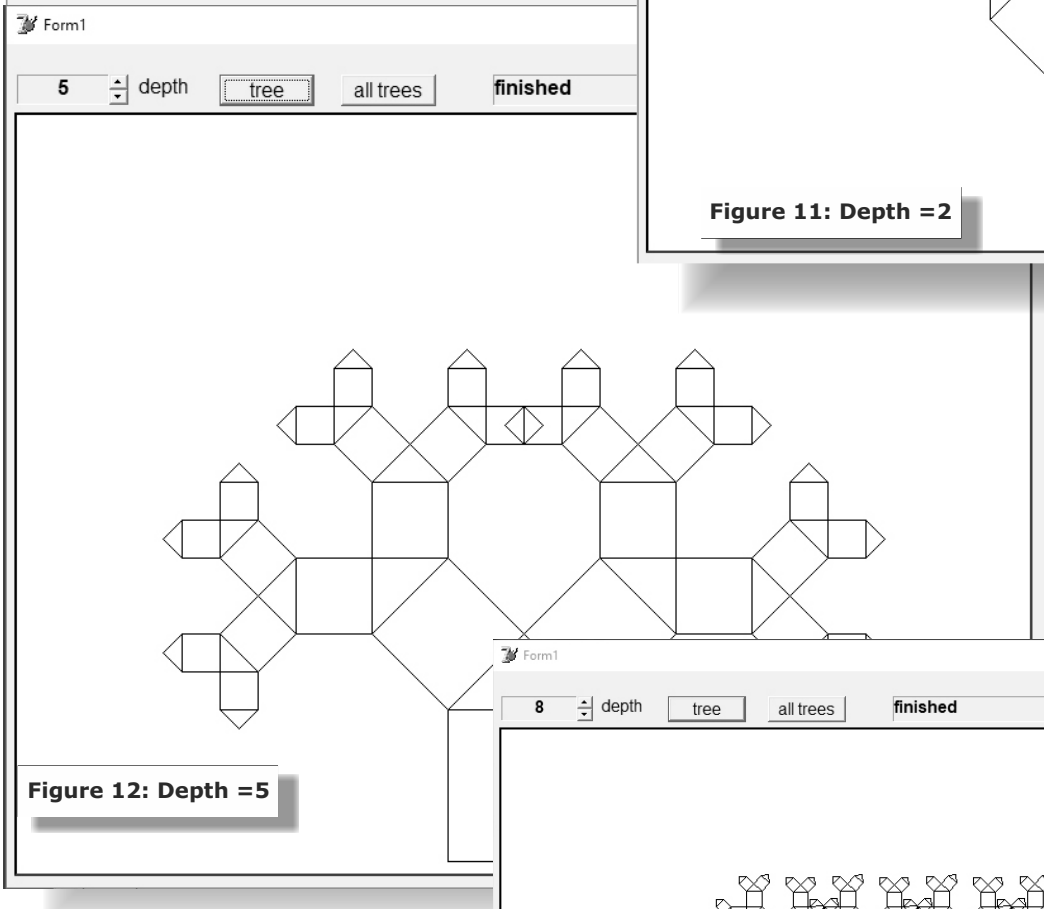


Figure 12: Depth =5

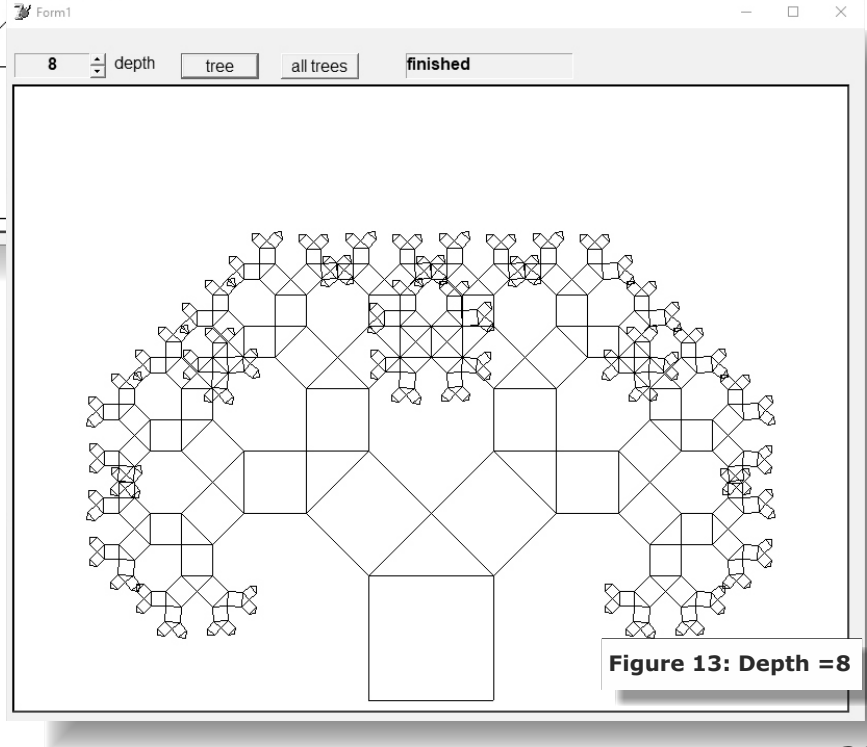
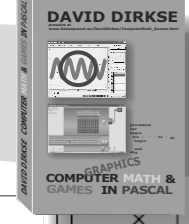


Figure 13: Depth =8

Figure 14:
This is the original Delphi 7 projectfile
bp-pythtree_D7
Other available projects:
bp-pythtree_LAZARUS



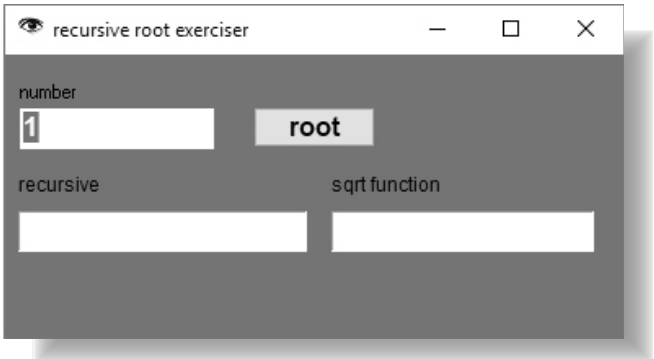


Figure 15: Recursive Root Execiser for Delphi Tokyo 2.3

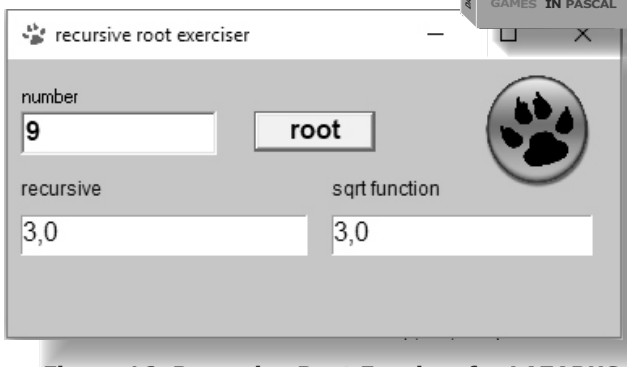


Figure 16: Recursive Root Execiser for LAZARUS

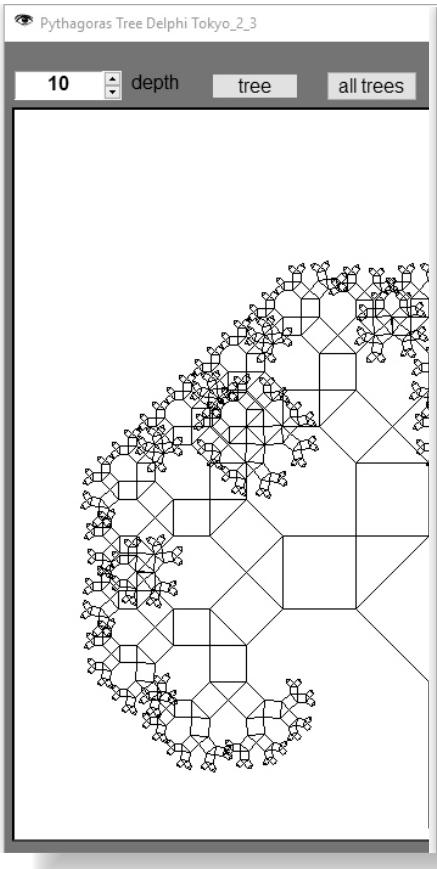


Figure 17: Pythagoras Tree for Delphi Tokyo 2.3

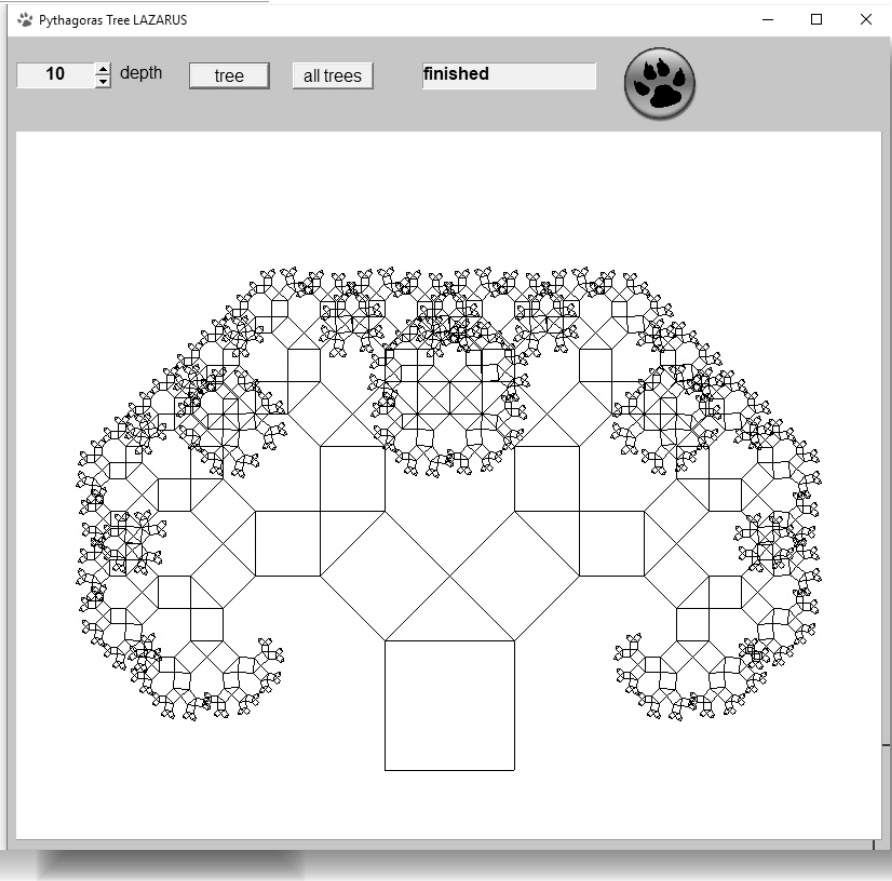


Figure 18: Pythagoras Tree for LAZARUS

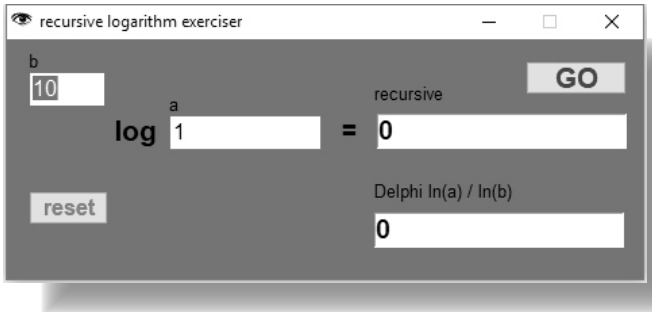


Figure 18: Recursive Loagarithm Exerciser for Delphi Tokyo 2.3

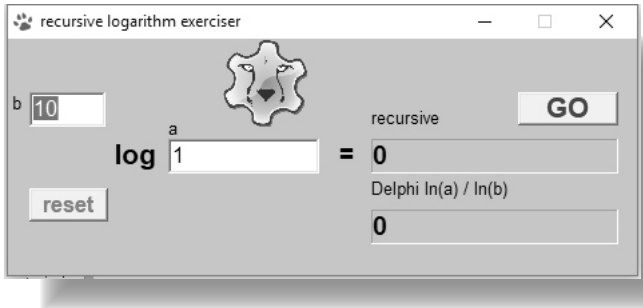


Figure 18: Recursive Loagarithm Exerciser for LAZARUS

LAZARUS PROFESSIONAL CONFERENCE

KÖLN/BONN

THURSDAY 20
FRIDAY 21
SATURDAY 22
SEPTEMBER 2018

IBEXPERT AND BLAISE PASCAL MAGAZINE

will organize in cooperation with LAZARUS FOUNDATION and LAZARUS FACTORY THE first

LAZARUS PROFESSIONAL CONFERENCE:

September 20 (Thursday) 21(Friday) 22 (Saturday) 2018.

ADDRESS:

Gustav-Stresemann-Institut e.V., Langer Grabenweg 68, DE-53175 Bonn

LOCATION AND APPROACH:

Langer Grabenweg 68 D-53175 Bonn

BY RAIL:

Between **Bonn central station** and Bonn-Bad Godesberg trams commute every 7 minutes (tram-number 16 and 63)

From **Bonn central-station:** U-Bahn Line (underground/tram number) 16 or 63, direction Bad Godesberg, leave tram at "Max-Löbner-Straße" - walk down "Max-Löbner-Strasse" to the end.

From **ICE-Station** Siegburg / Bonn: U/tram-line 66, direction Bonn/Bad Honnef - leave tram at station "Robert-Schuman-Platz" Kurt-Georg-Kiesinger-Allee, turn left to Jean-Monet-Straße, turn left to Heinemann-Straße

BY PLANE:

From airport Cologne / Bonn: Bus No. SB 60 until Hauptbahnhof (Bonn central station)
From Bonn central-station: take U-Bahn Line (underground/tram number) 16 or 63, direction Bad Godesberg. Leave at "Max-Löbner-Strasse" - Walk down "Max-Löbner-Strasse" to the end

ALLES Vorträge werden auf Deutsch und Englischer Sprache ausgetragen

Workshop Themen 20.-21.09.2018

Donnerstag 20 September 2018

Der Lazarus Desktop
Die wichtigsten Projekteinstellungen
Konvertierung von Delphi Quelltexten
Konvertierung Tips und Tricks
Sprachunterschiede Delphi Lazarus fpc

Mittagessen für alle Teilnehmer

Online Package Manager
Komponenten für Lazarus
Report Lösungen
Online Ressourcen:
Die wichtigsten Tools, Foren und Webseiten
Debugger und Laufzeitmessungen

Abendessen für alle Teilnehmer

"Beer and Best Practice Unconference"
Die Teilnehmer entscheiden gemeinsam, wer welches Thema vortragen soll
Freibier und freie Softwaredrinks inklusive
Der Tagungsraum steht bis Mitternacht zur Verfügung

Freitag 20 September 2018

Datenbankzugriff mit SQLDB und Alternativen
Installation und erste Schritte unter Windows, Linux und Mac
Installation pas2js und erste Schritte zur Mobilen Anwendung auf iOS und Android Distribution von pas2js Anwendungen
Lazarus und Firebird Datenbanken

Mittagessen für alle Teilnehmer

Softwarearchitektur für kundenspezifische Anwendungen
Business Logik, Multitier, Datenbank Prozeduren
Chromium Integration in Fat Client Anwendungen
TMS Webcore für Lazarus

„Beer and Best View" Die Teilnehmer gehen gemeinsam zu einem Biergarten an den Rheinwiesen, sofern es das Wetter erlaubt. Abendessen und Getränke auf eigene Rechnung

Interessierte und Teilnehmer am Community Day sind herzlich eingeladen
Open End

Samstag Community Day 22.09.2018

Lazarus und fpc im Überblick: Roadmap
Der Umstieg von Delphi auf Lazarus
Komponenten: Fluch oder Segen

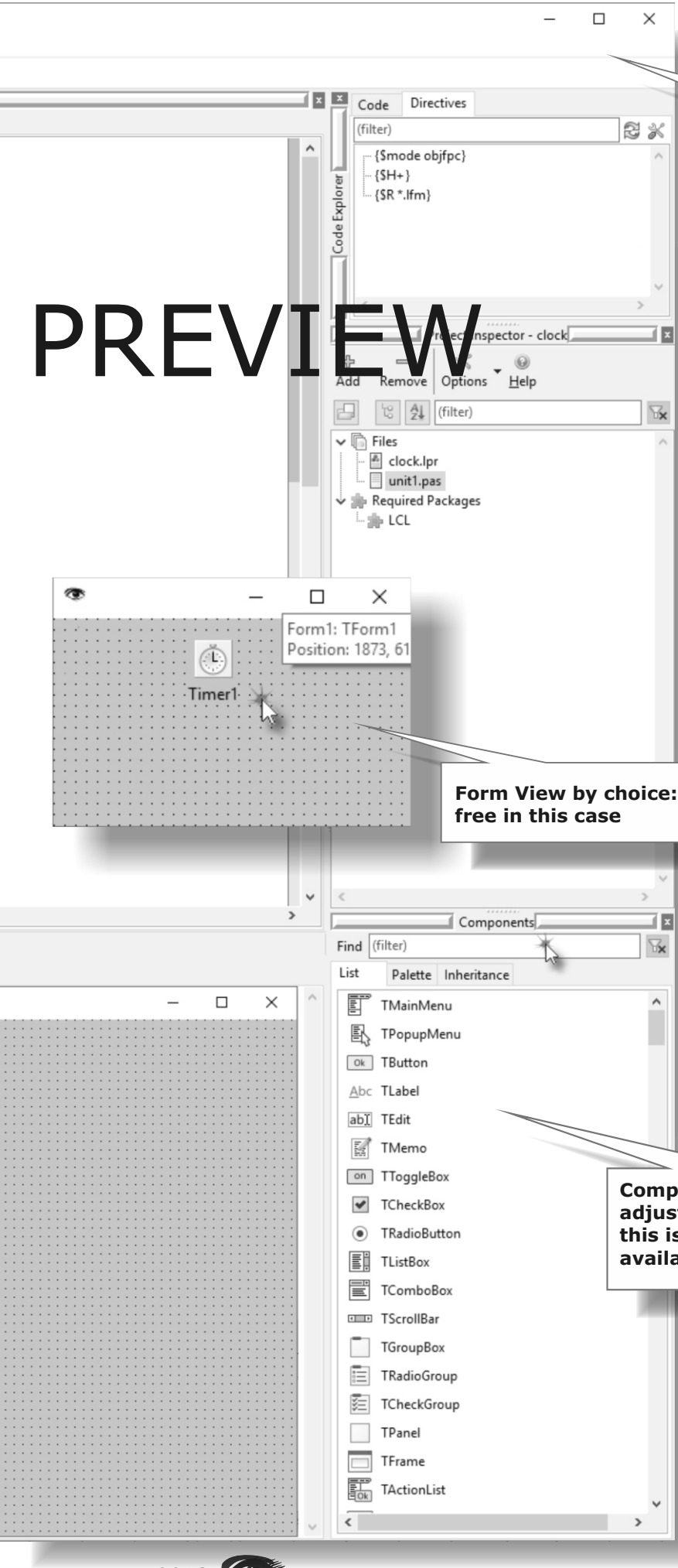
Mittagessen für alle Teilnehmer

Echte Multiplattform Anwendungen und Besonderheiten
Windows / Linux / Mac
Pas2js iOS/Android
Offene Diskussion: Was fehlt euch noch in Lazarus (oder was kennt ihr nur noch nicht)
Aushang: Mitarbeiter gesucht, Firmen suchen Pascal Entwickler, Pascal Entwickler suchen Arbeitgeber

Im Ticketpreis sind Getränke, Kaffee und ein Mittagessen, jedoch keine Übernachtungskosten enthalten. Die meisten Sessions werden in deutscher Sprache durchgeführt, wir übersetzen natürlich auch auf English und Holländisch da wir genug Mitarbeiter in diesen sprachen haben.

Anmeldung hier:
https://www.lazprocon.net/anmeldung


Lazarus IDE v1.8.2 - clock
File Edit Search View Source Project Run Package Tools Window Help
Object Inspector
Components (filter)
Form1: TForm1
Label1: TLabel
Label2: TLabel
Timer1: TTimer
Properties (filter)
Properties Events Favorites Restricted
Action
ActiveControl
Align alNone
AllowDropFiles (False)
AlphaBlend (False)
AlphaBlendValu 255
Anchors [akTop,akLeft]
AutoScroll (False)
AutoSize (False)
BiDiMode bdLeftToRight
BorderIcons [biSystemMenu,biMinimize
BorderStyle bsNone
BorderWidth 0
Caption
ChildSizing (TControlChildSizing)
Color clSkyBlue
Constraints (TSizeConstraints)
Cursor crDefault
DefaultMonitor dmActiveForm
DesignTimePPI 96
DockSite (False)
DragKind dkDrag
DragMode dmManual
Enabled (True)
Font (TFont)
FormStyle fsNormal
Height 477
HelpContext 0
HelpFile
HelpKeyword
HelpType htContext
Hint
HorzScrollBar (TControlScrollBar)
Icon (TIcon)
KeyPreview (False)
Left 768
Menu
Name Form1
ParentBiDiMode (True)
ParentFont (False)
The form's background color.
The background color of the control.
TForm.Color:TColor = TGraphicsColor = -$7FFFFFFF-
*Unit1 clock
The component list was moved by choice to the right bottom
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
ExtCtrls, IniFiles;
type
{ TForm1 }
TForm1 = class(TForm)
Label1: TLabel;
Label2: TLabel;
Timer1: TTimer;
procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
procedure FormCreate(Sender: TObject);
procedure Label1Click(Sender: TObject);
procedure Label2Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
public
end;
var
Form1: TForm1;
appINI : TIniFile;
implementation
{$R *.lfm}
DOCKING
Devider
13: 22 Modified INS F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\Lazarus\unit1.pas
Formview Watch List Search Results Assembler BreakPoints
Devider
Timer1
Form View by choice: embedded in this case
Free Pascal
Lazarus
Project
Version #: 1.9.0
Date: 2018-06-14
FPC Version: 3.0.4
SVN Revision:
i386-win32-win32/win64
Write Once
Compile Anywhere



PREVIEW

Note Book View or so called compact view

Form View by choice: free in this case

LAZARUS PROFESSIONAL CONFERENCE KÖLN/BONN

Component pallete, adjustable, you can query it, this is of course also available in normal view

In the next issue several other new features will be shown

BY KIM MADSEN

starter expert DX Delphi

The ORM in kbmMW continues to evolve. The upcoming release contains some new nice features that makes it easy to fetch adjacent data from other joined tables, but all defined by Delphi model style classes.
To illustrate the new features, let's think that we have a small bank, and we want to keep track of people and their accounts in any database of our choosing.

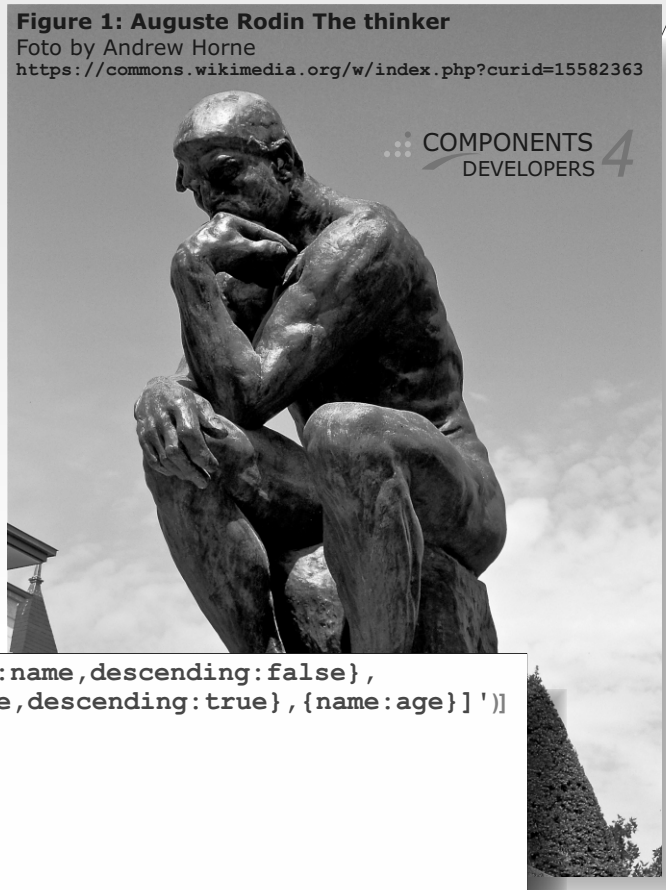


Figure 1: Auguste Rodin The thinker
Foto by Andrew Horne
https://commons.wikimedia.org/w/index.php?curid=15582363

We represent a person by the TPerson Delphi class, and an account by the TAccount Delphi class. People are represented as TObjectList<TPerson> and multiple accounts are represented as TObjectList<TAccount>.

The TPerson class could be defined like this:

```
[kbmMW_Table('name:person, index:{name:i1,field:name,descending:false},
 index:{name:i2,unique:true,fields:[{name:name,descending:true},{name:age}]')]
TPerson = class
private
  FID:kbmMWNullable<string>;
  FName:kbmMWNullable<string>;
  FAddress:kbmMWNullable<string>;
  FAge:kbmMWNullable<integer>;
public
  [kbmMW_Field('name:id, primary:true, generator:shortGuid',ftString,40)]
  property ID:kbmMWNullable<string> read FID write FID;

  [kbmMW_Field('name:name',ftWideString,50)]
  property FullName:kbmMWNullable<string> read FName write FName;

  [kbmMW_Field('name:address',ftWideString,50)]
  property Address:kbmMWNullable<string> read FAddress write FAddress;

  [kbmMW_Field('name:age',ftInteger)]
  property Age:kbmMWNullable<integer> read FAge write FAge;
end;
```

As you notice there are attributes like kbmMW_Table and kbmMW_Field here. Those attributes governs how the Delphi class will be mapped towards a datastorage. As a side note its possible to add other attributes like kbmMW_Root, kbmMW_Child, kbmMW_Element and kbmMW_Attribute attributes, which governs how contents of the class is streamed/unstreamed when converting it to and from JSON, XML , YAML, BSON, MessagePack and CSV (also new in this release… a separate blog post may talk about that) which all are formats supported natively by kbmMW. The above uses of the kbmMW_Table and kbmMW_Field attributes has been covered in several previous blog posts. For that reason I will only go into details with what's new.

```
  [kbmMW_Table('name:account')]
  TAccount = class
  private
    FID:kbmMWNullable<string>;
    FPersonID:string;
    FName:kbmMWNullable<string>;
    FValue:double;
  public
    [kbmMW_Field('name:id, primary:true,
generator:shortGuid',ftString,40)]
    [kbmMW_NotNull]
    property ID:kbmMWNullable<string> read FID write FID;

    [kbmMW_Field('name:personid',ftString,40)]
    [kbmMW_NotNull]
    [kbmMW_Null('')]
    property PID:string read FPersonID write FPersonID;

    [kbmMW_Field('name:name, default:"Unknown"',ftString,30)]
    [kbmMW_NotNull]
    property Name:kbmMWNullable<string> read FName write FName;

    [kbmMW_Field]
    [kbmMW_Null(Math.NaN)]
    property Value:double read FValue write FValue;
  end;
```

WIKIPEDIA *In object oriented programming, data-management tasks act on object-oriented (OO) objects that are almost always non-scalar values.*

For example, an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on.

The address-book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as SQL database management systems (DBMS) can only store and manipulate scalar values such as integers and strings organized within tables.

The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping implements the first approach. The heart of the problem involves translating the logical representation of the objects into an atomized form that is capable of being stored in the database while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent.

This one is also pretty standard **kbmMW ORM** definition stuff. Only one minor thing (*which is actually not new*) is the use of a default value for the Name field.The reason for that, is that this **Account** class has evolved over time. In the previous versions, there was no Name property.
Many account records may already have been added to the datastorage.
Since I now added the Name property, and declared it should not accept Null values, I will have to tell **kbmMW** what to do with the already existing records in the database.
Using the Upgrade Table or **CreateOrUpgradeTable** methods, **kbmMW** will automatically figure out how to change the ➜ datastorage to now also accept and store the new Name property.
However the old records needs updating to conform with the Not null constraint defined, and for that purpose, the default value is used.
The value property is also (for sample purpose) declared differently, in the sense that it will interpret a value of Math.NaN as a Null value, instead of using the kbmMWNullable generics construction used by the other properties.
We can query for all people in the datastorage or a selection of them like this:

```
procedure TForm6.btnQueryList2Click(Sender: TObject);
var
  o:TObjectList<TPerson>;
begin
   o:=orm.QueryList<TPerson>;
   // Here we have a complete list of all people in the datastorage.
   o.Free;
end;
```

Or a selection:

```
procedure TForm6.btnQueryOne2Click(Sender: TObject);
var
  o:TPerson;
begin
   o:=orm.Query<TPerson>(['FullName'],['%IM%'],mwoqoLike);
   if o=nil then
     raise Exception.Create('Not found');
   // We found at least one person containing IM in the FullName property.
   // Only the first one found is returned since we have not asked for
   // a list.
   o.Free;
end;
```

But what if we would like to return a person along with the person's accounts?
It can be done in a number of ways. Now I will show how to do it in a way, where we can keep the original classes as is, and make a new class which contains both the person information and a list of accounts.

```
  [kbmMW_VirtualTable(TPerson)]
  TPersonWithAccounts = class(TPerson)
  private
    FAccounts:TObjectList<TAccount>;
  public
    destructor Destroy; override;

    [kbmMW_VirtualField('name:accounts, source:uData.TAccount,
key:ID, sourceKey:PID')]
    property Accounts:TObjectList<TAccount> read FAccounts write FAccounts;
  end;
...

destructor TPersonWithAccounts.Destroy;
begin
   FAccounts.Free;
   inherited;
end;
```

What I have done is define a virtual table.
A virtual table can be used for any type of ORM operations except being base for datastorage definition.
Hence calling **CreateTable(TPersonWithAccounts)** will raise an exception, since we have declared that **TPersonWithAccounts** as a virtual table. It is only used for representing its based table **(TPerson)** in a different way. Basically we augment the **TPerson** class with additional information.
You can read more about augmented data structures in one of the previous articles.

What's even more interesting is that there is an Accounts property which can contain a list of TAccount, and that property has been declared as being a virtual field by using the **kbmMW_VirtualField** attribute. Similar to a virtual table a virtual field can't exist in a datastorage. It is only for internal use by the developer. Since we already defined the table as virtual, we could have chosen to just use the **kbmMW_Field** attribute instead of the **kbmMW_VirtualField** attribute, since the field would anyway never materialize itself into a field in a datastorage.
However it's good practice to do as I have shown. The new bits here is that the **kbmMW_Field** and **kbmMW_VirtualField** attributes now also understands a source, key, sourceKey and optionally a value setting. I'll talk about the value setting later.

The source setting refers to the fully scoped **TAccount** class. Fully scoped means that you need to tell which unit it was defined in. **kbmMW** requires use of fully scoped names because there could be another **TAccount** class defined in another unit and kbmMW needs to know exactly which one you want it to use.
The key setting refers to which field (or array of fields) should be used in TPerson to use as a key when finding matching TAccount instances in the datastorage.

The sourceKey setting obviously refers to which fields in the the TAccount class that the key settings should be matched up against. The number of fields must be equal between key and sourceKey.

A number of fields can be defined like this:
key:[field1,field2], sourceKey:[afield1,afield2]
Now if we query using the **TPersonWithAccounts** class like this:

```
procedure TForm6.Button4Click(Sender: TObject);
var
  o:TObjectList<TPersonWithAccounts>;
begin
   o:=orm.QueryList<TPersonWithAccounts>('SELECT * FROM
   uData.TPersonWithAccounts WHERE FullName LIKE ?',['%MS%']);
   // The above shows another way to query using kbmMW's built in SQL support.
   // We will now have recieved a list of TPersonWithAccounts which match the query,
   // all populated from the datastorage represented by TPerson, and
   // magically the Account property of each of the TPersonWithAccounts instances will
  // contain the accounts matching that specific person.
   o.Free;
end;
```

We will have received not only the person information, but also the matching accounts.

Let us look at a reverse scenario. We have a **TAccount** but we would also like to get the matching person that holds the account.

Again that can be done in multiple ways, but a nice clean one is like this:

```
 [kbmMW_VirtualTable(TAccount)]
 TAccountWithPerson = class(TAccount)
 private
   FPerson:TPerson;
 public
   destructor Destroy; override;

   [kbmMW_VirtualField('name:person,
    source:uData.TPerson, key:PID, sourceKey:ID')]
   property Person:TPerson read FPerson write FPerson;
 end;

...

destructor TAccountWithPerson.Destroy;
begin
   FPerson.Free;
   inherited;
end;
```

Now querying for an account will automatically also return an object matching the person holding the account.

```
procedure TForm6.Button3Click(Sender: TObject);
var
  o:TAccountWithPerson;
begin
   o:=orm.Query<TAccountWithPerson>('SELECT * FROM
uData.TAccountWithPerson WHERE Value>?',[9000]);
   o.Free;
end;
```

In this case we get a list of all accounts with matching **TPerson** instance for accounts having a value of more than 9000. If there are multiple accounts held by the same person, that match this, each of the returned instances will have each their own instance of the **TPerson.**
But what if we actually do not need a complete **TPerson** instance, but only the name of the person?
Let us make a nice clean class for that:

```
[kbmMW_VirtualTable(TAccount)]
  TAccountWithPersonName = class(TAccount)
  private
    FFullName:kbmMWNullable<string>;
  public
    [kbmMW_VirtualField('name:fullName, source:uData.TPerson, key:PID, sourceKey:ID,
      value:uData.TPerson.FullName')]
    property FullName:kbmMWNullable<string> read FFullName write FFullName;
  end;
```

And let us query it:

```
procedure TForm6.Button5Click(Sender: TObject);
var o:TAccountWithPersonName;
begin
  o:=orm.Query<TAccountWithPersonName>('SELECT * FROM uData.TAccountWithPersonName
  WHERE Value>?',[9000]);
  o.Free;
end;
```

Now the first record found matching in the **TAccount** datastorage is returned in the **TAccountWithPersonName,** along with the person's name.
In fact it is even possible to do complex things in the value setting, by writing any expression that **kbmMW** handles, like:

```
[kbmMW_VirtualField('name:fullName, source:uData.TPerson, key:PID, sourceKey:ID,
value:"Mr. "||uData.TPerson.FullName')]
```

Which will set the field value to "Mr. Hans Hansen" in case the persons full name is "Hans Hansen", or

```
[kbmMW_VirtualField('name:fullName, source:uData.TPerson, key:PID, sourceKey:ID,
value:"uData.TPerson.FullName||\" Age:\"||uData.TPerson.Age"')]
Which will concatenate the persons full name with the persons age and enter that in the FullName
```

Which will concatenate the persons full name with the persons age and enter that in the **FullName property** of **TAccountWithPersonName.**

In fact all kbmMW's SQL column expression features can be used here, including things like string, math, date etc. manipulation functions.

How does **kbmMW** do this?
Its **ORM** is clever enough to understand the query statement you may have provided, and rewrite it to match the datastorage database engine, including adding needed manipulations and joins if the database engine supports it.
If it does not, **kbmMW** will instead attempt to emulate the features needed.

KBMMW FEATURES #3 – DATE/TIME, TIMEZONES AND MORE

BY KIM MADSEN

The new **kbmMW** Features blog post serie will talk about various smaller, but useful, features within **kbmMW.**

This blog will be about the **TkbmMWDateTime** structure, which is a **TDateTime** on stereoids.

A plain **TDateTime** is essentially nothing but a double sized floating point value. It stores the number of days (*and a fraction of a day*) since 12/30/1899 12:00 AM.

It seems easy to work with, but unfortunately its ease of use is deceptive.

Why you may ask? Because it usually ends up with the developer storing local date/time values in databases and elsewhere.

It is simply lacking the concept of timezones.

TkbmMWDateTime always operates with full knowledge about timezones. Hence you can easily ask for a **TkbmMWDateTime** expressed in any currently available timezone on earth. which essentially means that **TkbmMWDateTime** is universal. In fact timezones by themselfs are not enough, since we often operate with daylightsaving in some countries in the world.

TkbmMWDateTime also handles that.
Further **TkbmMWDateTime** makes it easy to express dates and times in various standard and non standard formats, easing conversion between string representations and numerical representations, on which one can do date and time manipulation.

So how to work with **TkbmMWDateTime?**

Its easy.

```
var
 dt:TkbmMWDateTime;
begin
 dt:=TkbmMWDateTime.Now;
end;
```

This instantiate a **TkbmMWDateTime** to the current date/time. If you want to figure out what that is in **UTC** time (*or in other words in the GMT timezone*), type:

```
'The time is '+DateTimeToString(dt.UTC)
```

And to get the current timezone registered with the time:

```
'Current timezone is :'+dt.GetTimeZone;
```

It will return something similar to +2:00. As you may notice, despite **TkbmMWDateTime** has full knowledge of the worlds timezone names, it will not return the timezone name. The reason is that there are many timezone names that can match a single timezone offset. Instead presenting a timezone as an offset value is a generally accepted way.
And to return the date/time as the local time:

```
'The local time is '+DateTimeToString(dt.Local)
```

kbmMW can convert to and from ISO8601, RFC-1123 and NSCA formatted date/time strings along with handling Unix like epoch based values.

To return an **ISO8601** formatted string based:

```
s:=dt.ISO8601String
```

Which will put something like '01-10-2001T12:00:00.000+01:00' in the variable s (The actual value obviously depends on what value dt has been assigned).

Similarly assigning a new ISO8601 formatted string value to dt is done like this:

```
dt.ISO8601String:='01-10-2001T12:00:00.000+01:00'
```

The **ISO8601** format is the defacto standard in most data exchange protocols like **XML, JSON** etc.

The **RFC-1123** format is often used in **HTML** headers and other internet protocols.

The **NSCA** format is often used in **WWW** and **FTP** server log files.

The **TkbmMWDateTime** is also able to understand and produce **ISO8601** formatted duration values, like 5 weeks, 2 days and 1 hour.

Just a few other formats it understands are **fixed format, temporenc, since epoch** in **ns** or **ms**, and **custom formatted strings.**

You can also make calculations using TkbmMWDateTime, adding or subtracting seconds, minutes, hours, days, months and years, add or subtract durations even expressed in fractional weeks and so forth.

The upcoming release of kbmMW now also supports advanced custom formatting features.

```
var
  dt1:TkbmMWDateTime; s : string;
begin
   dt1.UTCAsFormat['%Y-%D-%M']:='2018-17-12'; // Set according to format
   s:=dt1.UTCAsFormat['%Y/%D/%M'];         // Get according to format

   dt1.UTCAsFormat['%Y-%M-%D %H:%N:%S']:='2018-12-17 23:15:22';

   dt1.UTCAsFormat['%Y-%M-%D %H:%N:%S.%Z']:='2018-12-17 23:15:22.123';

   dt1.UTCAsFormat['%Y-%M-%D %H:%N:%S %P']:='2018-12-17 11:15:22 AM';

   dt1.UTCAsFormat['%Y-%M1-%D %H:%N:%S %P']:='2018-Feb-17 11:15:22 PM';

   dt1.UTCAsFormat['%Y-%M-%D%i%i%i%H:%N:%S %P']:='2018-12-17abc11:15:22 PM';

   dt1.UTCAsFormat['%Y-%M-%D%I, %H:%N:%S %P']:='2018-12-17 Monday, 11:15:22
PM';
```

Just to show some of the setting and getter options.

```
%Y = 4 digit year
%Y1 = 2 digit year 19xx
%Y2 = 2 digit year 20xx
%Y3 = 2 digit year >=50=19xx, <50=20xx
%M = 1 or 2 digit month
%M1 = 3+ char US month name
%M2 = 3+ char locale month name
%D = 1 or 2 digit day
%H = 1 or 2 digit hour
%N = 1 or 2 digit minute
%S = 1 or 2 digit second
%Z = 1,2 or 3 digit millisecond
%T = Timezone
%P = A/P/AM/PM
%i = Ignore one character.
%Ix = Ignore all characters until x
%% = %

Eg.
 %Y-%M-%D %H:%M:%S
 %D.%M.%Y %H:%M:%S
```

The following two example format strings shows it is possible to make advanced filtering using regular expressions to extract the values and interpret them as required. Just follow the format as given above with an equal sign and a regular expression extracting data to populate those bits and pieces.

The first one:

```
%D.%M.%Y=(\d{2}).(\d{2}).(\d{4})
```

is interpreted as 2 digits that is a day of month, 2 digits that is a month of year and 4 digits that is the year.

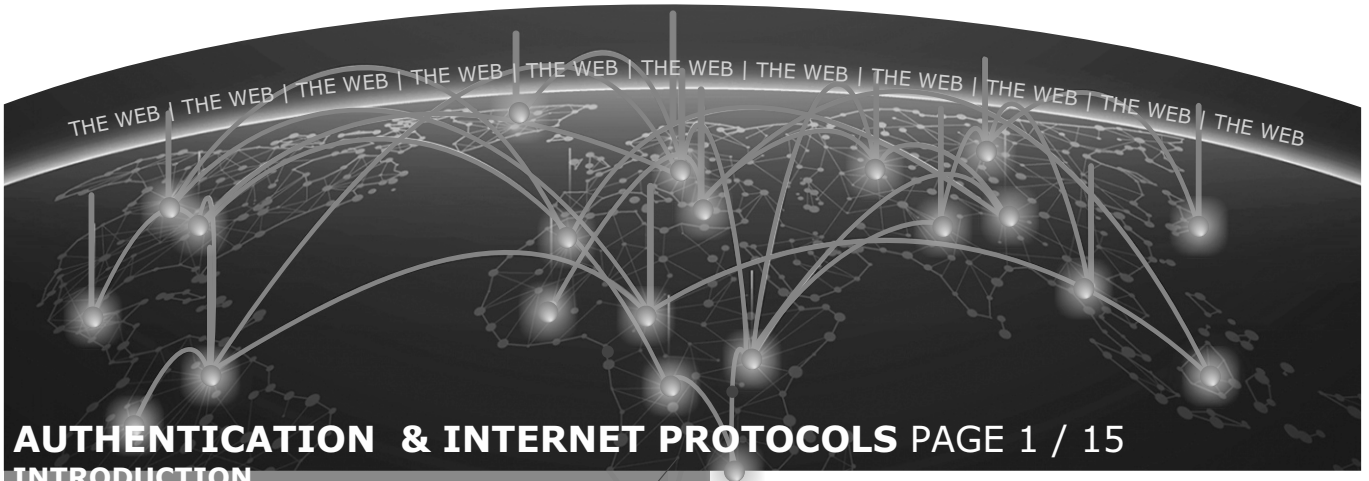
The second one:

```
Dato\=%D.%M.%Y='Dato='(\d{2}).(\d{2}).(\d{4}
)
```

outputs
"Dato=27.05.2018"
when converting to a string, and is able to parse the strings of same format.

This is just a taste of the nice features in TkbmMWDateTime. In addition it supports having a null value, tracks value changes, can have a default value and more.

Finally the same unit now also includes a TkbmMWGregorianCalendar class which for now can calculate easter start/end for any year, in both the socalled western (gregorian) style and in eastern (orthodox or julian) style.



AUTHENTICATION & INTERNET PROTOCOLS

INTRODUCTION

In this series of articles I will try to handle the following main goals: first of all a summary of the various problems that have to be hanlded:

- WHAT MEANS AUTHENTICATION?
- AUTHENTICATION FACTORS AND NEEDS
- TYPES OF AUTHENTICATION
- ENROLMENT AND AUTHENTICATION PROCESS

- A SECTION ABOUT INTERNET PROTOCOLS: PAGE 56

1. A Client - Server app, capable only of communication with each other. I will try to explain what the various possibilities are and show some diagrams to create an overview of possible services and how to handle them. The architecture of all of the main items will be shown separately. I will try to explain what needs to be done to make a secure authentication process possible.
2. Authentication with TMS XDATA
   – This will be the next article in Issue 73
3. Authentication for EMBARCADERO RAD SERVER
4. Authentication for NODE.JS
5. Authentication for ASP.NET CORE MICROSERVICES

WHAT MEANS AUTHENTICATION?

Authentication refers to a computed process that allows for the electronic identification and that could be a natural person. On top of that, authentication can also confirm the origin and integrity of data in electronic form, like a digital certificate to attest to the authenticity of a website.
The main goal of authentication is to diminish as far as possible the ability for fraud, in the event of an individual purposely misrepresenting their identity or through the unauthorized use of another person's credentials.

The terms **DIGITAL AUTHENTICATION** or **ELECTRONIC AUTHENTICATION (*e-authentication*)** synonymously refer to the process where the confidence in user identities is established and presented electronically to an information system.

The digital authentication process presents a technical challenge due to the necessity of authenticating individual people or entities remotely over a network.
Its level of security depends on the applied type of authentication, the authentication factors used, as well as the process of authentication applied.
This first article describes the theoretical background of a first step to create a Client and a Server that will be able to communicate with each other. We have build a small model so you can immediately start your first steps towards understanding what is necessary for this process.

AUTHENTICATION FACTORS AND NEEDS

For authenticating a user, there are three main items that could be examined to assure that the user is who he or she makes a claim to be, (this is what it is all about). These factor categories are:

1 PRIVATE KNOWLEDGE

– this would include a user's **password, passphrase, personal identification** number (pin) or a **challenge response** where the user would be required to answer a pre-selected security question.

2 OWNERSHIP PROOVE

– these includes things that the user has , a **BANK CARD**, a hardware or software **One Time Password** (**OTP**) **token** or a **CELL PHONE**,
or at least the number of it, possibly able to be called by a service.

3 IRREFUTABLE INHERENCE

– these factors relate to special circumstances that a user can be or do and includes **BIOMETRIC IDENTIFIERS** such as **FACIAL, FINGERPRINT** or **RETINAL PATTERN RECOGNITION**
and other personal trait identifiers.

TYPES OF AUTHENTICATION

The subsequent categorization lists the most frequently used types of online user authentication sorted based on increasing levels of security:

1. **ONE ITEM AUTHENTICATION**
   only one component out of one of the 3 categories is used for authentication. It is quite obvious that one single item does **not provide sufficient protection against malicious intrusion and misuse.**

   So as soon as financially or personally relevant transactions are involved, a higher level of security must be used.
2. **TWO FACTOR AUTHENTIFICATION (2FA**),
   by which the user's identity is confirmed by using a combination of two independent components from two different factor categories.

   For example, where a user has logged on to their online bank account, with their username and password, and wishes to complete an online transaction, he or she would need to enter an authentication factor in addition to the knowledge factor (*username and password*) that was used to log on. **That is factor 1.**
   The additional **factor (2**) must also be from a different factor category than the username and password.
   An online banking user would normally use an authentication method from the ownership category such as an **OTP** device (*created by some special device*) or mobile phone to receive an **OTP** in a text message.
   OTPs are dynamic passwords which can only be used once and thereby provide a strong level of protection against a range of attacks. You could even extend this - what banks sometimes do: a time slot .

3. **STRONG AUTHENTICATION**
   this type is often used as synonym for multi-factor authentication or **2FA**.
   However, unlike **multi-factor authentication** and **2FA,** strong authentication mandatorily requires non replicable factors or the use of digital certificates to provide a higher level of authentication for users.
   If those criteria are fulfilled, multi-factor authentication and **2FA** are able to provide strong authentication.

In **Europe** the **European Central Bank (ECB)** wants strong customer authentication
"a procedure based on two or more of the three authentication factors, 1. Knowledge, 2. Ownership, and 3. Inherence. According to the ECB definition, the individual elements that are chosen for strong authentication factors must be mutually independent and at least one must be non-reusable and non-replicable (except for inherence), and not capable of being surreptitiously stolen via the internet."

Independence is relatively easy to grasp - if one component is related to the other, hacking one means all components are tampered with.
Non-replicability implies the aspect of time or usage. Used once it cannot be used again.
An example is a one-time-password which is only valid during a short period of time (*e.g., 30 seconds*).
In the **United States**, the **National Information Assurance (IA) Glossary** produced by the **Committee on National Security Systems** wants strong authentication similarly, requiring
"multiple factors for authentication and advanced technology, such as dynamic passwords or digital certificates to verify an entity's identity."
Dynamic passwords include the aspect of time.

As an extra to the ECB definition, the IA also accepts cryptographic means such as a public key certificates to co-authenticate a user.

ENROLMENT AND AUTHENTICATION PROCESS

The **American National Institute of Standards and Technology (NIST)** has outlined a quite generic digital authentication model, which can be used as a basic explanation model for the authentication process, regardless of the geographical region or area of jurisdiction.

NIST AUTHENTICATION GUIDELINE

In the **NIST** model, an individual **(APPLICANT)** applies to a **CREDENTIAL SERVICE PROVIDER (CSP)** and thus initiates the enrolment process. Once the **CSP** has successfully proven the applicant's identity, he or she becomes a "SUBSCRIBER", and an AUTHENTICATOR (*e.g., token*) as well as a corresponding credential, such as a **USERNAME,** are established between the **CSP** and the APPLICANT (*whome now is owning the the role of the* SUBSCRIBER). The **CSP** has the task of maintaining the credential including its status and all enrolment data over the whole lifetime of the credential. The SUBSCRIBER needs to maintain the AUTHENTICATOR(S).

This first part of the **NIST REFERENCE** model is applied in any enrollment process, where subsequent authentication is required, e.g., when a bank account is created or when a person signs up in an e-government process

Once the APPLICANT has become a "SUBSCRIBER", he or she can perform online transactions within an authenticated session, conducted with a relying party.

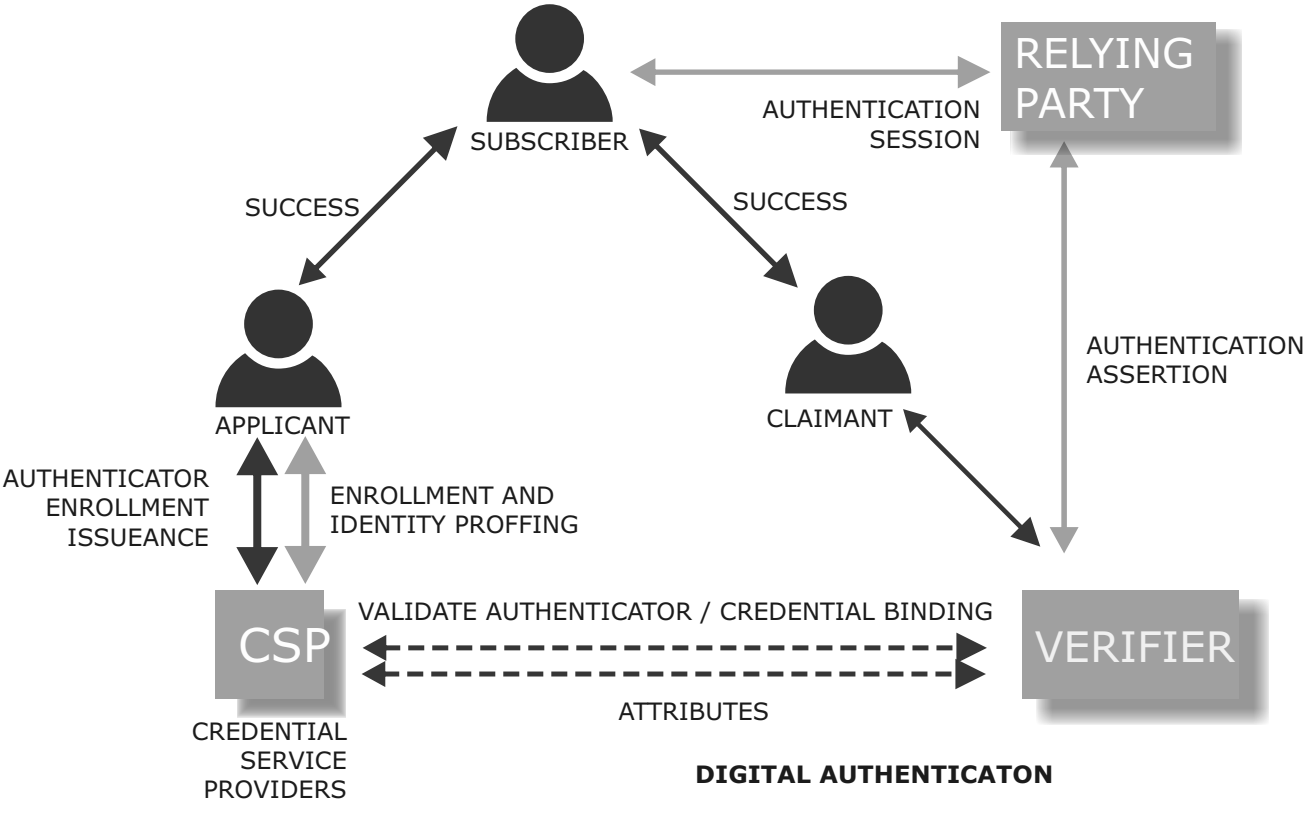
In such a transaction, the person holds the role of a CLAIMANT, proving to a verifier the possession of one or more authenticators.
VERIFIER and relying party might be the same or alternatively two independent entities.
If VERIFIER and relying party are separate, the verifier has to provide assertion about the SUBSCRIBER to the relying party.
Subsequent to this this assertion, the relying party may then initiate the transaction process.
Important in case you want to create a Shop.

In the following, a typical payment sequence illustrates this reference process:
The account owner (**"SUBSCRIBER"**) wants to initiate a transaction.

He or she first needs to prove through one or more authenticators that he or she, who claims to be the account owner (**"CLAIMANT"**) actually is the person he claims to be (subscriber).
The validation is done by a **"VERIFIER"** who verifies the authenticators at the **"CREDENTIAL SERVICE PROVIDER"** and after validation gives authentication assertion to the transaction department of the bank (*relying party*).
In many banks the entities **"VERIFIER"** and **"CREDENTIAL SERVICE PROVIDER"** are probably entities within the the bank.

One familiar use of authentication and authorization is access control. A computer system that is supposed to be used only by those authorized must attempt to detect and exclude the unauthorized.

Figure 1 **ENROLLMENT AUTHENTICATOR AND LIFECICLE MAINTENANCE**



Access to it is therefore usually controlled by insisting on an authentication procedure to establish with some degree of confidence the identity of the user, granting privileges established for that identity. One such procedure involves the usage of **LAYER 8** which allows IT administrators to identify users, control Internet activity of users in the network, set user based policies and generate reports by username.

Some examples of access control involving authentication include:

1. Using **CAPTCHA*** as a means of asserting that a user is a person and not a computer program.
2. By using a one-time password (**OTP**), received on a tele-network enabled device like mobile phone, as an authentication password or **PIN (Personal Identification Number)**
3. A computer program using a **BLIND CREDENTIAL** to authenticate to another
4. **LOGGING INTO** a computer
5. Using a confirmation E-mail to verify ownership of an e-mail address
6. Using an Internet banking system

*A **CAPTCHA** an acronym for "**Completely Automated Public Turing** test to tell **Computers and Humans Apart**") is a type of challenge–response test used in computing to determine whether or not the user is human.*
This form of CAPTCHA requires that the user type the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen. Because the test is administered by a computer, in contrast to the standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.
This user identification procedure has received many criticisms, especially from disabled people, but also from other people who feel that their everyday work is slowed down by distorted words that are difficult to read. It takes the average person approximately 10 seconds to solve a typical CAPTCHA.

In some cases, ease of access is balanced against the strictness of access checks.
For example, the credit card network does not require a personal identification number for authentication of the claimed identity, and a small transaction usually does not require a signature of the authenticated person for proof of authorization of the transaction. The security of the system is maintained by limiting distribution of credit card numbers, and by the threat of punishment for fraud.
As to our readers it is welknown that it is impossible to prove the identity of a computer user with absolute certainty. It is only possible to apply one or more tests which, if passed, have been previously declared to be sufficient to proceed. The problem is to determine which tests are sufficient, and many such are inadequate.
Atomic authorization is the act of securing authorization rights independently from the intermediary applications to which they are granted and the parties to which they apply.

For an application using strong (N-factor) authentication, traditional authorization techniques pose a security vulnerability. The application must rely upon technologies like database queries or directory lookups, which are protected using single-factor authentication, for authorization information and management.

INTERNET (NETWORK) PROTOCOLS

A network protocol defines rules and conventions for communication between network devices.
Network protocols include mechanisms for devices to identify and make connections with each other, as well as formatting rules that specify how data is packaged into messages sent and received.
Some protocols also support message acknowledgment and data compression designed for reliable and/or high-performance network communication.

Modern protocols for computer networking all generally use **packet switching techniques to send and receive messages in the form of packets** - messages subdivided into pieces that are collected and re-assembled at their destination.
Hundreds of different computer network protocols have been developed each designed for specific purposes and environments.

The **Internet Protocol** family contains a set of related *(and among the most widely used network protocols).*
Beside **Internet Protocol (IP)** itself, higher-level protocols like **TCP, UDP, HTTP**, and **FTP** all integrate with **IP** to provide additional capabilities.
Similarly, lower-level **Internet Protocols** like **ARP** and **ICMP** also co-exist with **IP.**

In general, higher level protocols in the IP family interact more closely with applications like Web browsers while lower-level protocols interact with network adapters and other computer hardware.

WIRELESS NETWORK PROTOCOLS

Thanks to **Wi-Fi, Bluetooth** and **LTE,** wireless networks have become commonplace. Network protocols designed for use on wireless networks must support **roaming* mobile devices** and deal with issues such as variable data rates and network security.

** Roaming is a wireless telecommunication term typically used with mobile devices (like mobile phones). It refers to the mobile phone being used outside the range of its home network and connects to another available cell network.*

Network Routing Protocols

Routing protocols are special-purpose protocols designed specifically for use by network routers on the Internet.
A routing protocol can identify other routers, manage the pathways (*called routes*) between sources and destinations of network messages, and make dynamic routing decisions. Common routing protocols include **EIGRP, OSPF,** and **BGP.**

How Network Protocols are Implemented

Modern operating systems contain built-in software services that implement support for some network protocols.
Applications like Web browsers contain software libraries that support the high level protocols necessary for that application to function.

For some lower level **TCP/IP** and routing protocols, support is implemented in direct hardware (*silicon chipsets*) for improved performance.
Each packet transmitted and received over a network contains binary data (*ones and zeros that encode the contents of each message*).
Most protocols add a small header at the beginning of each packet to store information about the message's sender and its intended destination.
Some protocols also add a footer at the end.
Each network protocol has the ability to identify messages of its own kind and process the headers and footers as part of moving data among devices.

A group of network protocols that work together at higher and lower levels is often called a **protocol family**.
Students of networking traditionally learn about the **OSI*** model that conceptually organizes network protocol families into specific layers for teaching purposes.

***Open Systems Interconnection model.**
See Figure 2 next page.
The (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology.

Its goal is the interoperability of diverse communication systems with standard protocols. The model partitions a communication system into abstraction layers.
The original version of the model **defined seven layers.**

A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that comprise the contents of that path.
Two instances at the same layer are visualized as connected by a horizontal connection in that layer.
The model is a product of the Open Systems Interconnection project at the International Organization for Standardization (ISO), maintained by the identification ISO/IEC 7498-1.

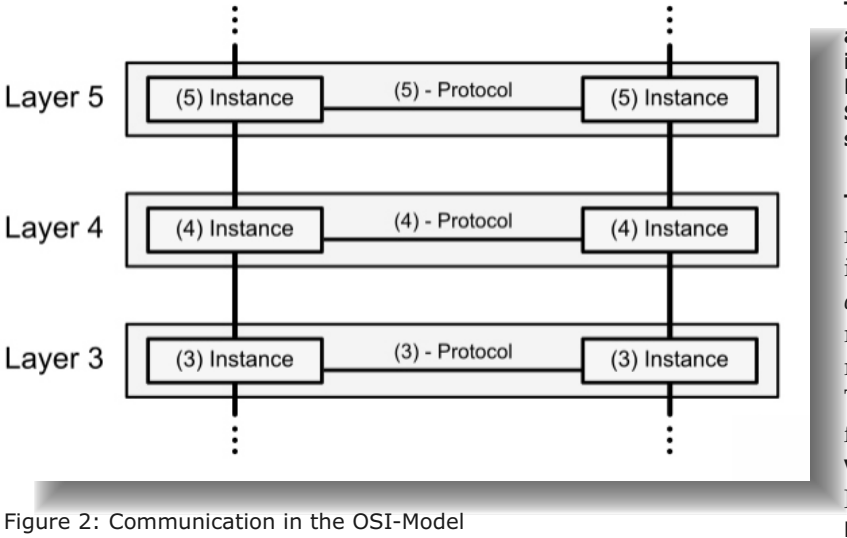


Figure 2: Communication in the OSI-Model (example with layers 3 to 5)

NETWORK FUNCTION

The **Transmission Control Protocol** provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the transport layer of the Internet model.

An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required **IP** fragmentation to accommodate the maximum transmission unit of the transmission medium.
At the transport layer, **TCP** handles all handshaking and transmission details and presents an abstraction of the network connection to the application typically through a network socket interface.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behaviour, **IP** packets may be lost, duplicated, or delivered out of order.
TCP detects these problems, requests re-transmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems.

If the data still remains undelivered, the source is notified of this failure.
Once the **TCP** receiver has reassembled the sequence
of octets originally transmitted, it passes them to the receiving application. Thus, **TCP** abstracts the application's communication from the underlying networking details.

TCP is used extensively by many applications available by internet, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery and can incur relatively long delays (*on the order of seconds*) while waiting for out-of-order messages or re-transmissions of lost messages.
Therefore, it is not particularly suitable for real-time applications such as **Voice over IP**.
For such applications, protocols like the **Real-time Transport Protocol (RTP)** operating over the **User Datagram Protocol (UDP)** are usually recommended instead. **TCP** is a reliable **stream delivery service** which guarantees that all bytes received will be identical with bytes sent and in the correct order.

Since packet transfer by many networks is not reliable, a technique known as **positive acknowledgement** with re-transmission is used to guarantee reliability.
This fundamental technique requires the receiver to respond with an acknowledgement message as it receives the data. The sender keeps a record of each packet it sends and maintains a timer from when the packet was sent. The sender re-transmits a packet if the timer expires before receiving the message acknowledgement. The timer is needed in case a packet gets lost or corrupted.
While **IP** handles actual delivery of the data, **TCP** keeps track of 'segments' - the individual units of data transmission that a message is divided into for efficient routing through the network.

For example, when an HTML file is sent from a web server, the TCP software layer of that server divides the sequence of file octets into segments and forwards them individually to the IP software layer (Internet Layer).

The **Internet Layer** encapsulates each **TCP** segment into an **IP** packet by adding a header that includes *(among other data)* the destination **IP** address. When the client program on the destination computer receives them, the **TCP** layer (**Transport Layer**) re-assembles the individual segments and ensures they are correctly ordered and error-free as it streams them to an application.

FUNCTION OF COMMON NETWORKING PROTOCOLS

In computing, **a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between computing endpoints.**

In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication.
Protocols may be implemented by hardware, software, or a combination of the two.

At the lowest level, a protocol defines the behavior of a hardware connection.
While protocols can vary greatly in purpose and sophistication, most specify one or more of the following properties:

- **DETECTION OF THE UNDERLYING PHYSICAL CONNECTION** (*wired or wireless*), or the existence of the other endpoint or node
- **HANDSHAKING** (*dynamically setting parameters of a communications channel*)
- **NEGOTIATION OF VARIOUS CONNECTION CHARACTERISTICS**
- **HOW TO START AND END A MESSAGE**
- **HOW TO FORMAT A MESSAGE**
- **WHAT TO DO WITH CORRUPTED OR IMPROPERLY FORMATTED MESSAGES** (*error correction*)
- **HOW TO DETECT** unexpected loss of the connection, and what to do next
- **TERMINATION OF THE SESSION AND OR CONNECTION.**

TCP/IP (TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL) SUITE

The **Internet Protocol Suite** (*commonly known as TCP/IP*) is the set of communications protocols used for the Internet and other similar networks.
The **Internet Protocol Suite**, like many protocol suites, may be viewed as **a set of layers**.

Each layer solves a set of problems involving the transmission of data, and provides a well-defined service to the upper layer protocols based on using services from some lower layers.
Upper layers are logically closer to the user and deal with more abstract data, relying on lower layer protocols to translate data into forms that can eventually be physically transmitted.

The **TCP/IP** model consists of four layers.
From lowest to highest, these are the **Link Layer**, the **Internet Layer**, the **Transport Layer**, and the **Application Layer**. Some have attempted to map the **Internet Protocol** model onto the **seven-layer OSI Model**. The mapping results in the **TCP/IP Link Layer** corresponding to the **OSI Data Link** and **Physical layers** in terms of functionality. The **Internet Layer** is usually directly mapped to the **OSI's Network Layer**. At the top of the hierarchy, the **Transport Layer** is always mapped directly into the **OSI Layer 4** of the same name. **OSI's Application Layer**, **Presentation Layer**, and **Session Layer** are collapsed into **TCP/IP's Application Layer.**

The following table provides some examples of the protocols grouped in their respective layers. See the below sections for details on each protocol.

Application	DNS, TFTP, TLS/SSL, FTP, HTTP,
	IMAP4, POP3, SIP, SMTP, SNMP, SSH,
	Telnet, RTP
Transport	TCP, UDP
Internet	IP (IPv4, IPv6), ICMP, IGMP
Link	ARP

ARP (Address Resolution Protocol)
The Address Resolution Protocol (ARP) is a communications protocol used for resolution of Internet layer addresses into link layer addresses, a critical function in the **Internet protocol suite**.

ARP was defined by **RFC 826** in 1982, and is **Internet Standard STD 37**.

ARP is also the name of the program for manipulating these addresses in most operating systems.

ARP is used for mapping a network address (*e.g. an IPv4 address*) to a physical address like an Ethernet address (*also named a MAC address*).

ARP has been implemented with many combinations of network and data link layer technologies, like **IPv4, Chaosnet, DECnet** and **Xerox PARC Universal Packet (PUP)** using **IEEE 802** standards, **FDDI, X.25,** Frame Relay and **Asynchronous Transfer Mode (ATM)**.

IPv4 over **IEEE 802.3** and **IEEE 802.11** is the most common usage. In **Internet Protocol Version 6 (IPv6)** networks, the functionality of **ARP** is provided by the **Neighbor Discovery Protocol (NDP).**

DHCP (Dynamic Host Configuration Protocol)
This is a protocol that is used to assist users to configure multiple network devices from a single source. This protocol is used to assist user to configure multiple network.

DNS (Domain Name System)
This is a protocol that assists the users by helping to link between common usernames/works to an IP address and back, e.g. instead of cramming the IP address for **BLAISE PASCAL MAGAZINE** the user can easily type
`https://www.BlaisePascalMagazine.eu`
to be able to access the content from there.

FTP (File Transfer Protocol)
FTP is a standard internet protocol for transmitting files between computers on the internet over **TCP/IP** connections.
It is a client server protocol that relies on two communications channels between client and server; a command channel for controlling the conversation and a data channel for transmitting files content.

Clients initiate conversations with servers by requesting to download a file.

FTP sessions work in a passive and active mode, after a client initiates a session via a command channel request, the server initiates a data connection back to the client and begins transferring data. In passive mode the server instead uses the command channel to send the client the information it needs to open a data channel. It works across firewalls and network Address Translation **(NAT)** gateways

HTTP (Hyper Text Transfer Protocol)
This was the initial protocol that were used to access web content, because of its security vulnerabilities it was replaced by **`HTTPS`**.

HTTPS (Hypertext Transfer Protocol Secure)
This is the secure version of HTTP that is now commonly used to access website content.

ICMP (Internet Control Message Protocol)

IGMP (Internet Group Management Protocol)

IMAP4 (Internet Message Access Protocol version 4)

NTP (Network Time Protocol)
POP3 (Post Office Protocol version 3)
RTP (Real-time Transport Protocol) - VoIP
(Voice over Internet Protocol)

SIP (Session Initiation Protocol) - VoIP
(Voice over Internet Protocol)

SMTP (Simple Mail Transfer Protocol)
This is the protocol that is used to send and receive emails between users.

SNMP2/3 (Simple Network Management Protocol version 2 or 3**)**

SSH (Secure Shell) see below thsi is now the most common used protocol.

TCP (Transmission Control Protocol)

Telnet

TFTP (Trivial File Transfer Protocol)

TLS (Transport Layer Security)

UDP (User Datagram Protocol)

SSH (Secure Shell)
Secure Shell **(SSH)** is a cryptographic network protocol for operating network services securely over an unsecured network.
The best known example application is for remote login to computer systems by users.

SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution, **but any network service can be secured with SSH.**

The protocol specification distinguishes between two major versions, referred to as **SSH-1** and **SSH-2.**

The most visible application of the protocol is for access to shell accounts on Unix-like operating systems.
In 2015, **Microsoft** announced that they would include native support for **SSH** in a future release.
In WINDOS 10 new versions it is now standard supported

SSH was **designed as a replacement** for **Telnet** and for **unsecured remote shell protocols** such as the B**erkeley rlogin, rsh,** and **rexec protocols**.

Those protocols send information, notably passwords, in plaintext, rendering them susceptible to interception and disclosure using packet analysis.

The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet, although files leaked by Edward Snowden indicate that
THE NATIONAL SECURITY AGENCY CAN SOMETIMES DECRYPT SSH, ALLOWING THEM TO READ THE CONTENTS OF SSH SESSIONS.

DEFINITION

SSH uses **public-key cryptography** to authenticate the remote computer and allow it to authenticate the user, if necessary. There are several ways to use **SSH;** one is to use automatically generated public-private key pairs to simply encrypt a network connection, and then use password authentication to log on.

Another is to use a manually generated **public-private key pair** to perform the authentication, allowing users or programs to log in without having to specify a password.
In this scenario, anyone can produce a matching pair of different keys (*public and private*).
The public key is placed on all computers that must allow access to the owner of the matching private key (*the owner keeps the private key secret*).
While authentication is based on the private key, the key itself is never transferred through the network during authentication. **SSH** only verifies whether the same person offering the public key also owns the matching private key.
In all versions of **SSH** it is important to verify unknown public keys, i.e. associate the public keys with identities, before accepting them as valid.
Accepting an attacker's public key without validation will authorize an unauthorized attacker as a valid user.

KEY MANAGEMENT

On Unix-like systems, the list of authorized public keys is typically stored in the home directory of the user that is allowed to log in remotely, in the file `~/.ssh/authorized_keys`.
This file is respected by **SSH** only if it is not writable by anything apart from the owner and root. When the public key is present on the remote end and the matching private key is present on the local end, typing in the password is no longer required (*some software like* **Message Passing Interface (MPI)** *stack may need this password-less access to run properly*). However, for additional security the private key itself can be locked with a passphrase.

The private key can also be looked for in standard places, and its full path can be specified as a command line setting (the option -i for ssh). The ssh-keygen utility produces the public and private keys, always in pairs.

SSH also supports password-based authentication that is encrypted by automatically generated keys. In this case, the attacker could imitate the legitimate server side, ask for the password, and obtain it (*man-in-the-middle attack*). However, this is possible only if the two sides have never authenticated before, as **SSH** remembers the key that the server side previously used. The **SSH** client raises a warning before accepting the key of a new, previously unknown server. Password authentication can be disabled.

USING SSH

Logging into **OpenWrt** via **SSH** using **PuTTY** running on **Windows. SSH** is a protocol that can be used for many applications across many platforms including most **Unix** variants **(Linux,** the **BSDs** including **Apple's macOS, and Solaris**), as well as **Microsoft Windows**. Some of the applications below may require features that are only available or compatible with specific **SSH** clients or servers. For example, using the **SSH protocol** to implement a **VPN** is possible, but presently only with the **OpenSSH server** and **client** implementation.

- For login to a shell on a remote host (*replacing Telnet and rlogin*)
- For executing a single command on a remote host (*replacing rsh*)
- For setting up automatic *(passwordless)* login to a remote server (*for example, using OpenSSH*)
- In combination with rsync to back up, copy and mirror files efficiently and securely
- For forwarding or tunneling a port (*not to be confused with a VPN, which routes packets between different networks, or bridges two broadcast domains into one*).
- For using as a full-fledged encrypted **VPN.** Note that only **OpenSSH** server and client supports this feature.
- For forwarding X from a remote host (*possible through multiple intermediate hosts*)
- For browsing the web through an encrypted proxy connection with **SSH** clients that support the **SOCKS** protocol.
- For securely mounting a directory on a remote server as a filesystem on a local computer using **SSHFS.**
- For automated remote monitoring and management of servers through one or more of the mechanisms discussed above.
- For development on a mobile or embedded device that supports **SSH.**

File transfer protocols

The **Secure Shell** protocols are used in several file transfer mechanisms.

Secure copy (SCP),
which evolved from **RCP** protocol over **SSH**

rsync, intended to be more efficient than **SCP.** Generally runs over an **SSH** connection.

SSH File Transfer Protocol (SFTP), a secure alternative to **FTP** (not to be confused with **FTP** over **SSH** or **FTPS)**

Files transferred over shell protocol (a.k.a. FISH), released in 1998, which evolved from **Unix Shell** commands over **SSH**

Fast and Secure Protocol (FASP), aka **Aspera,** uses **SSH** for control and **UDP** ports for data transfer.

ARCHITECTURE
The **SSH-2** protocol has an internal architecture (*defined in RFC 4251*) with well-separated layers:

The transport layer (RFC 4253).
This layer handles **initial key exchange** as well as **server authentication**, and **sets up encryption**, compression and integrity verification. It exposes to the upper layer an interface for sending and receiving plaintext packets with sizes of up to 32,768 bytes each (*more can be allowed by the implementation*).

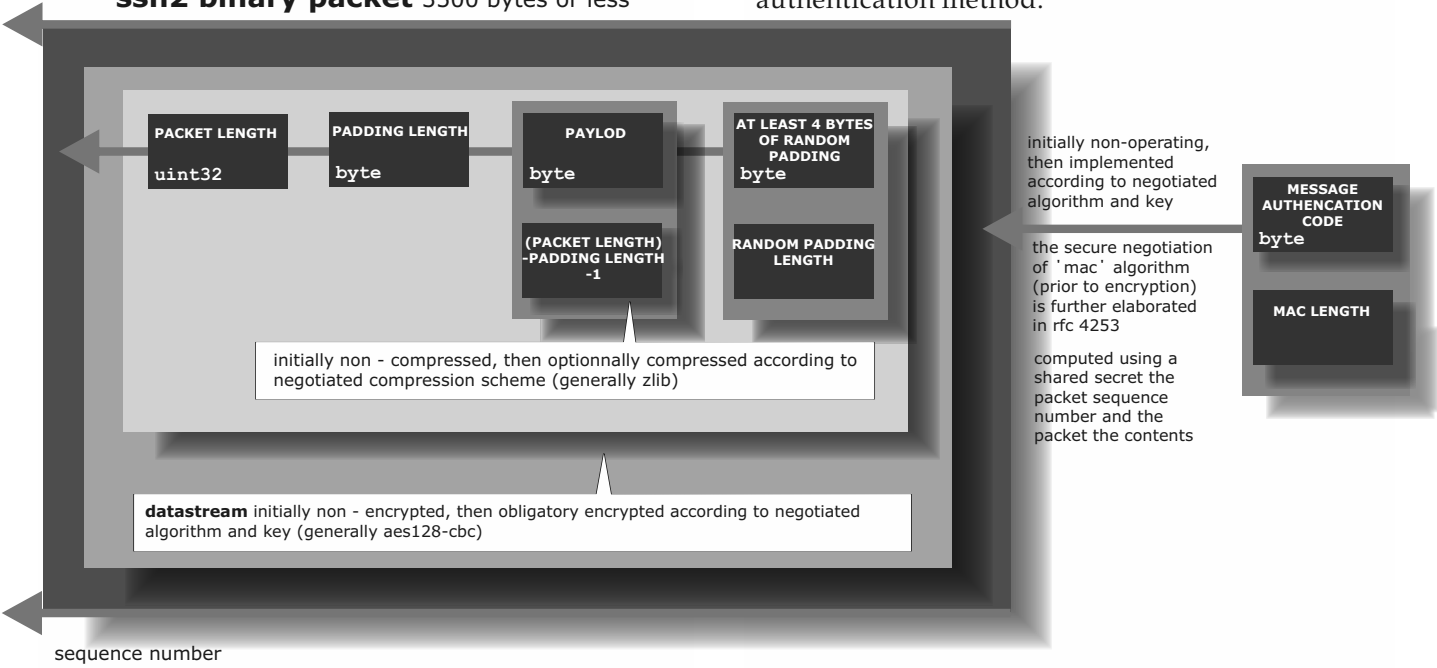
The transport layer also arranges for key re-exchange, usually after 1 GB of data has been transferred or after 1 hour has passed, whichever occurs first.

The user authentication layer (RFC 4252).
This layer handles client authentication and provides a number of authentication methods. Authentication is client-driven: when one is prompted for a password, it may be the **SSH client** prompting, not the server.
The server merely responds to the client's authentication requests. **Widely used user-authentication methods include the following:**

password:
a method for straightforward password authentication, including a facility allowing a password to be changed. Not all programs implement this method.

public key:
a method for public key-based authentication, usually supporting at least **DSA** or **RSA** keypairs, with other implementations also **supporting X.509** certificates.

keyboard-interactive (RFC 4256):
a versatile method where the server sends one or more prompts to enter information and the client displays them and sends back responses keyed-in by the user. Used to provide one-time password authentication such as **S/Key** or **SecurID.**
Used by some **OpenSSH** configurations when **PAM** is the underlying host-authentication provider to effectively provide password authentication, sometimes leading to inability to log in with a client that supports just the plain password authentication method.

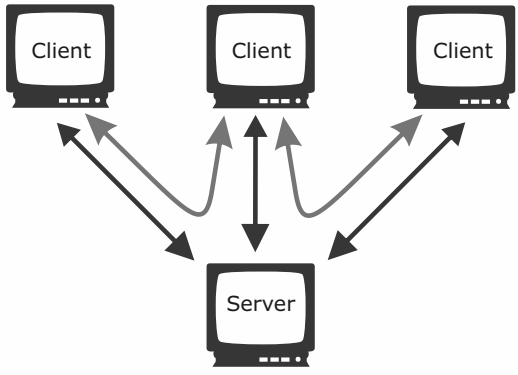


sequence number
...is an implicit packet sequence number represented as uint32. The packet sequence number itself is not included in the packet send over the wire. The sequence number is initialized to zero for the first packet, and is after every packet (regardless of wether encryption or MAC is in use.) It is never reset (even if key/algorithms are renegotiated later!) It wraps around to zero after every 2^32 packets.

starter expert Delphi

BUILDING YOUR FIRST CLIENT SERVER MODEL

This application is meant to give you some insight in the basic system: Communication. Actually this workes byte for byte. That is important to keep in mind. We have a server that has to communicate with the client and the other way around.
That is the most basic element. In this project we build a server and a client in one project. If you want to run this project to see the functionality, start with the server and set it to active. After that you can connect the client. You also can create several clients by repeatedly starting one from the exe file within the project directory.
Al ready now you can see its complexity, looking at the code. But remember problems are not solved by their entirety but by breaking them in to smaller parts helps. So that is what we do: in this first part we create the communication. In a second article we create the many layers we need to create an authentication. In that case there will be an encrypted filetransfer. Jean Pierre Hoefnagel is the creator and he wrote this for us and is using his own encryption model, which has its advantages. SSH works in a similar way but will be handled apart.

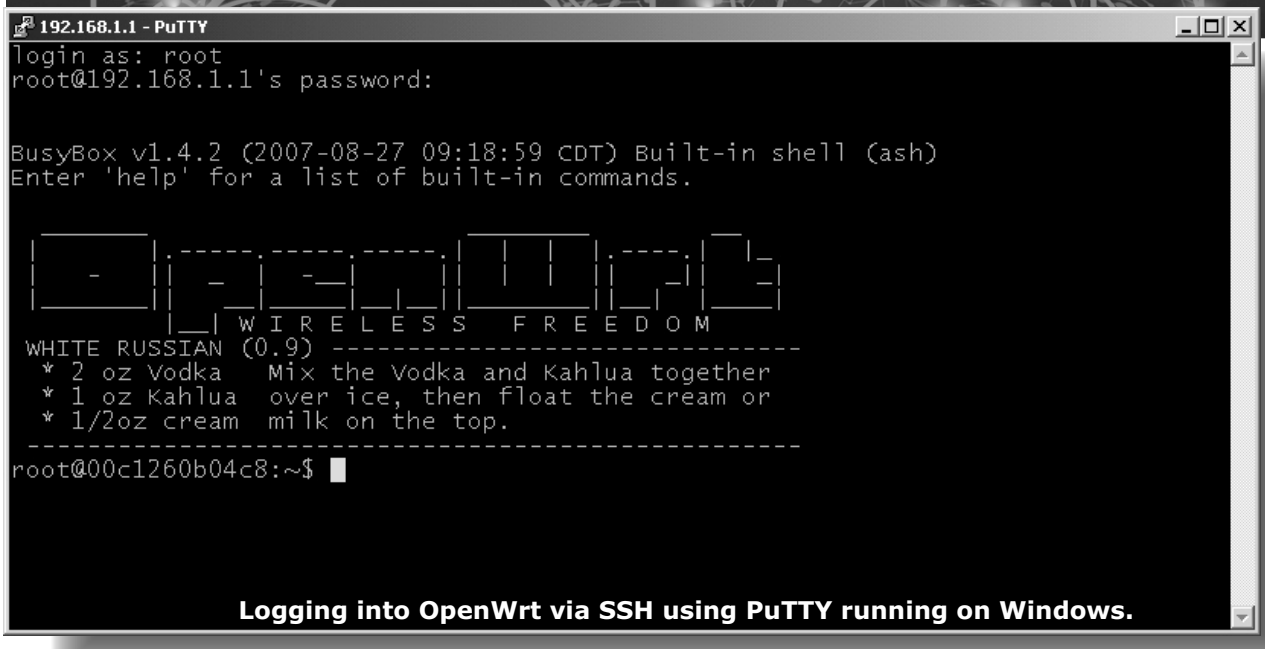



```
procedure TFServer.ButtonLogInClick(Sender: TObject);
begin
 IdTCPServer1.DefaultPort := StrToIntdef(Edit1.Text, 888);
 Edit1.Text := IntTostr(IdtcpServer1.DefaultPort);
 if IdTCPServer1.Active then  IdTCPServer1.Active := False;
 try
   IdTCPServer1.Active := ButtonLogIn.Down;
 except
   on e: exception do log(E.message, clred);// (log is vcl click)
 end;
 if IdTCPServer1.Active then
   log('Server activated', clgreen)
 else
   log(' Server NOT activated', clBlue);
end;
```

THE CODE:

```
procedure TFServer.ColLog(aEdit:TRichEdit; aMsg:string; aColor:TColor;
          UpdateNow:boolean=false; Limitlines:boolean=true);
var  h, fh : integer;
begin
 for h: = 1 to length(aMsg) do if not (charinset(aMsg[h],[#32..#127])) then aMsg[h]:='|';
 aEdit.CaretPos := point(0,aEdit.lines.count);
 aEdit.SelAttributes.Color := aColor; aEdit.Lines.add(aMsg);
 h := aEdit.ClientHeight;
 if h<26 then h:=26; // if larger then set for maximum
 fh := abs(aEdit.Font.Height)+3;
 if limitlines then while aEdit.lines.count >= h div fh {14} do aEdit.Lines.Delete(0);
 if UpdateNow then aEdit.Update;
end;
```

A problem for logging is usually to make sure the number of lines that are to be remembered add up and become after some time a huge list, usually without meaning.
To keep this line short this function was created which do two things: Keep the number of lines as large as the viewable part of the list and it can give colour. You need of course to use for the list a Richedit and use a trick give the Richedit anchors so it will enlarge if you want it to, without scrolling. The colouring is an extra dimension: it gives you better insight in certain aspects.



Logging into OpenWrt via SSH using PuTTY running on Windows.

Description OpenWrt (White Russian 0.9) SSH session after root login in PuTTY client
Date 15 November 2007 **Source** self-made screenshot
Author Casablanca **Permission** (Reusing this file) OpenWrt and PuTTY are free software, licensed under GPL

- **GSSAPI authentication methods** which provide an extensible scheme to perform **SSH** authentication using external mechanisms such as Kerberos 5 or **NTLM,** providing single sign-on capability to SSH sessions. These methods are usually implemented by commercial **SSH** implementations for use in organizations, though **OpenSSH** does have a working **GSSAPI** implementation.

- **The connection layer (RFC 4254).** This layer defines the concept of channels, channel requests and global requests using which **SSH** services are provided. A single **SSH** connection can host multiple channels simultaneously, each transferring data in both directions. Channel requests are used to relay out-of-band channel-specific data, such as the changed size of a terminal window or the exit code of a server-side process. The **SSH** client requests a server-side port to be forwarded using a global request. Standard channel types include:
  - shell for terminal shells, **SFTP** and exec requests *(including* **SCP** *transfers)*
  - direct-tcpip for client-to-server forwarded connections **forwarded-tcpip** for server-to-client forwarded connections

- The **SSHFP DNS** record (**RFC 4255**) provides the public host key fingerprints in order to aid in verifying the authenticity of the host.

This open architecture provides considerable flexibility, allowing the use of **SSH** for a variety of purposes beyond a secure shell. The functionality of the transport layer alone is comparable to T**ransport Layer Security (TLS)**; the user-authentication layer is highly extensible with custom authentication methods; and the connection layer provides the ability to multiplex many secondary sessions into a single SSH connection, a feature comparable to **BEEP** and not available in **TLS.**

TO BE READ
Implementing SSL / TLS Using Cryptography and PKI
AUTHOR: Joshua Davies
PUBLISHER: John Wiley & Sons,

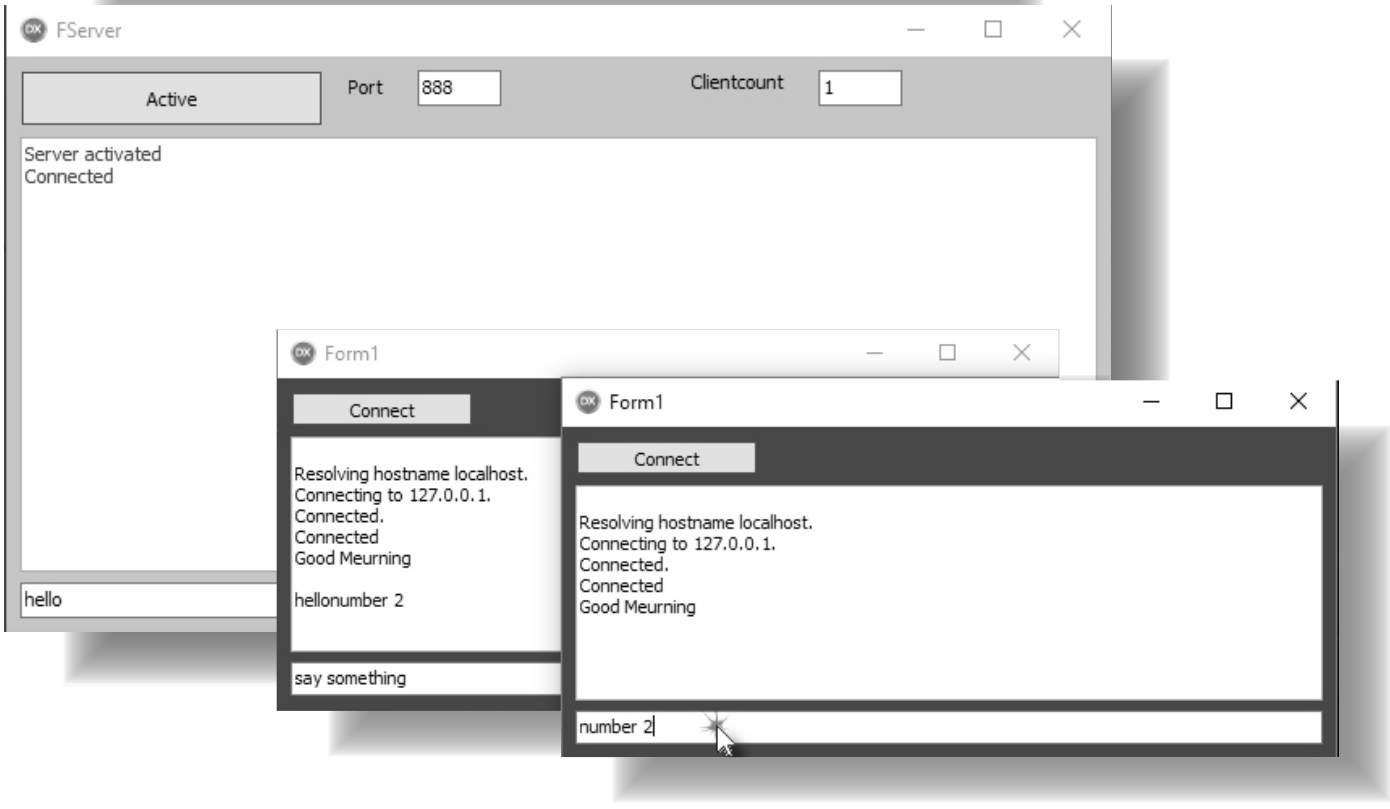
Network Security with OpenSSL: Cryptography for
AUTHOR: Secure Communications
PUBLISHER: "O'Reilly Media, Inc.",

Bulletproof SSL and TLS <-- RECENTly UPDATED
Understanding and deploying SSL/TLS and PKI to secure servers and web applications,
AUTHOR: Ivan Ristić
PUBLISHER: Feisty Duck

THIS ARTICLE IS LARGELY BASED ON MATERIAL TAKEN FROM WIKI AND A LOT OF OTHER WEBSITES

```
procedure TFServer.SendText(s: string; SkipMePlease: TIdContext);
var List : Tlist;
    N  : integer;
    ctx : TidContext;
begin
 List:= IdTcpServer1.Contexts.LockList;
 // Must be unlocked in extra try finally
 // sever has separate thread for ech cneected client, so it must be locked and unlocked
 // contains all client threads and will noet change until unlock list
 // (threadsafe list)
 Try
  for N:=0 to list.Count -1 do
  begin
   CTX := TIdContext(list[N]);  //for all connected clients
   if CTX <> SkipMePlease then  //if client is not me
   try
    if ctx.Connection.Connected then // and client is still connected send string
     ctx.Connection.Socket.Write(S);
   Except
    on e: exception do log(E.message, clred);
   End;
  end;
 Finally
  IdTCPServer1.contexts.UnlockList;
 End;
end;

procedure TFServer.Timer1Timer(Sender: TObject);
begin
 EdtClientCount.Text := IntToStr(IdTCPServer1.Contexts.Count)
end;
```



```
procedure TFServer.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
 sendtext(key,Nil); // only 1 char, but string is also allowed
 if Key = #13 then  // carriagereturn
 begin
  edit2.text := '';
  Key := #0;  // overcomes warning sound
 end;
end;

procedure TFServer.FormCreate(Sender: TObject);
begin
 lock:=TCriticalsection.create();
 slLog:=TStringlist.Create();
end;

procedure TFServer.FormDestroy(Sender: TObject);
begin
 lock.Free;
 slLog.Free;
end;
```

```
procedure TFServer.Log(Msg: string; col:Tcolor);
begin
// can be called from any thread (or VCL), no acces to VCL
// controlls directly.
 lock.enter;       // lock stringlist from all other threads
 try
  slLog.Add(msg); // add logged message to stringlist
 finally
  lock.Leave;      // allow other threads to access
 end;                                  // stringlist again.
end;
```

```
procedure TFServer.IdTCPServer1Connect(AContext: TIdContext);
begin
 log(AContext.Connection.Socket.BoundIP + 'Connected',ClBlue) ;
 AContext.Connection.Socket.WriteLn('Good Meurning');
end;

procedure TFServer.IdTCPServer1Exception(AContext: TIdContext; AException: Exception);
begin
 log(AContext.Connection.Socket.BoundIP +'  '+ AException.Message, clred);
end;

procedure TFServer.IdTCPServer1Execute(AContext: TIdContext);
Var B: Char;
begin
  Try
   if AContext.Connection.Connected then
   begin
    B:= Char(AContext.Connection.IOHandler.ReadByte());
    sendtext(B,acontext);
   end;
  Except
   on e: exception do log(E.message, clred);
  End;
end;
```

```
procedure TFServer.ApplicationEvents1Idle(Sender: TObject; var Done:
Boolean);
begin // running from vcl thread after all activity is done.
      // it is safe to copy content of slLog to edit box here.
 lock.Enter;
 try
  while slLog.Count>0 do begin
   collog(RichEdit1,slLog[0],clBlue);
   slLog.Delete(0);
  end;
 finally
  lock.Leave;
 end;
end;
```

```
procedure ColLog(aEdit:TRichEdit; aMsg:string; aColor:TColor;// no form connection - library function
          UpdateNow:boolean=false; Limitlines:boolean=true);
var h,fh:integer;
begin
 for h:=1 to length(aMsg) do if not (charinset(aMsg[h],[#32..#127])) then aMsg[h]:='|';
 aEdit.CaretPos:=point(0,aEdit.lines.count);
 aEdit.SelAttributes.Color:=aColor; aEdit.Lines.add(aMsg);
 h:=aEdit.ClientHeight; if h<26 then h:=26; fh:= abs(aEdit.Font.Height)+3;
 if limitlines then while aEdit.lines.count >= h div fh {14} do aEdit.Lines.Delete(0);
 if UpdateNow then aEdit.Update;
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
 Try
 if not IdTCPClient1.Connected then raise Exception.Create('First make connection');
 IdTCPClient1.Socket.Write(Key);
 except
  on e: exception do log(e.Message,Clred)
 end
end;

procedure TForm1.IdTCPClient1Connected(Sender: TObject);
begin
 log('Connected', clgreen);
end;

procedure TForm1.IdTCPClient1Status(ASender: TObject;
      const AStatus: TIdStatus; const AStatusText: string);
begin
 log(AStatusText,clred)
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
 if SpeedButton1.Down then
 IdTCPClient1.Connect
 else IdTCPClient1.DisConnect ;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
Var Cnt :integer;  c: Char;
begin
 try
  if IdTCPClient1.Connected then
  begin
   while not IdTCPClient1.IOHandler.InputBufferIsEmpty do
   begin
    C := (Char(IdTCPClient1.IOHandler.ReadByte)) ;
    if C= #13
    then RichEdit1.lines.add('')
    else RichEdit1.Text := RichEdit1.Text + C;
   end;
  end;
 except
  on e: exception do log(e.Message,Clred)
 end;
end;

procedure TForm1.Log(Msg: string; col:Tcolor);
begin
 Collog(RichEdit1, Msg, Col);
end;
```

THE COMPLETE PROJECT IS AVAILABLE ON YOUR SPECIAL DOWNLOAD PAGE:
https://www.blaisepascalmagazine.eu/my-downloads/

CREATE AN APP / CREATE A SETTINGS MODULE / BUILD AN INSTALLER

ALL DONE IN LAZARUS DELPHI7 AND DELPHI TOKYO

BY DETLEF OVERBEEK PAGE 1/15

starter expert

CREATING A CLOCK

INTRODUCTION

These days I had been plagued with the latest update of Win10. After this update my Clock was gone (*I had one of those beautyfull gadgets: an analogue clock*). After trying to find something on the internet I quitted. Let's make my own. A simple 5 minutes project. And since this seems so easy I could well write about it and do some extras that might be interesting:
How to create a setting for your clock so it would present it self next time you start it at the place you chose, create an Installer and make it be started automatically each time Window starts.
So all in all I thought would be useful for quite a lot reasons and create it for Lazarus, Delphi 7 and Delphi Tokyo. So almost for everyone. I did not know what I would stumble over Windows problems more then enough... but there is an easy solution as well I will show you...For all versions: Lazarus, Delphi 7 and Tokyo and the several windows versions

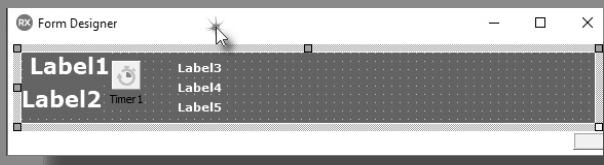


Figure 1: The form view

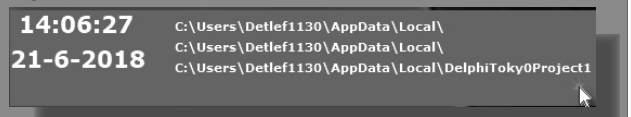


Figure 2 : Start view of the program, clicking on the form will autosize the form to the smallest size

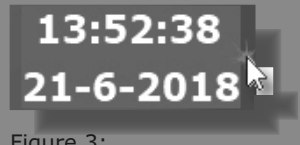


Figure 3:
Clicking on the form again will show the labels again

Figure 4: clicking on the time label will show the drag -area. You can move the clock window

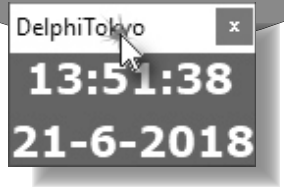


Figure 5 The button for closing is now avaliable

Figure 6:
Click on the date label to return to the borderless setting

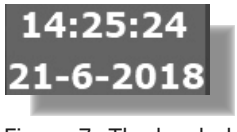


Figure 7: The borderless setting

BUILDING THE APP

Let's start with **Lazarus,** this is what we need: a small form put 2 labels on it and of of course a timer.
The two labels need to show the time and a date. That's all. The code fits on one page and is even for a very beginner simple to follow. Although there are some extras you need to know:
If you place the labels you will have to keep in mind some extra settings: Both labels have an **`onclick`** event, because I do not want to have buttons on this very limited surface.

So let's start making the font size of the labels big enough to read without putting your glasses on and I think a good contrasting colour might show off.
I chose for Lazarus for a nice blue (clHighlight) and the font in white.

In the **Object Inspector** select the **`Onclick`** event to do something about the kind of window we will be presenting: the **`BorderStyle`** of the form should be set to **`bsNone`** at **Designtime** but at **run time** it should give you the opportunity to do something clever: change the **`BorderStyle`** so you can drag your running application to the position where you want it and eventually even close this mini clock.
See Figure 1-7.

So actually there is a simple list of items that are handled though the labels and the form.
The details you see in the various images and its description. Actually what happens is that you open the app: click on the labels or click on the form. It all speaks for itself.

To make it better understood here is some code:
The timer makes of course the time an date available

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 Label1.Caption:= TimeToStr(Now);
 label2.Caption:= DateToStr(Now);
end;
```

```
procedure TForm1.Label2Click(Sender: TObject);
begin
 BorderStyle := bsNone;
end;

procedure TForm1.Label1Click(Sender: TObject);
begin
 BorderStyle := bsToolWindow;
end;
```

This actually concludes the whole of the app.
But now the trouble starts.

DRAGGING THE APPFORM

Dragging the Appform to the position where you want it to be First of all we must return the form to a kind of form where you can drag it and also a button appears so you can close the app.
We also want to make sure that the AppForm will remember the position it was dragged to.
This a litlle less easy. We will have to create an inifile and also an operation to save the data on closing: That's again easy:
Make a Formcreate event and a Close event.

CREATING AN INIFILE FOR THE SETINGS

Here is the code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  appINI := TIniFile.Create(ChangeFileExt(FN,'.ini'));
  try
   Top  := appINI.ReadInteger('Placement','Top', Top);
   Left  := appINI.ReadInteger('Placement','Left', Left);
   Width := appINI.ReadInteger('Placement','Width', Width);
   Height := appINI.ReadInteger('Placement','Height', Height);
  finally
   appINI.Free;
  end;
```

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  appINI := TIniFile.Create(ChangeFileExt(Fn,'.ini'));
 try
  with appINI, Form1 do
    begin
      WriteInteger('Placement','Top', Top);
      WriteInteger('Placement','Left', Left);
      WriteInteger('Placement','Width', Width);
      WriteInteger('Placement','Height', Height);
    end;
  finally
    appIni.Free;
  end;
end;
```

This actually works quite nice. But we want more. We want to save the inifile somewhere.

CAN'T UPDATE YOUR INIFILE

And her we get into trouble. This is not so easy at all:
If you want to make an App you install simply by hand no problem. On your system no problem.
But If you would like to make it installable for other people? Supposedly we are working on Windows. (Linux and Mac are quite different). And we could do that with Lazarus but not with **Delph**i versions.
So lets take windows. Most of you know that Windows nowadays has a very restricted regime to where you can put your files. You are, exept if you use the administrator-rights, not allowed to make any changes in the official directory where you put you program files: Let's say:
c:\Program Files (x86)\BlaisePascalMagazine
That area is protected and you can put your inifile here , but: you can't update it!

APPDATA\LOCAL?

Of course there is a solution for this, but this is not so easy as the rest of the program. First of all you will have to place your Ini file wher windows wants it: **c:\Users\Detlef1130\AppData\Local\DelphiToky0Project1**
This is quite troublesome: How to do that?
I have chosen for **Win 7** up to **Win 10** and all in between, otherwise the project would have become too complex.

So for older windows versions there are also solutions available, but since older versions are hardly anymore supported it seems to me a right descision.
So if you would like to use an installer we will explain where to get it and how to use it. But first of all the code for an **ini file** which is capable of updating your settings as you want it.

Happily I got some advise from Michael van Canneyt . There are two functions we need to create: we need to add to the second **uses** clause just under the implementation section:

```
uses SHFolder, shlObj;

var
 Form1: TForm1;

 appINI      : TIniFile;
 FN,Dir      : String;
```

and add to the variable the code above. For Delphi 7 there is no problem. For Lazarus you need to add Windows to the top **uses** section.

```
uses
 Classes, Windows {had to be added}, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
 ExtCtrls, IniFiles;  //<-- Lazarus!
```

```
function GetFolderPath(Wnd: HWnd; CSIDLFolder: Integer): string;
begin
 SetLength(Result, MAX_PATH);
 SHGetFolderPath(Wnd, CSIDLFolder, 0, 0, PChar(Result));
 SetLength(Result, StrLen(PChar(Result)));
 if (Result <> '') then  Result := IncludeTrailingBackslash(Result);
end;

function GetShellFolder(CSIDLFolder : integer) : string;
begin
 SetLength(Result, MAX_PATH);
 SHGetSpecialFolderPath(0, PChar(Result), CSIDLFolder, false);
 SetLength(Result, StrLen(PChar(Result)));
 if (Result <> '') then  Result := IncludeTrailingBackslash(Result);
end;
```

These functions make it possible to do settings in your **ini file** as needed. I have added a few extra labels to make sure you can see what is happening: set breakpoints and step with F8 through your project. They will show the path you want.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 Dir:=GetFolderPath(0,CSIDL_LOCAL_APPDATA);
 Label3.Caption:=Dir; // see the running app
 Label4.Caption:=GetShellFolder(CSIDL_LOCAL_APPDATA); // see the running app
 Dir:=Dir+ChangeFileExt(ExtractFileName(Application.ExeName),'');
 Label5.Caption:=Dir;  // see the running app
 If not ForceDirectories(Dir) then
  Raise Exception.CreateFmt('Failed to create settings directory %s',[Dir]);
 FN:= Dir+'\'+'appINI.ini';

  appINI := TIniFile.Create(ChangeFileExt(FN,'.ini'));
  try
   Top    := appINI.ReadInteger('Placement','Top', Top);
   Left   := appINI.ReadInteger('Placement','Left', Left);
   Width  := appINI.ReadInteger('Placement','Width', Width);
   Height := appINI.ReadInteger('Placement','Height', Height);
  finally
   appINI.Free;
  end;
end;
```

As you will understand the **oncreate** event creates the **ini file.** If it does not exist it creates the file anyway. The following code switches the labels on and off since you probably do not want to see them all the time - you want just a nice little clock...

```
procedure TForm1.FormClick(Sender: TObject);
begin
// show label 3,4,5
  Label3.Visible := Not Label3.Visible;
  Label4.Visible := Not Label4.Visible;
  Label5.Visible := Not Label5.Visible;
  Autosize       := Not AutoSize;
end;
```

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 appINI := TIniFile.Create(ChangeFileExt(Fn,'.ini'));
 try
 with appINI, Form1 do
  begin
   WriteInteger('Placement','Top', Top);
   WriteInteger('Placement','Left', Left);
   WriteInteger('Placement','Width', Width);
   WriteInteger('Placement','Height', Height);
  end;
 finally
  appIni.Free;
 end;
end;
```

And finally the on close event which sets the infile at the right place in your Appdata/Local Dir. This is important so you can update it after being installed like promised in the next section:

CREATING AN INSTALLER FOR AN APPLICATION

under windows - wether it's **Delphi 7, Delphi Tokyio** or **Lazarus**. After some reserach I found one I was especially interested in: The one where the **Windows-Lazarus installations** are made in: **Innosetup.** You can find it at: **http://www.jrsoftware.org/isdl.php#stable** or at **https://github.com/jrsoftware/ispack.** I chose this one because it installs your software without problems and it has a wizard so even if you never did anything like this before: Just try it.

Figure 8: The Innosetup logo

The intaller of the install-program is very easy to handle: In the directory **C:\Program Files (x86)\Inno Setup 5\Examples** you will find very good example code for your projects as wel and FAQ (*Frequent Asked Questions*) anda Helpfile. Its free for use and I must say I am impressed about it's simplicty and its wel behaviour. After installing you will find (Win10) this section in the Windows Program section.

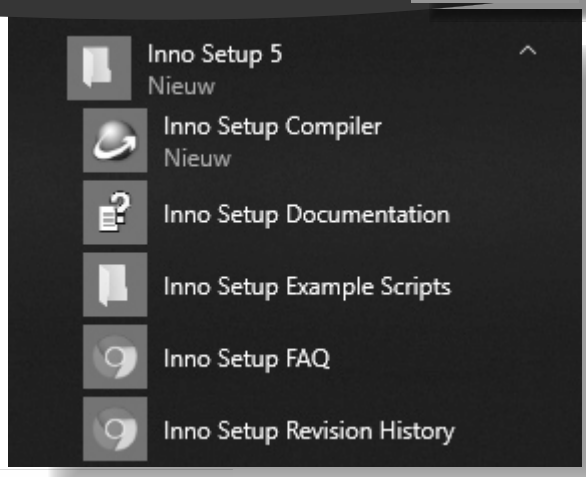


Figure 9: overview of start

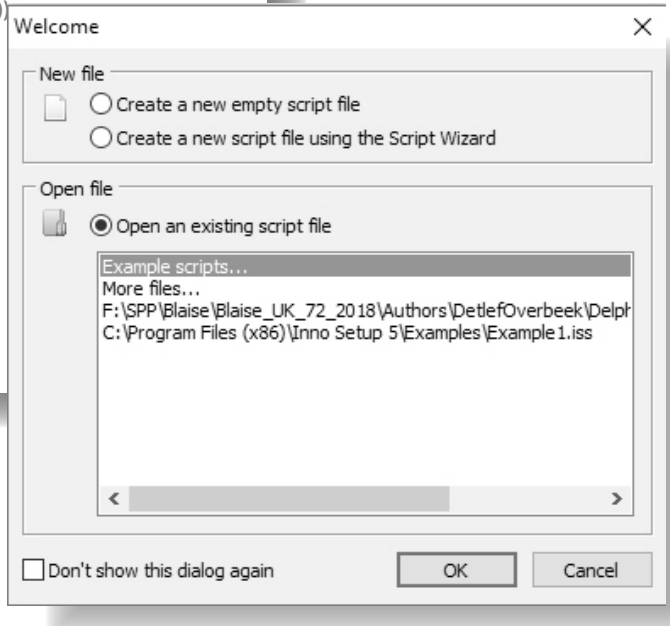


Figure 10:you best choose the create new script file

If you want to remove a project of this list you best remove it from the Dir where it has been saved. Then after starting this window you choose what you want. If it was removed in its Dir it will ask if you want to remove that project.

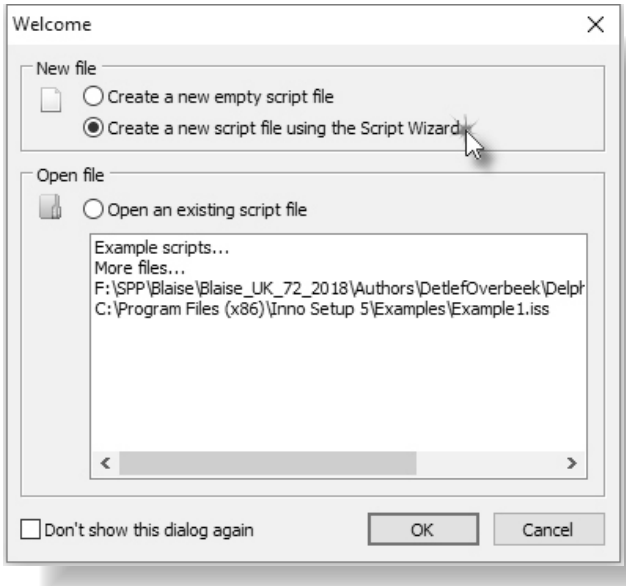


Figure 11: choose Create

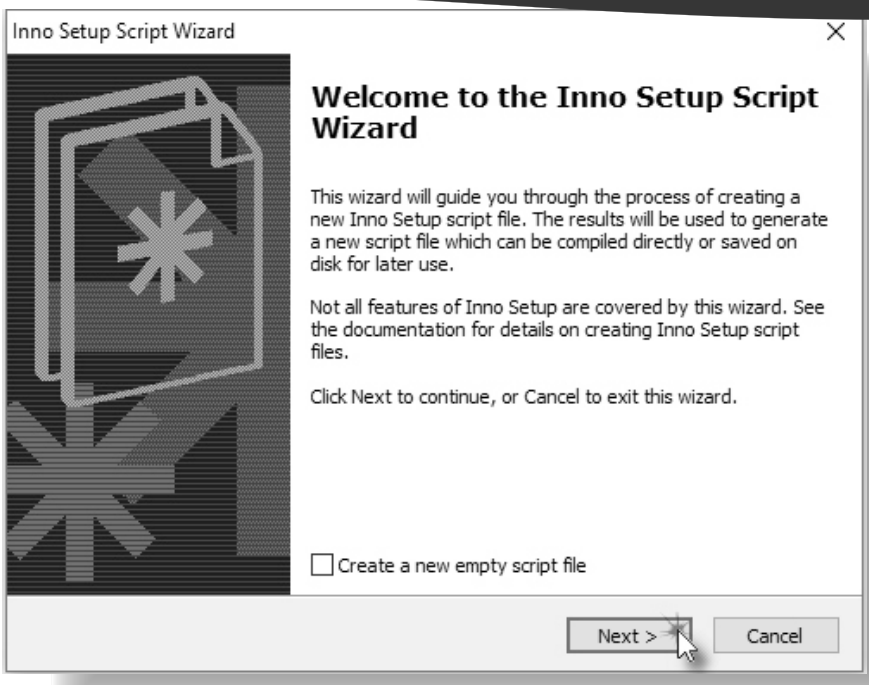


Figure 12:
Do not create a new empty file unless you are expierenced.

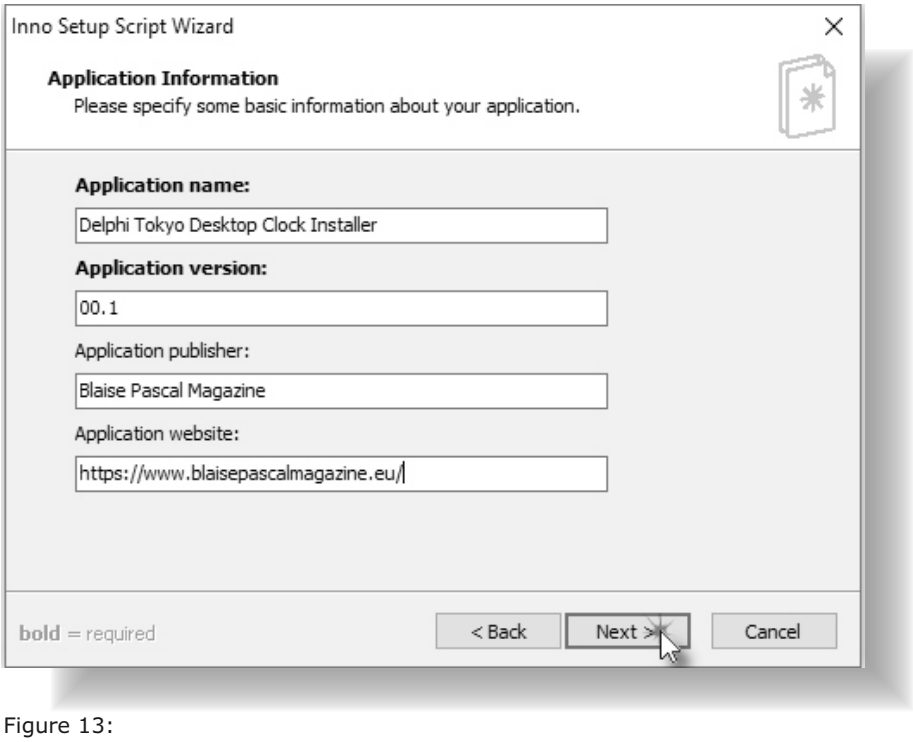


Figure 13:
Application name- the name of the program you want to instal Application Version the version number you want it to have: could be the same as in your project options - see next page.
Application Publisher: Could be you or some firm
Below: the website address of that company-or yours...

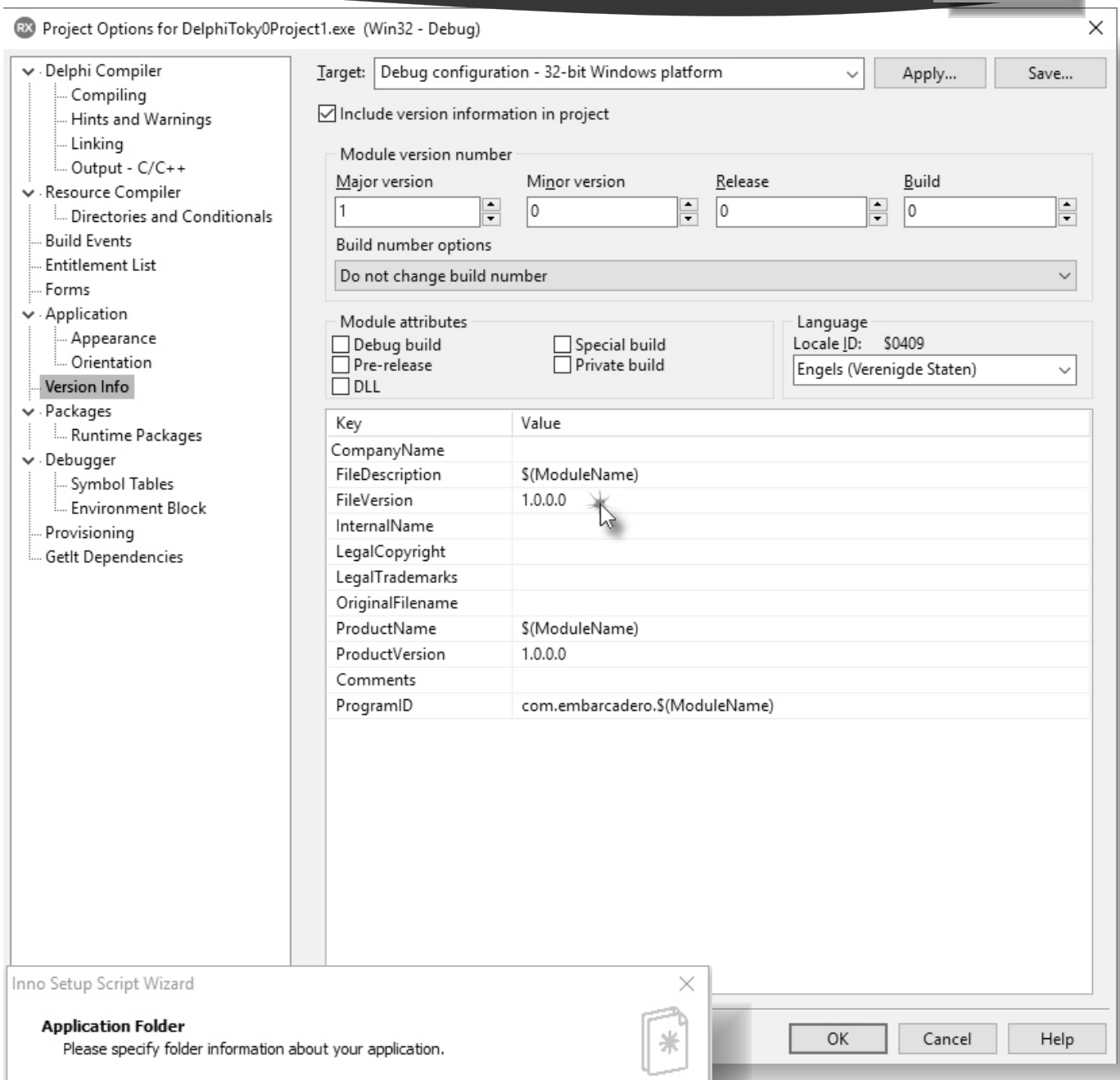
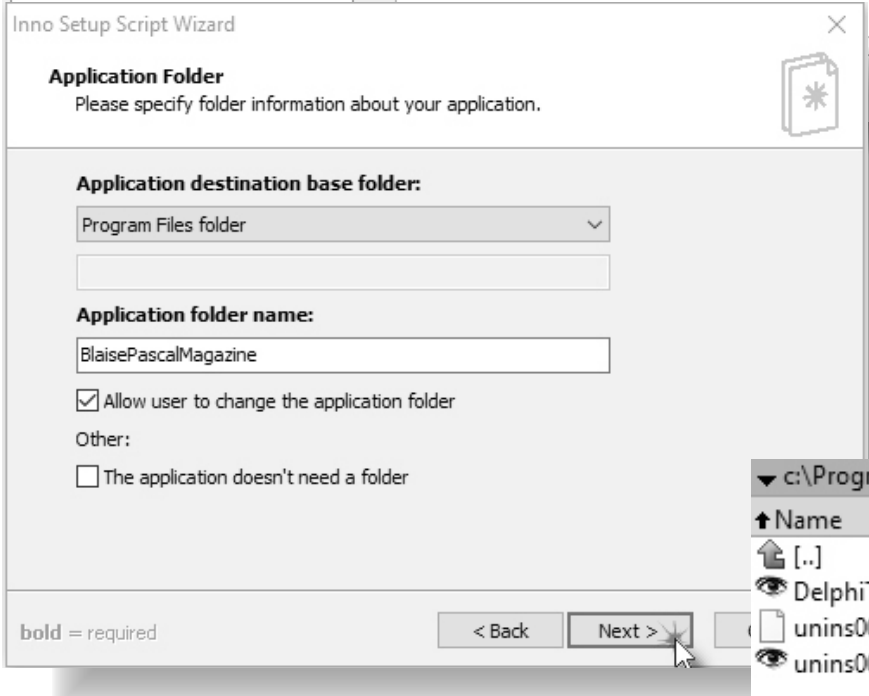


Figure 15: choose the file version or the productversion. They can differ.



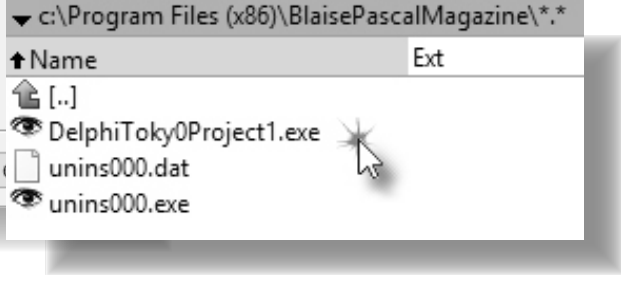


Figure 16,17: (left and below)
The folder means the Windows program files folder:
`c:\Program Files (x86)\BlaisePascalMagazine\`
Aplication folder name is the name of the folder that will be created after installing. The user can still chose an other setting.

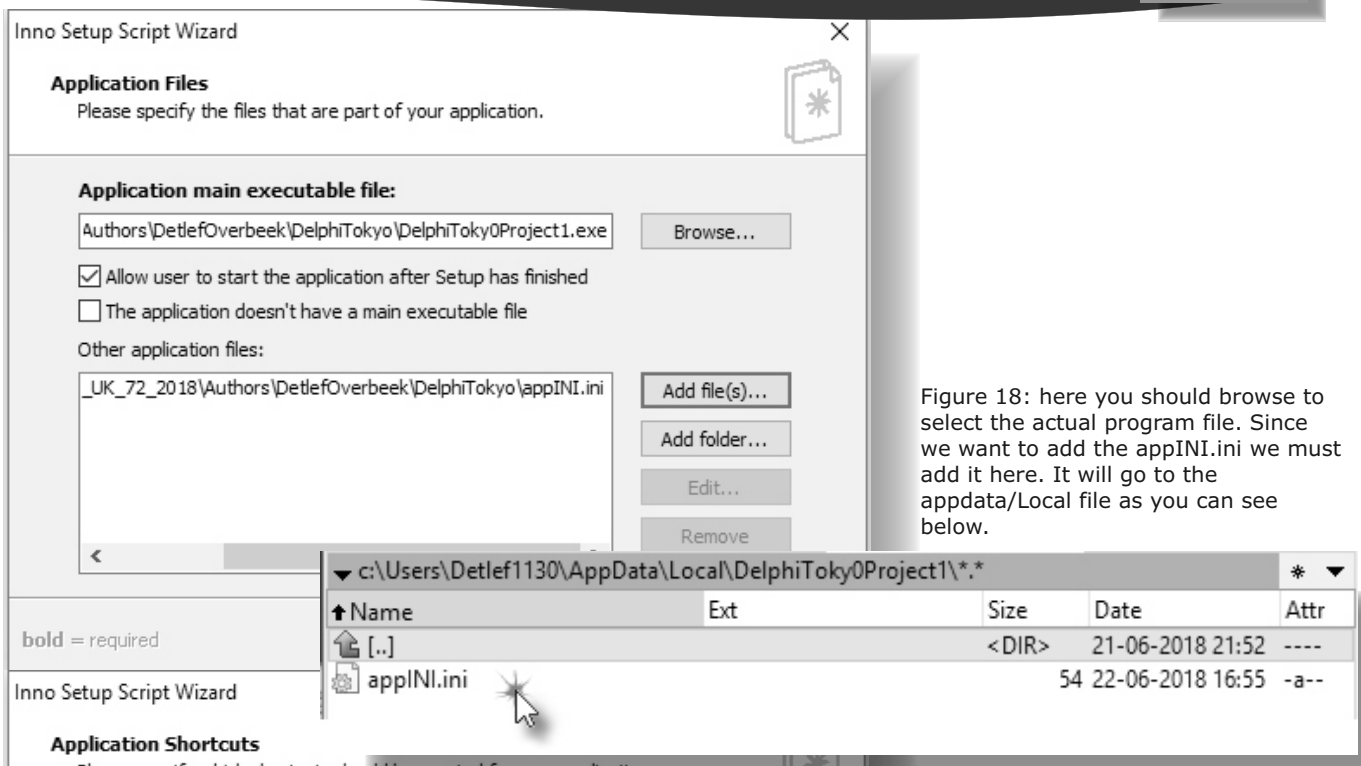


Figure 18: here you should browse to select the actual program file. Since we want to add the appINI.ini we must add it here. It will go to the appdata/Local file as you can see below.

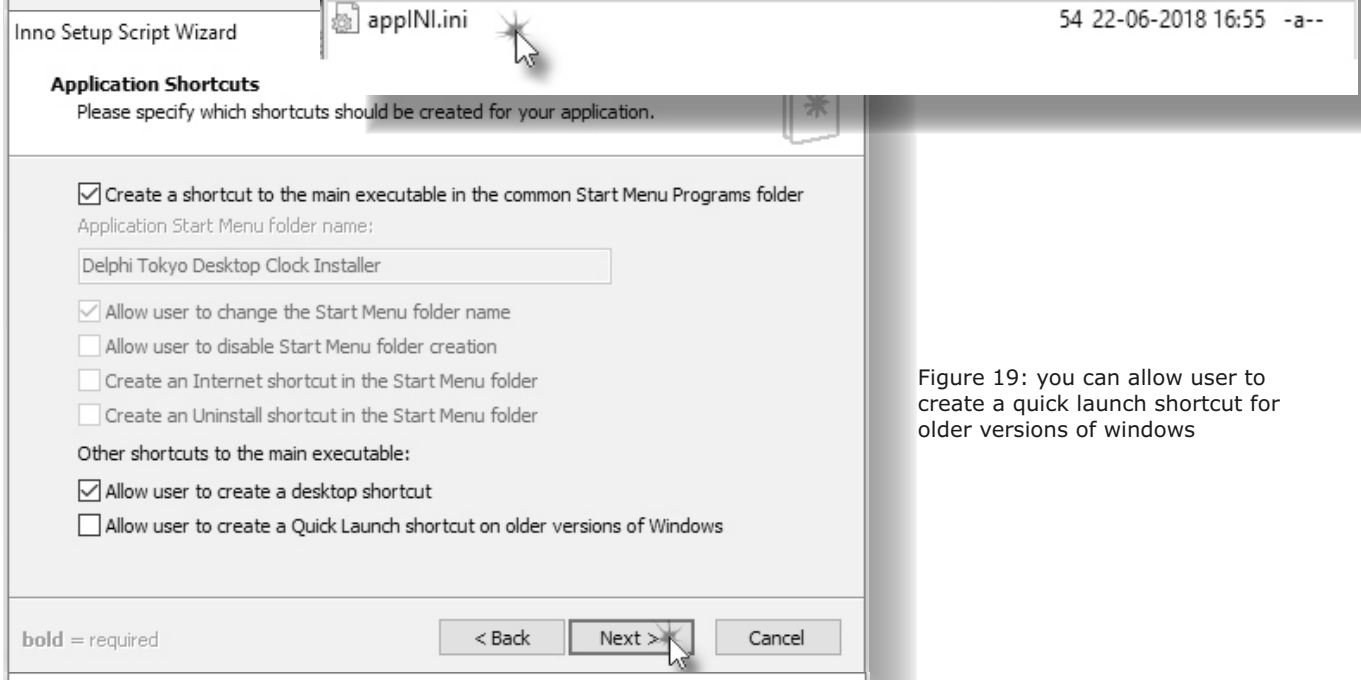


Figure 19: you can allow user to create a quick launch shortcut for older versions of windows

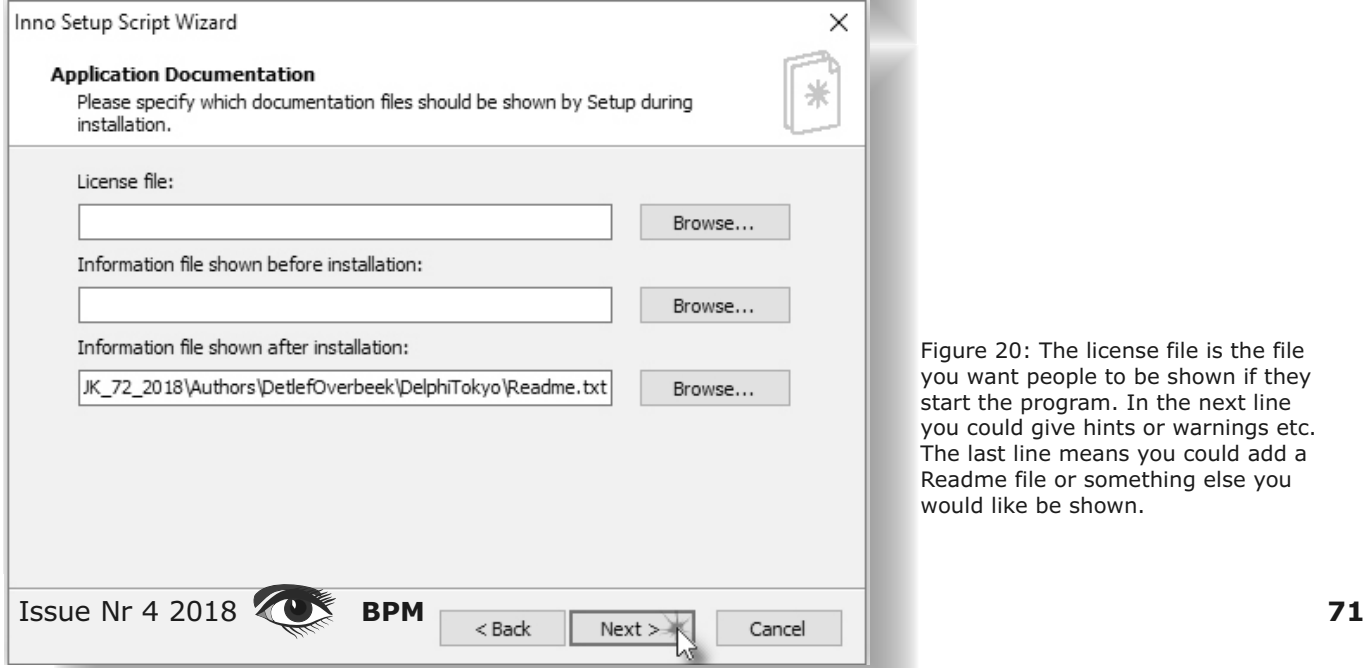


Figure 20: The license file is the file you want people to be shown if they start the program. In the next line you could give hints or warnings etc. The last line means you could add a Readme file or something else you would like be shown.

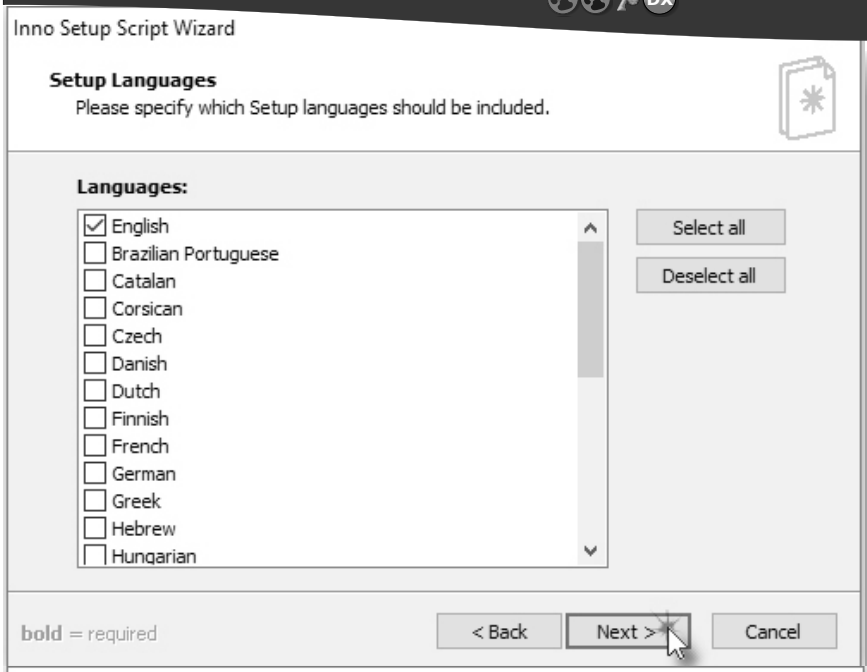


Figure 21:
languages of the installer to be chosen.
It has nothing to do with your program.
It could be the English installer and the program could be Dutch.

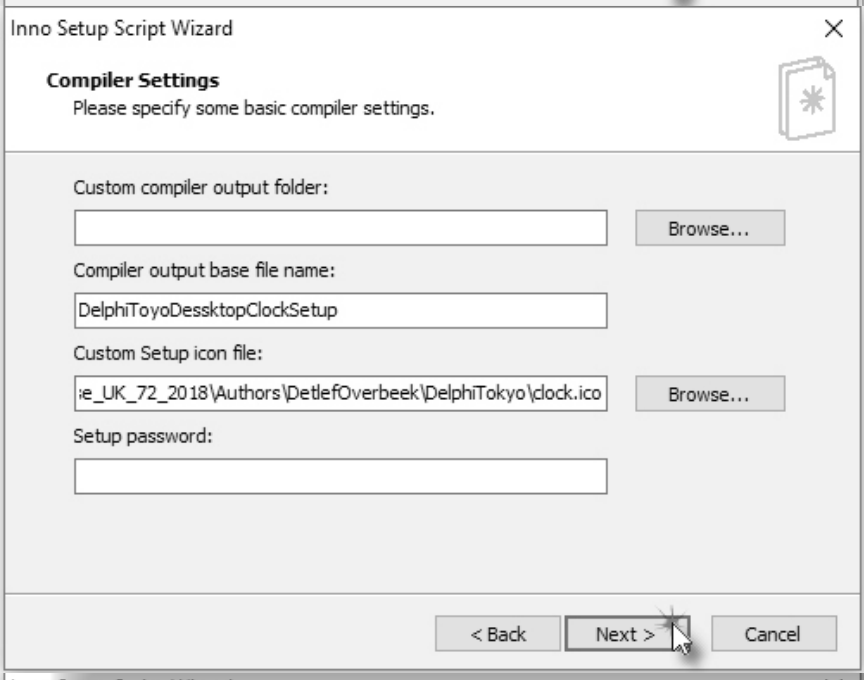


Figure 22:
Custom compiler output file.
Don't use these unless you are experienced.
Compiler output base file name
The name you give the project
The icon file is the icon that will appear under start of windows.

Figure 23: You'd better say yes

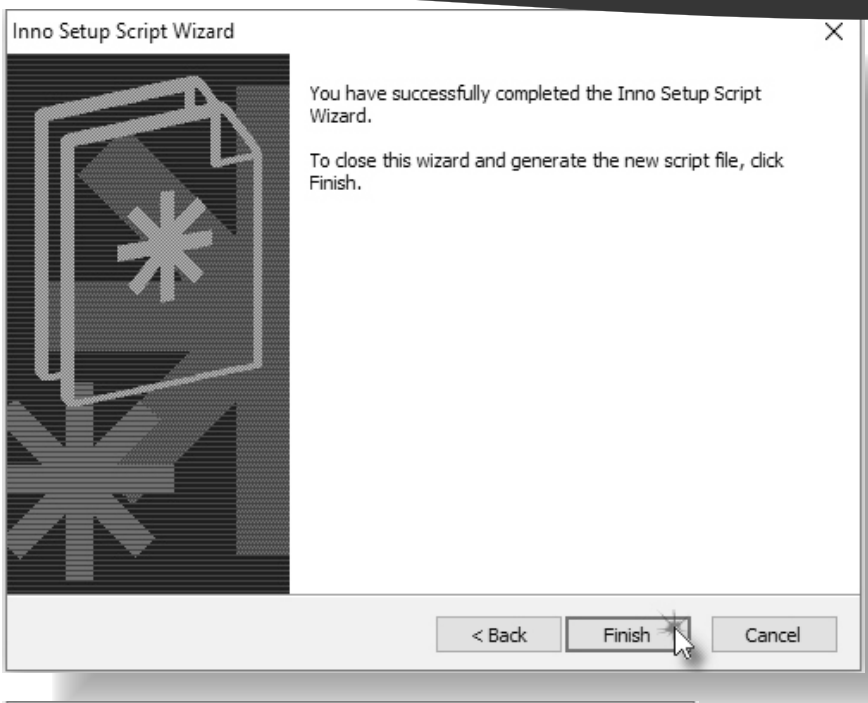


Figure 24: The setup of the script is ready to be compiled

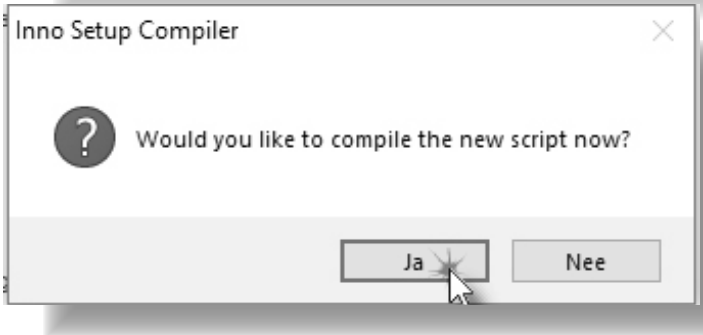


Figure 25: Start compiling

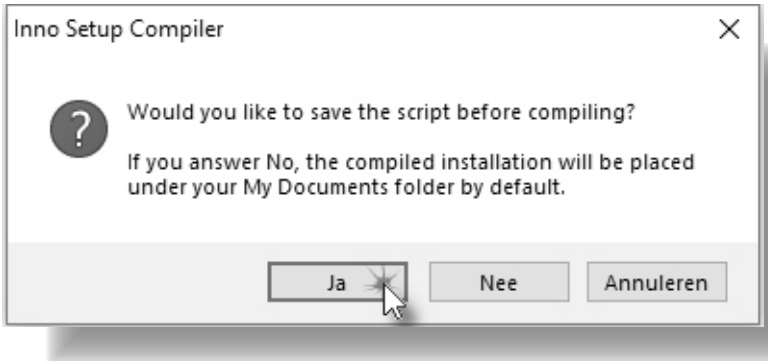


Figure 26: If you want to save it there will come a saving window.
In case you don't want that it will be saved in the Mydocuments folder on your Disk

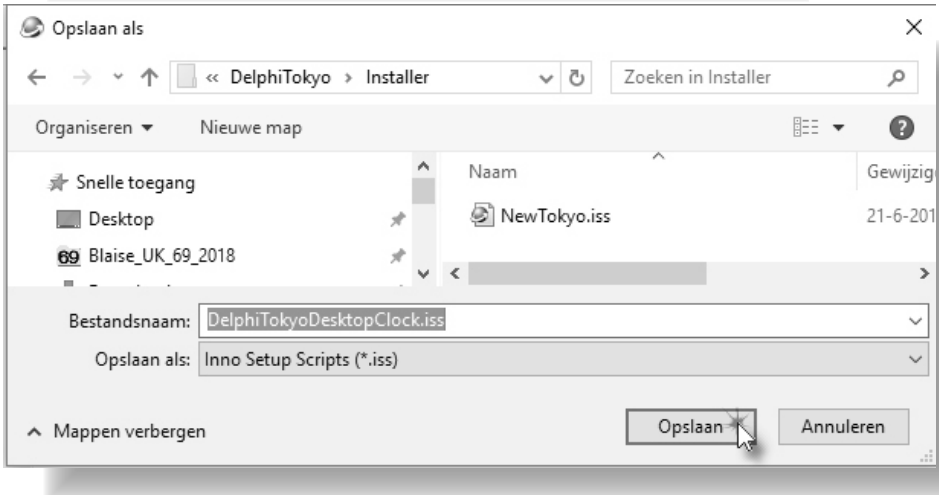


Figure 27: I created a Directory „Installer" in my Project of Delphi Tokyo

DelphiTokyoDesktopClock.iss - Inno Setup Compiler 5.6.1 (a)

File Edit View Build Run Tools Help

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "Delphi Tokyo Desktop Clock Installer"
#define MyAppVersion "00.1"
#define MyAppPublisher "Blaise Pascal Magazine"
#define MyAppURL "https://www.blaisepascalmagazine.eu/"
#define MyAppExeName "DelphiToky0Project1.exe"

[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{A33AA215-48AF-4DFB-9B43-748CBE9BF2C1}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\BlaisePascalMagazine
DisableProgramGroupPage=yes
InfoAfterFile=F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\Readme.txt
OutputBaseFilename=DelphiToyoDessktopClockSetup
SetupIconFile=F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\clock.ico
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\DelphiToky0Project1.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\appINI.ini"; DestDir: "{app}"; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon

[Run]
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait postinstall skipifsilent
```

```
*** Starting compile.  [21:59:38]

[ISPP] Preprocessing.
[ISPP] Preprocessed.

Parsing [Setup] section, line 14
Parsing [Setup] section, line 15
Parsing [Setup] section, line 16
Parsing [Setup] section, line 18
Parsing [Setup] section, line 19
Parsing [Setup] section, line 20
Parsing [Setup] section, line 21
Parsing [Setup] section, line 22
Parsing [Setup] section, line 23
Parsing [Setup] section, line 24
Parsing [Setup] section, line 25
Parsing [Setup] section, line 26
Parsing [Setup] section, line 27
Parsing [Setup] section, line 28
Creating output directory: F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\Installer\Output
Reading file (InfoAfterFile)
Reading file (WizardImageFile)
   File: C:\Program Files (x86)\Inno Setup 5\WIZMODERNIMAGE.BMP
Reading file (WizardSmallImageFile)
   File: C:\Program Files (x86)\Inno Setup 5\WIZMODERNSMALLIMAGE.BMP
Preparing Setup program executable
   Updating icons (SETUP.E32)
Reading default messages from Default.isl
Parsing [Languages] section, line 31
   File: C:\Program Files (x86)\Inno Setup 5\Default.isl
Parsing [LangOptions], [Messages], and [CustomMessages] sections
   Messages in script file
Reading [Code] section
Parsing [Tasks] section, line 34
Parsing [Icons] section, line 42
Parsing [Icons] section, line 43
Parsing [Run] section, line 46
Parsing [Files] section, line 37
Parsing [Files] section, line 38
Creating setup files
   Updating icons (SETUP.EXE)
   Compressing: F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\DelphiToky0Project1.exe
   Compressing: F:\SPP\Blaise\Blaise_UK_72_2018\Authors\DetlefOverbeek\DelphiTokyo\appINI.ini
   Compressing Setup program executable
   Updating version info

*** Finished.  [21:59:43, 00:05,015 elapsed]

Compiler Output | Debug Output
42: 119    Insert
```

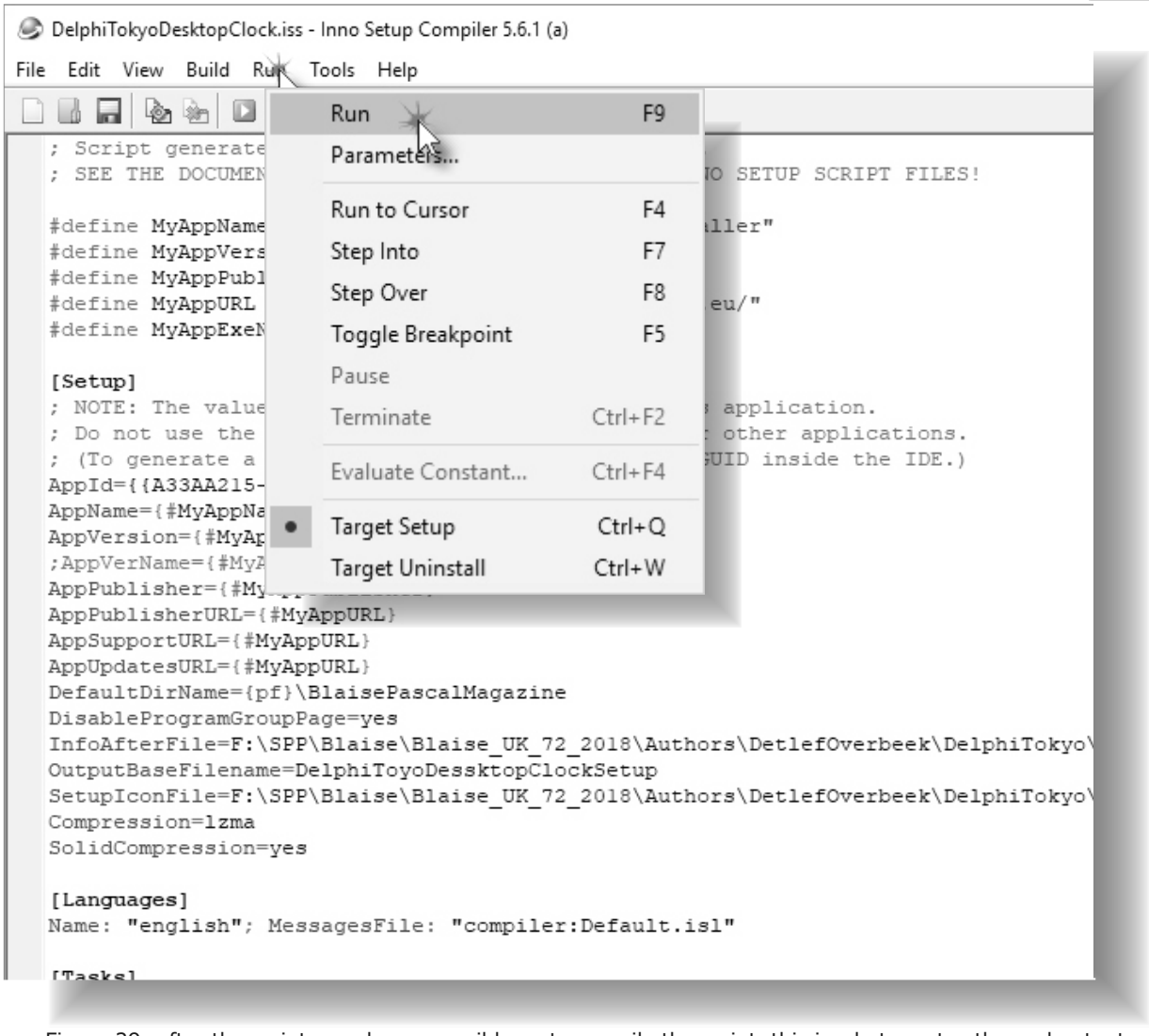


Figure 29: after the script was done you wil have to compile the script, this is what creates the real output

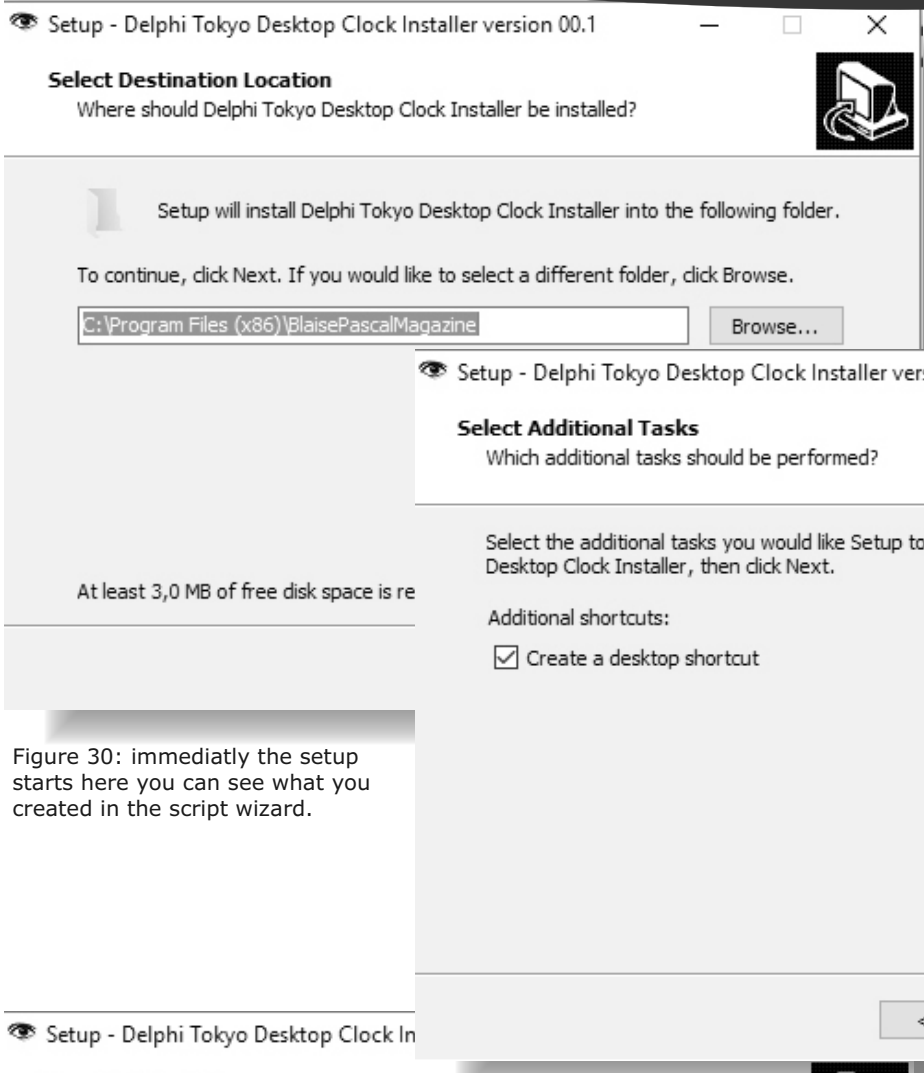


Figure 30: immediatly the setup starts here you can see what you created in the script wizard.

Figure 31:
You can create a desktop shortcut.
Sometimes it's anice feature.
You can of course always delete it later.

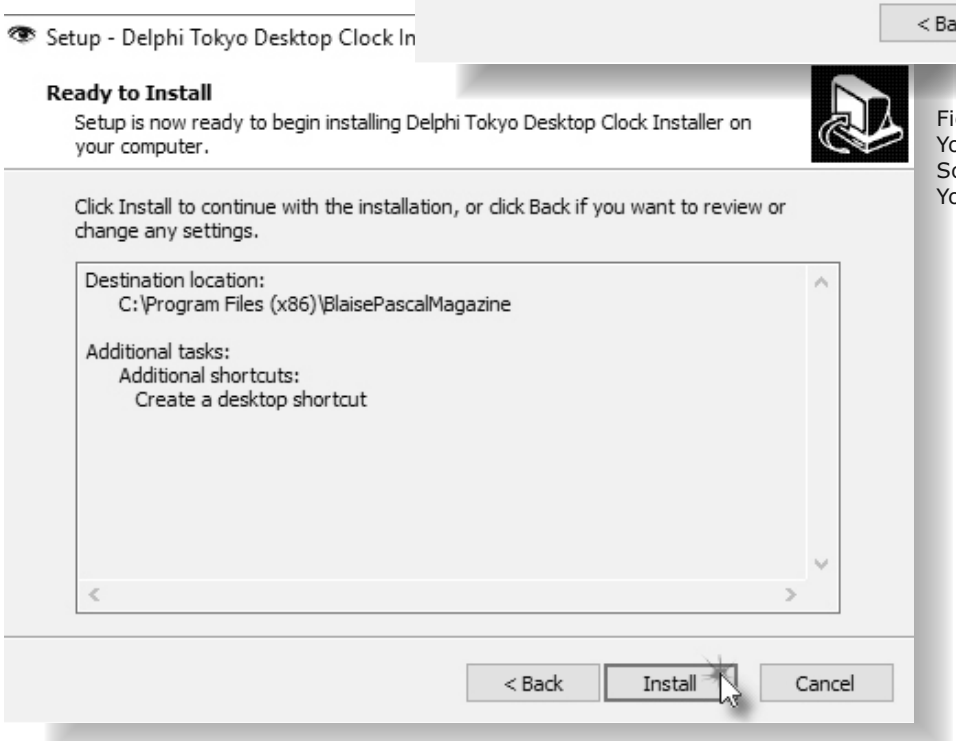


Figure 32: Installation starts...

AUTOSTART

Now for this little app I wanted it to be started automatically. That is not as easy as it ever was before: WINDOWS 10 changed it policies for autostartup and it simply does not work anymore. However its possible to get it done with some more effort:

1. Hold the windows key+R and bring up the run dialog box and type in "regedit" .
Hit enter or go for the taskbar and search:

2. Navigate to this registry key location:
„hkey_local_machine\software\wow6432node\microsoft\windows\currentversion\run"

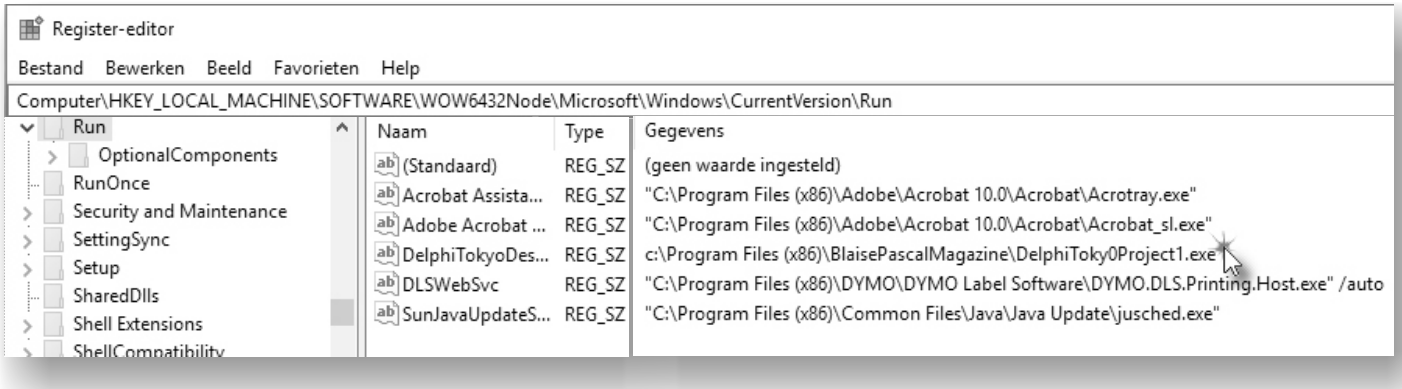
3. Right click on the right side of the panel and go to
New->String value
and that will create a new string value that you can rename to the name of the program you want to run ("Name of the app" in this case)

4. Double click your new string value and under where it says "value data" type/paste in the dialog box the path to the program (*copying it from the desktop shortcut is the easiest*) .
In my case, my DelphiToky0Project1 is in:
„c:\Program Files (x86)\BlaisePascalMagazine\DelphiToky0Project1.exe"

5. That's it! You're done. And now you can restart your PC to check and see that the desired program now starts up automatically with Windows 10!

https://www.privateinternetaccess.com/forum/discussion/18237/fixed-windows-10-automatic-startup-issue.

In the next issue I will try to solve the next question:
how do I create an installer that solves even this auto start problem?



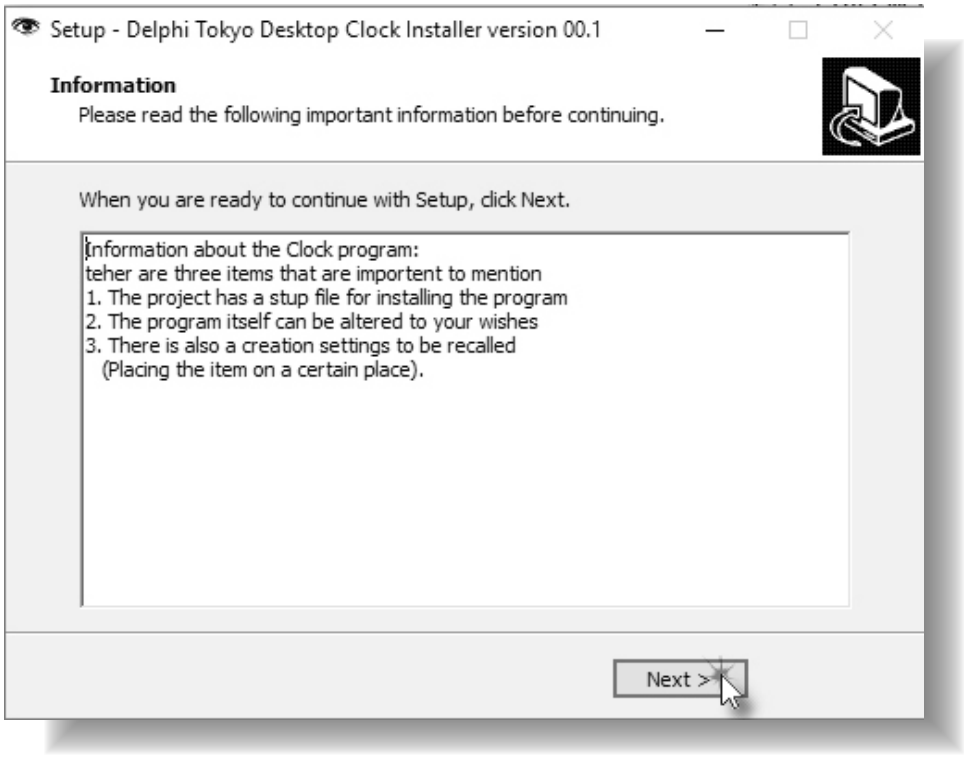


Figure 33: Read the above carefully

Figure 34: Finally launch your program, try it on any other Windows 7-10 computer to test it.

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.06 BETA RELEASED!

NEW! TKBMMWISAPIRESTSERVERTRANSPORT REST CAPABLE ISAPI SERVER SIDE TRANSPORT.

- **RAD Studio 10.2 Tokyo support including Linux support** (in beta).
- **Huge number of new features** and improvements!
- **New Smart services and clients** for very easy publication of functionality and use from clients and REST aware systems without any boilerplate code.
- **New ORM OPF** (Object Relational Model Object Persistence Framework) to easy storage and retrieval of objects from/to databases.
- **New high quality random functions.**
- **New high quality pronouncable password generators.**
- **New support for YAML, BSON, Messagepack** in addition to JSON and XML.
- **New Object Notation framework which JSON, YAML, BSON and Messagepack** is directly based on, making very easy conversion between these formats and also XML which now also supports the object notation framework.
- **Lots of new object marshalling improvements,** including support for marshalling native Delphi objects to and from YAML, BSON and Messagepack in addition to JSON and XML.
- **New LogFormatter support** making it possible to customize actual logoutput format.
- **CORS support in REST/HTML services.**
- **High performance HTTPSys transport for Windows.**
- Focus on central performance improvements.
- Pre XE2 compilers no longer officially supported.
- Bug fixes
- **Multimonitor** remote desktop V5 (VCL and FMX)
- RAD Studio and Delphi XE2 to 10.2 Tokyo support
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support!
- **Native PHP**, Java, OCX, ANSI C, C#, Apache Flex client support!
- **High performance LZ4 and Jpeg compression**
- **Native high performance** 100% developer defined app server with support for loadbalancing and failover
- **Native improved XSD importer** for generating marshal able Delphi objects from XML schemas.
- **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
- **Full FastCGI hosting support.** Host PHP/Ruby/Perl/Python applications in kbmMW!
- **Native AMQP support** (Advanced Message Queuing Protocol) with AMQP 0.91 client side gateway support and sample.
- **Fully end 2 end secure brandable Remote Desktop** with near REALTIME HD video, 8 monitor support, texture detection, compression and clipboard sharing.
- **Bundled kbmMemTable Professional** which is the fastest and most feature rich in memory table for Embarcadero products.

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

COMPONENTS 4 DEVELOPERS

EESB, SOA,MoM, EAI TOOLS FOR INTELLIGENT SOLUTIONS. kbmMW IS THE PREMIERE N-TIER PRODUCT FOR DELPHI / C++BUILDER BDS DEVELOPMENT FRAMEWORK FOR WIN 32 / 64, .NET AND LINUX WITH CLIENTS RESIDING ON WIN32 / 64, .NET, LINUX, UNIX MAINFRAMES, MINIS, EMBEDDED DEVICES, SMART PHONES AND TABLETS.