Python4Delphi PART 3
Code with Python4Delphi

Author: Max Kleiner

maXbox Starter86_3

Try except end. — Max

Thanks to Python4Delphi we now can evaluate (for expressions) or exec (for statements) some Python code in our scripts. This version 4.7.5.80 July 2021 allows us with the help of a Python Dll and an environment with modules in site-packages execute Py-functions. But the most is only available in a 32-bit space as maXbox is still 32-bit, possible also with 64-bit Python means the call of the external shell (ExecuteShell) with installed Python versions to choose from. By the way also a Python4Lazarus is available.

Imagine you need a 512-bit hash and you don't have the available function. SHA256 or SHA512 is a secure hash algorithm which creates a fixed length one way string from any input data.
OK you start the Python-engine in your maXbox script and load the DLL.
Most of the time you don't need to install Python cause you find a DLL or subdirectory for example in the Wow64 subsystem or in mySQL and load it. WoW64 (Windows 32-bit on Windows 64-bit) is a subsystem of the Windows operating system capable of running 32-bit applications on 64-bit Windows. To get a Dll that fits your size and space you can check with

```
writeln('is x64 '+botostr(Isx64('C:\maXbox\EKON25\python37.dll')));
```

You do also have helper functions in the unit PythonEngine.pas as global subroutines to test the environment:

- `GetPythonEngine (Returns the global TPythonEngine)`
- `PythonOK (checks engine init)`
- `PythonToDelphi`
- `IsDelphiObject`
- `PyObjectDestructor`
- `FreeSubtypeInst`
- `PyType_HasFeature`

```
function GetPythonEngine : TPythonEngine;
function PythonOK : Boolean;
function PythonToDelphi( obj : PPyObject ) : TPyObject;
function IsDelphiObject( obj : PPyObject ) : Boolean;
procedure PyObjectDestructor( pSelf : PPyObject); cdecl;
procedure FreeSubtypeInst(ob:PPyObject); cdecl;
procedure Register;
function PyType_HasFeature(AType : PPyTypeObject; AFlag : Integer): Boolean;
function SysVersionFromDLLName(const DLLFileName : string): string;
procedure PythonVersionFromDLLName( LibName: string; out MajorVersion, MinorVersion: integer);
```

FOR EXAMPLE THE PYTHONOK:

```
function PythonOK: Boolean;
begin
  Result:= Assigned( gPythonEngine ) and
       (gPythonEngine.Initialized or gPythonEngine.Finalizing);
end;
```

Or best you install the environment with:
`https://www.python.org/ftp/python/3.7.9/python-3.7.9.exe`
Python source code and installers are available for download for all versions!
I provide also just a Dll which we use most at:
`https://sourceforge.net/projects/maxbox/files/Examples/EKON/P4D/python37.dll/download`
Search for registered versions is possible with the function
`GetRegisteredPythonVersions: TPythonVersions;`
On 64-bit Windows the 32-bit `python27.dll` is really in
`C:\Windows\sysWOW64.`
But if you try opening the
`C:\Windows\system32\python27.dll`
in a 32-bit process, it'll open just fine.
If I'm not mistaken, WOW stands for Woodoo Of Windows.

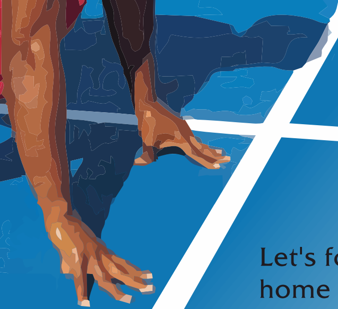
```
//if PythonVersionFromPath(PYHOME, aPythonVersion, false) then begin

 if GetLatestRegisteredPythonVersion(aPythonVersion) then begin
   aPythonVersion.AssignTo(eng);
   writeln('APIVersion: '+itoa(TPythonEngine(eng).APIVersion));
   writeln('RegVersion: '+TPythonEngine(eng).RegVersion);
   writeln('RegVersion: '+TPythonEngine(eng).DLLName);
   //TPythonEngine(PythonEngine).LoadDLL;
 end;

>>>      APIVersion: 1013
         RegVersion: 3.6
         RegVersion: python36.dll
```

To make sure your install path of Python is the right one test it with OpenDll() passing the path and call explicitly OpenDll():

```
procedure TDynamicDll.LoadDll;
begin
   OpenDll( DllName );
end;
   eng.dllpath:= 'C:\maXbox\EKON25'
   eng.dllname:= 'python37.dll';
   eng.AutoLoad:= false;
  try
   eng.OpenDll('C:\maXbox\EKON25\python37.dll');
```



Let's follow the **Sha512** example as our topic and then you type the path, home and name of the Dll the given way:

```
with TPythonEngine.create(self) do begin
  //Config Dll or Autoload
  pythonhome:= PYHOME;
  LoadDll;
  writeln(pythonhome)
  writeln(ExecModule)
  pypara:= 'https://en.wikipedia.org/wiki/WoW64';
  //pypara:= filetostring(exepath+'maXbox4.exe')
  try
    writeln(evalstr('__import__("math").sqrt(45)'));
    writeln(evalstr('__import__("hashlib").sha256(b"'+
                     pypara+'").hexdigest().upper()'));
    writeln(evalstr('__import__("hashlib").sha512(b"'+
                     pypara+'").hexdigest().upper()'));
  except
    eng.raiseError;
    writeln(ExceptionToString(ExceptionType, ExceptionParam));
  finally
    free
  end;
end;
```

A better way would be to open the hashing file with **`evalstr()`** and open itself, so we open with with open!:

```
eng.Execstring('with open(r"'+exepath+'maXbox4.exe", "rb") as afile:'+
          ' fbuf = afile.read()');
println(eng.evalstr('__import__("hashlib").algorithms_available'));
println(eng.evalstr('__import__("hashlib").sha512('+
                    'fbuf).hexdigest().upper()'));
println(eng.evalstr('__import__("hashlib").sha1(fbuf).hexdigest().upper()'));

>>> 72342518C27207099612...
>>> 3E38A48072D4F828A4BE4A52320F092FE50AE9C3
```

So the second last line is the **Sha512** and the result is: `72342518C272070...` and so on. The important thing is the **`evalstr()`** function. The **`eval()`** allows us to execute arbitrary strings as **Python** code.
It accepts a source string and returns an object. But we can also import modules with the usefule inbuilt syntax 'import(“hashlib”)'.

Note that in **Python GUI by Python4maXbox,** to print the result, you just need to state the inbuilt print() or println() or writeln function, it’s not enough just by return statement. The output is re-

The eval() is not just limited to simple expression. We can execute functions, call methods, reference variables and so on. So we use this by using the __import__() built-in function. Note also that the computed hash is converted to a readable hexadecimal string by hexdigest().upper() and uppercase the hex-values in one line, amazing isn't it.
We step a bit further to exec a script in a script! If we call a file or an const Python command then we use ExecString(PYCMD); The script you can find at:
http://www.softwareschule.ch/examples/pydemo3.txt

The essence is a bit of script as a const:

```
const PYCMD = 'print("this is box")'+LB+
        'import sys'+LB+
        'f=open(r"1050pytest21_5powers.txt","w")'+LB+
        'f.write("Hello PyWorld_mX47580, \n")'+LB+
        'f.write("This data will be written on the file.")'+LB+
        'f.close()';
```

The LB = CR+LF; is important because we call it like a file or stream and exec() is cleaning (delete CR) and encoding the passing script afterwards, LF alone is also sufficient:

```
writeln('ExecSynCheck1 '+botostr(eng.CheckExecSyntax(PYCMD)));
eng.ExecString(PYCMD);
```

We also check the syntax before eval to prevent an exception like this: Exception:
`Access violation at address 6BA3BA66 in module 'python36.dll'`. or
`'python37_32.dll' Read of address 000000AD.`
Free the engine means destroying it calls Py_Finalize, which frees all memory allocated by the Python Dll.
Or, if you're just using the Python API without the VCL wrappers like we do, you can probably just call Py_NewInterpreter on your TPythonInterface object to get a fresh execution environment without necessarily discarding everything done before!

By success of execute PYCMD a file (1050pytest21.txt) is written with some text so we executed line by line the PYCMD. When an application uses the SysUtils unit, most runtime errors are automatically converted into exceptions. Many errors that would otherwise terminate an application – such as insufficient memory, division by zero, and general protection faults – can be caught and handled by raiseError(). This is now the whole tester Procedure PYLaz_P4D_Demo3; but my key takeaway is that only use eval() with a trusted source!

```
Procedure PYLaz_P4D_Demo3;
//https://wiki.freepascal.org/Python4Delphi
var eng : TPythonEngine; out1: TPythonGUIInputOutput;
begin
 eng:= TPythonEngine.Create(Nil);
 out1:= TPythonGUIInputOutput.create(nil)
 out1.output:= pyMemo; //debugout.output;  //memo2;
 out1.RawOutput:= False;
 out1.UnicodeIO:= False;
 out1.maxlines:= 20;
 out1.displaystring('this string thing king')
 //eng.IO:= Out1;
 Out1.writeline('draw the line');
 try
  eng.LoadDll;
  eng.IO:= Out1;
  if eng.IsHandleValid then begin
   writeln('DLLhandle: '+botostr(eng.IsHandleValid))
   WriteLn('evens: '+ eng.EvalStringAsStr('[x**2 for x in range(15)]'));
   WriteLn('gauss: '+ eng.EvalStringAsStr('sum([x for x in range(101)])'));
   WriteLn('gauss2: '+ eng.EvalStr('sum([x % 2 for x in range(10100)])'));
   WriteLn('mathstr: '+ eng.EvalStr('"py " * 7'));
   WriteLn('builtins: '+ eng.EvalStr('dir(__builtins__)'));
   WriteLn('upperstr: '+ eng.EvalStr('"hello again".upper()'));
   WriteLn('workdir: '+ eng.EvalStr('__import__("os").getcwd()'));

   eng.ExecString('print("powers:",[x**2 for x in range(10)])');
   writeln('ExecSynCheck1 '+botostr(eng.CheckExecSyntax(PYCMD)));
   eng.ExecString(PYCMD);
   writeln('ExecSynCheck2 '+botostr(eng.CheckExecSyntax(myloadscript)));
   writeln('ExecSynCheck3 '+
      botostr(eng.CheckExecSyntax(filetostring(PYSCRIPT))));
   //eng.ExecString(filetostring(PYSCRIPT));
   writeln(eng.Run_CommandAsString('print("powers:",[x**2 for x in
                                   range(10)])',eval_input));
   pymemo.update;
  end
   else writeln('invalid library handle! '+Getlasterrortext);
  writeln('PythonOK: '+botostr(PythonOK));
 except
  eng.raiseError;
  writeln('PyErr '+ExceptionToString(ExceptionType, ExceptionParam));
 finally
  eng.free;
 end;
 out1.free;
 //pyImport(PyModule);
end;
```

The procedure raiseError helps to find errors for example: Exception: : SRE __main__ module mismatch. Make sure you do not have any mismatch between Python interpreter version used (like 3.7) and the “re” python module (like 3.6.1). By the way the resolution of Dlls has changed in Python 3.8 for Windows. New in version 3.8: Previous versions of CPython would resolve Dlls using the default behavior for the current process. This led to inconsistencies, such as only sometimes searching PATH or the current working directory, and OS functions such as AddDllDirectory having no effect.

Conclusion: The eval() method parses the expression passed to it and runs python expression(code) (but no statements) within the program. For you and for me 5 functions are crucial:

```
Function CheckEvalSyntax(const str: AnsiString):Boolean);
Function CheckExecSyntax(const str: AnsiString):Boolean);
Procedure ExecString(const command: AnsiString););
Procedure ExecString3(const command: AnsiString););//alias
Procedure ExecStrings4(strings: TStrings););
Function EvalStringAsStr(const command: AnsiString):string);//alias
Function EvalStr(const command: AnsiString): string);
```

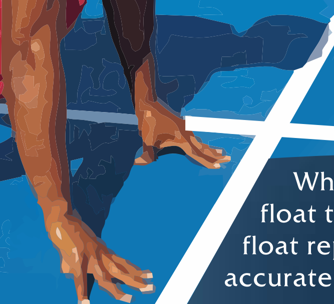
Also, consider a situation when you have imported os module in your python program like above WriteLn(‘workdir: ‘+ eng.EvalStr(‘import(“os”).getcwd()’));. The os module provides portable way to use operating system functionalities like: read or write a file. But a single command can delete all files in your system!

So eval expects an expression, import is a statement. That said, what you can trying is the following combination:

```
Println('exec as eval: '+eng.EvalStr('exec("import os as o")'));
Println('exec: '+eng.EvalStr('o.getcwd()'));
//>>> exec as eval: None
//>>> exec: C:\maXbox\mX47464\maxbox4
writeln('uuid: '+eng.evalstr('exec("import uuid") or str(uuid.uuid4())'));
//>>> uuid: 3b2e10f9-0e31-4961-9246-00852fd508bd
```

You can use exec in eval instead if you intend to import the module or also ExecString(): it depends on the global or local namespace you set, means also the second line knows the import statement from first line:

```
eng.ExecString('import math');
Println('evalexec: '+eng.EvalStr('dir(math)'));
```



When you use a float that doesn't have an exact binary float representation, the Decimal constructor cannot create an accurate decimal representation. For example:

And the same with an EvalExec:

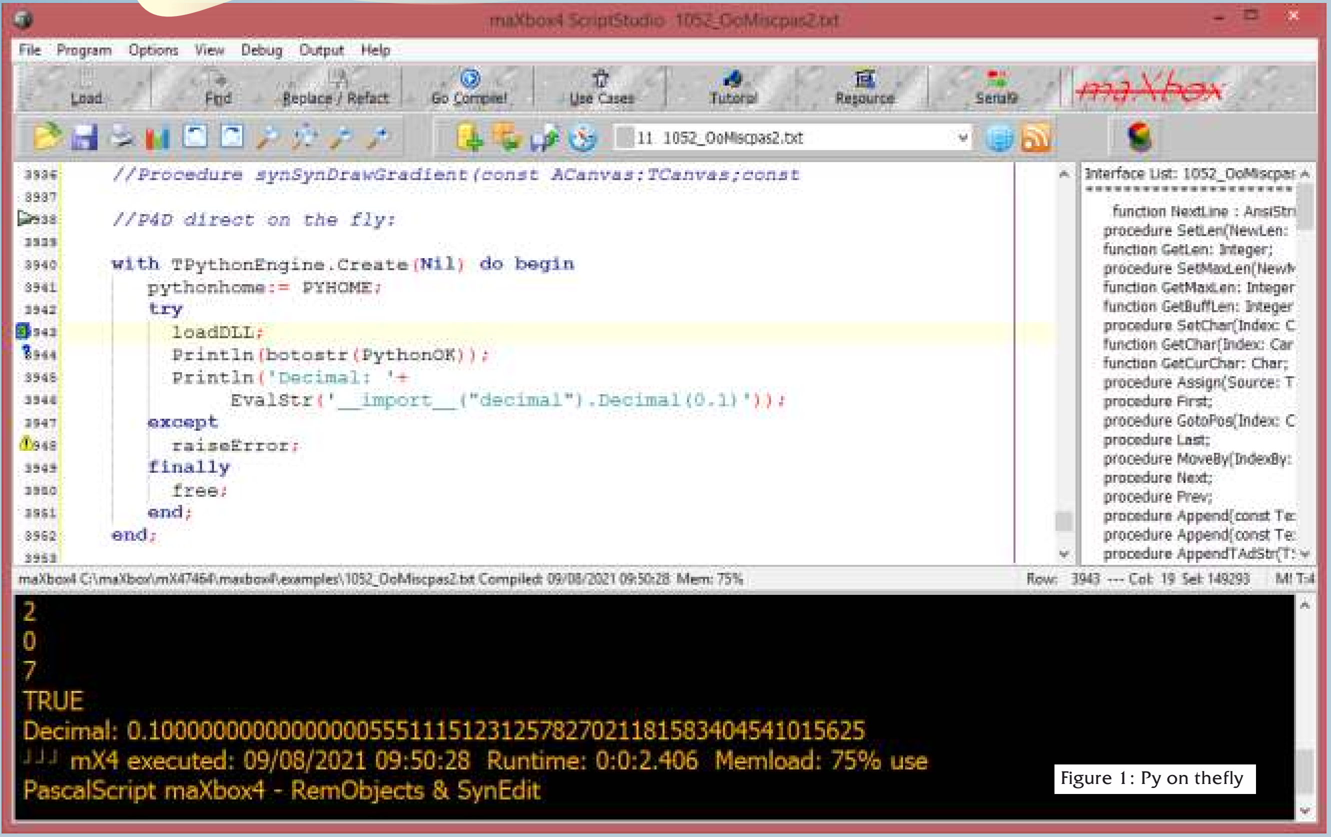
```
import decimal
from decimal import Decimal

x = Decimal(0.1)
print(x)

pymemo.lines.add('Decimal: '+
         eng.EvalStr('__import__("decimal").Decimal(0.1)'));
>>> 0.1000000000000000055511151231257827021181583404541015625
```

At last a minimal configuration called "Pyonfly". The minimal configuration depends on your Python-installation and the UseLastKnownVersion property in TDynamicDll but once known it goes like this with raiseError to get the Python exceptions:

```
with TPythonEngine.Create(Nil) do begin
 pythonhome:= PYHOME;
 try
  loadDLL;
  Println('Decimal: '+
     EvalStr('__import__("decimal").Decimal(0.1)'));
 except
  raiseError;
 finally
  free;
 end;
end;
```


maXbox4 ScriptStudio 1052_OoMiscpas2.txt
File Program Options View Debug Output Help
Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9
maXbox
11 1052_OoMiscpas2.txt

```
    //Procedure synSynDrawGradient(const ACanvas:TCanvas;const

    //P4D direct on the fly:

    with TPythonEngine.Create(Nil) do begin
      pythonhome:= PYHOME;
      try
        loadDLL;
        Println(botostr(PythonOK));
        Println('Decimal: '+
                EvalStr('__import__("decimal").Decimal(0.1)'));
      except
        raiseError;
      finally
        free;
      end;
    end;
```

```
Interface List: 1052_OoMiscpas
******************************
 function NextLine : AnsiStri
procedure SetLen(NewLen:
function GetLen: Integer;
procedure SetMaxLen(NewM
function GetMaxLen: Integer
function GetBuffLen: Integer
procedure SetChar(Index: C
function GetChar(Index: Car
function GetCurChar: Char;
procedure Assign(Source: T
procedure First;
procedure GotoPos(Index: C
procedure Last;
procedure MoveBy(IndexBy:
procedure Next;
procedure Prev;
procedure Append(const Te:
procedure Append(const Te:
procedure AppendTAdStr(T:
```

```
maXbox4 C:\maXbox\mX47464\maxbox4\examples\1052_OoMiscpas2.txt Compiled: 09/08/2021 09:50:28 Mem: 75%     Row: 3943 --- Col: 19 Sel: 149293   MI T:4
```

```
2
0
7
TRUE
Decimal: 0.1000000000000000055511151231257827021181583404541015625
 mX4 executed: 09/08/2021 09:50:28  Runtime: 0:0:2.406  Memload: 75% use
PascalScript maXbox4 - RemObjects & SynEdit
```

Figure 1: Py on thefly

The unit PythonEngine.pas is the main core-unit of the framework. Most of the Python/C API is presented as published/public member functions of the engine unit.

```
...
 Py_BuildValue              := Import('Py_BuildValue');
 Py_Initialize              := Import('Py_Initialize');
 PyRun_String               := Import('PyRun_String');
 PyRun_SimpleString         := Import('PyRun_SimpleString');
 PyDict_GetItemString       := Import('PyDict_GetItemString');
 PySys_SetArgv              := Import('PySys_SetArgv');
 Py_Exit                    := Import('Py_Exit');...
```

TIPS:

NOTE: You will need to adjust the demos from github or sourceforge accordingly, to successfully load the Python distribution that you have installed on your computer so here's a small troubleshooter:

❶ Set a path first:

```
pydllpath= ‘C:\Users\breitsch\AppData\Local\Programs\Python\Python37-32\python37.dll’;
```

❷ Load it: `pythonengine.openDll(pydllpath);`

❸ Test it: `PrintLn(‘builtins: ‘+ pythonengine.EvalStr(‘dir(__builtins__)’));`

If you get the error:

```
    Exception: :DLL load failed: %1 is not a valid Win32 application.
```

a solution is to set the `pythonhome` to 32bit:

```
    PYHOME = ‘C:\Users\max\AppData\Local\Programs\Python\Python36-32\’;
    eng.pythonhome:= PYHOME;
```

Be sure that Pyhome and Pydll are of the same filespace when installing a package,e.g. install from within script, ex. numpy:

```
eng.ExecString(‘import subprocess’);
eng.ExecString(‘subprocess.call([“pip”, “install”, “numpy”])’)
eng.ExecString(‘import numpy’);
```

Another complete 4 liner for environment test:

```
eng.ExecString(‘import subprocess’);
eng.ExecString(‘subprocess.call([“pip”, “install”, “langdetect”])’)
eng.ExecString(‘from langdetect import detect’);
println(‘detect: ‘+eng.EvalStr(‘detect(“bonjour mes ordinateurs”)’));
>>> detect: fr
```

IMPORTANT NOTE:

You should never pass untrusted source to the eval() directly. As it is quite easy for the malicious user to wreak havoc on your system. For example, the following code can be used to delete all the files from the system: `eval('os.system("RM -RF /")')`

WIKI & EKON P4D TOPICS

- https://entwickler-konferenz.de/delphi-innovations-fundamentals/python4delphi/
- http://www.softwareschule.ch/examples/weatherbox.txt
- https://learndelphi.org/python-native-windows-gui-with-delphi-vcl/

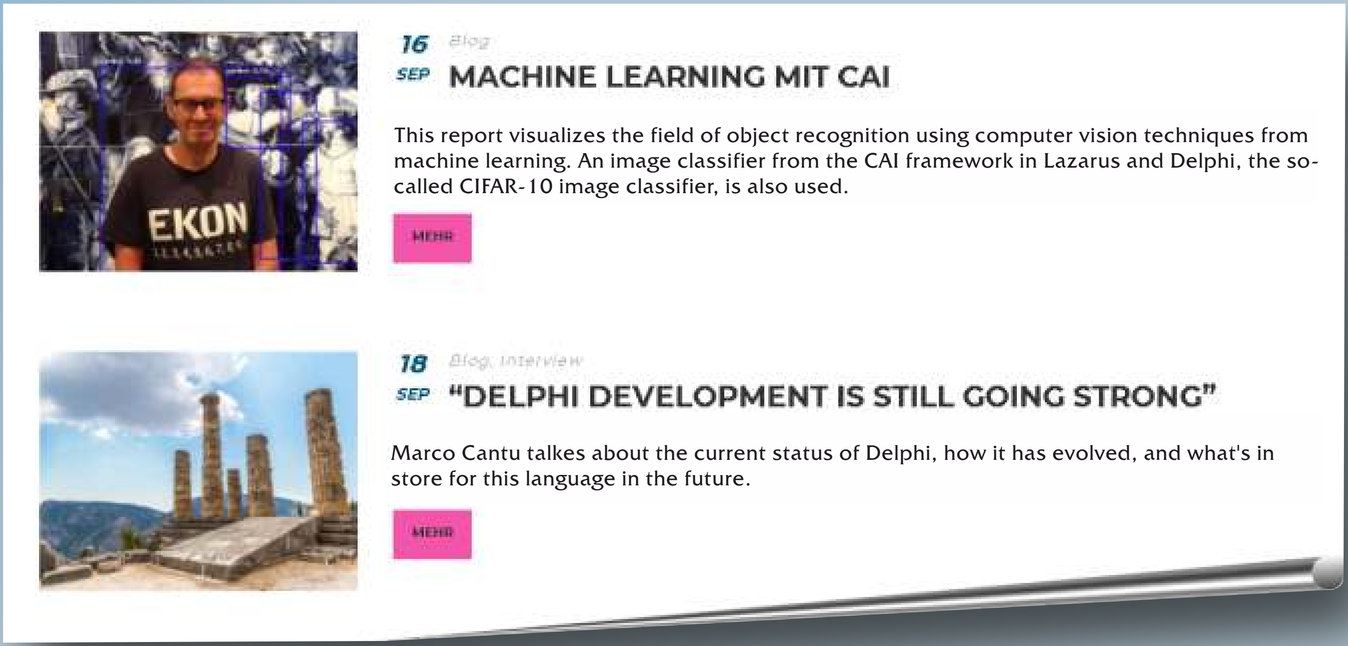
LEARN ABOUT PYTHON FOR DELPHI

- Tutorials
- Demos https://github.com/maxkleiner/python4delphi

Note: You will need to adjust the demos from github accordingly, to successfully load the Python distribution that you have installed on your computer.

Docs:
https://maxbox4.wordpress.com/blog/
http://www.softwareschule.ch/download/maxbox_starter86.pdf
http://www.softwareschule.ch/download/maxbox_starter86_1.pdf
http://www.softwareschule.ch/download/maxbox_starter86_2.pdf

https://entwickler-konferenz.de/location-en/



16 SEP Blog

MACHINE LEARNING MIT CAI

This report visualizes the field of object recognition using computer vision techniques from machine learning. An image classifier from the CAI framework in Lazarus and Delphi, the so-called CIFAR-10 image classifier, is also used.

18 SEP Blog, Interview

"DELPHI DEVELOPMENT IS STILL GOING STRONG"

Marco Cantu talkes about the current status of Delphi, how it has evolved, and what's in store for this language in the future.