//

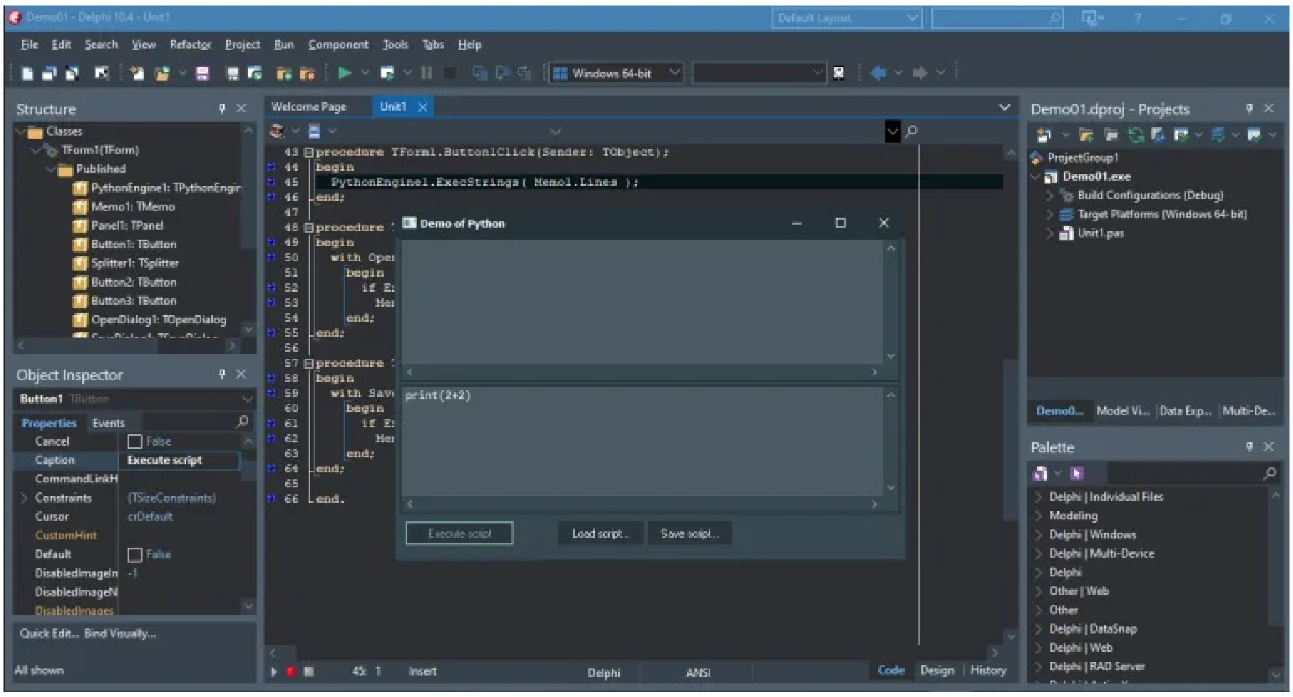
Python4Delphi II

maXbox Starter86_2 – Code with Python4Delphi

Be yourself; Everyone else is already taken. — Oscar Wilde.

In the last Article we have seen that P4D is a set of free components that wrap up the Python DLL into Delphi and Lazarus (FPC). For the next section I want to show more practical implementations. Let's start with P4D in Delphi:

```
    First create a new Form
    Drop a TMemo (or a TRichEdit)
    Drop a TPythonGUIInputOutput for displaying Python's results
    Drop a TMemo for source code
    Drop a TPythonEngine
    Connect the attribute IO of the TPythonEngine to
                                        TPythonGUIInputOutput.
    Connect attribute Output of TPythonGUIInputOutput to
                                        TRichEdit.
    Drop a TButton and call it "Execute script"
    Double-click on the button and add:
         PythonEngine1.ExecStrings(Memo1.Lines);
    That's almost all!
    Compile and execute.
    Write in the Memo1: print(2+3)
    Click on the Execute button
    You should see in the Output as Memo2 window: 5
```



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PythonEngine1.ExecStrings( Memo1.Lines );
end;
```

```
print(2+2)
```

_PIC: p4d_d10_4.png

As we can see the memo-control manifests the Python-script as input in memo1 and output in memo2:

```
object Memo1: TMemo
 ...
  Font.Pitch = fpVariable
  Font.Style = []
  Lines.Strings = (
            'print(2+3)')
  ParentFont = False
  ScrollBars = ssBoth
  TabOrder = 1
end

object PythonGUIInputOutput1: TpythonGUIInputOutput

  UnicodeIO = True

  RawOutput = False

  Output = Memo2

  Left = 64

end
```

So in a more complicated script we do have a same memo-control but simply with more lines:

```
    Lines.Strings = (
      'import sys'
      'print ("Version:", sys.version)'
      'import spam'
      'print (spam.foo('#39'hello world'#39', 1))'
      'p = spam.CreatePoint( 10, 25 )'
      'print ("Point:", p)'
      'p.x = 58'
      'print (p.x, p)'
      'p.OffsetBy( 5, 5 )'
      'print (p)'
      'print ("Current value of var test is: ", test)'
      'test.Value = "New value set by Python"'
      'print (spam.getdouble())'
      'print (spam.getdouble2())')
    ParentFont = False
```

You do also have the evaluation of an expression. But the evaluation of an expression works only for arithmetic expressions and not for instructions ! The use of variables and functions is of course possible but constructs like for, def, catch, class, print, import... are not implemented, you use for this *ExecStrings()* and not *EvalStrings()*.

Using Delphi methods as Python functions

What would be if we use in a internal Python-script some Delphi-methods like in the above script methods of the import module spam? First we had to initialize the module **spam**, we just need to add our new methods:

```
procedure TForm1.PythonModule1Initialization(Sender: TObject);
begin
  with Sender as TPythonModule do
    begin
      AddDelphiMethod( 'foo',
                       spam_foo,
                       'foo' );
      AddDelphiMethod( 'CreatePoint',
                       spam_CreatePoint,
                       'function CreatePoint'+LF+
                       'Args: x, y'+LF+
                       'Result: a new Point object' );
      AddDelphiMethod( 'getdouble',
                       spam_getdouble,
                       'getdouble' );
      AddDelphiMethod( 'getdouble2',
                       spam_getdouble2,
                       'getdouble2' );
    end;
end;
```

Ans here's the example of functions defined for the module spam in this context the function *spam_foo* with forms caption return:

```
function TForm1.spam_foo(pself, args : PPyObject): PPyObject; cdecl;
begin
  with GetPythonEngine do
    begin
      ShowMessage( 'args of foo: '+PyObjectAsString(args) );
      ShowMessage( 'Form''s caption = ' + Caption );
      Result := ReturnNone;
    end;
end;
```

Handshaking with Python arrays or tuples layout does have some complications. Normal Python arrays (as for standard CPython) are normally called “Lists”. A numpy.array type (or a mutable list) in Python is a special type that is more memory and layout efficient than a normal Python list of normal Py floating point objects.
If you want to use Delphi and access Numpy.array or list, I really suppose that the straightest way to do it would be to implement a way to export some simple straight C functions that access the Numpy.array type.
Numpy.array wraps a standard block of memory that is accessed as a native C array type. This in turn, does NOT map cleanly to Delphi array types as created by a Delphi method to Python.

Let me go deeper in that point, converting a Delphi-array or list to for example a list goes in the end with a dll-function from the Python library ('PyList_SetItem'):

```
function TPythonEngine.ArrayToPyList(const items: array of const) : PPyObject;
var
  i : Integer;
begin
  Result := PyList_New( High(items)+1 );
  if not Assigned(Result) then
    raise EPythonError.Create('Could not create a new list object');
  for i := Low(items) to High(items) do
    PyList_SetItem( Result, i, VarRecAsPyObject( items[i] ) );
end;

PyList_SetItem:function (dp:PPyObject;idx:NativeInt;item:PPyObject):integer;
cdecl;

PyList_SetItem:= Import('PyList_SetItem');
```

The other way round, as I said we can't map cleanly Python lists to Delphi array types, we get the data sort of as the base type strings from *PyObjectAsString*:

```
procedure TPythonEngine.PyListToStrings(list: PPyObject; strings: TStrings );
var
  i : Integer;
begin
  if not PyList_Check(list) then
    raise EPythonError.Create('the python object is not a list');
  strings.Clear;
  for i:= 0 to PyList_Size( list )- 1 do
    strings.Add( PyObjectAsString( PyList_GetItem( list, i ) ) );
end;
```

I think the common base type in Delphi (to export) is the array and the common base type in Python (to import) is the list. So this we can see as a proof of concept code:

```
function PythonToDelphi(obj : PPyObject ) : TPyObject;
begin
  if IsDelphiObject( obj ) then
    Result := TPyObject(PAnsiChar(obj)+Sizeof(PyObject))
  else
    raise EPythonError.CreateFmt( 'Python object "%s" is not a Delphi class',
                                  [GetPythonEngine.PyObjectAsString(obj)] );
end;
```

This exporting of Delphi-methods to use in Python-scripts works also with the creation of a dll as Demo09 Making a Python module as a dll explains (I'll show that in the Tutor III).

The Demo for the AddDelphiMethod concept you find at:

https://github.com/maxkleiner/python4delphi/blob/master/Demos/Demo07/test.py
http://py4d.pbworks.com/w/page/9174535/Wrapping%20Delphi%20Objects

More or less some external files as normal Python-scripts is also on your way. For example we call the script **test.py** and we import explicit the module spam, previously generated in Delphi:

```
import sys
print "Win version:", sys.winver
import spam
print (spam.foo('hello world', 1))
p = spam.CreatePoint( 10, 25 )
print ("Point:", p)
p.x = 58
print (p.x, p)
p.OffsetBy( 5, 5 )
print (p)
print ("Current value of var test is: ", test)
test.Value = "New value set by Python"
print (spam.getdouble())
```

Build your environment

On Win, the standard Python installer already associates the .py extension with a file type (Python.File) and gives that file type an open command that runs the interpreter (F:\Program Files\Python\python.exe "%1" %*). This is enough to make scripts executable from the command prompt. We use the python-dll as we use a windows dll. Therefore *.pyd files are dll's, but there are a few differences:
So far you have to know 3 different file types:

1. ***.py**: The norm input source code that we've written.

2. ***.pyc**: The compiled bytecode. If you import a module, py will build a *.pyc file that contains bytecode to make importing it again later easier and faster.
3. ***.pyd**: The mentioned windows dll file for Python.

If you have a DLL named foo.pyd, then it must have a function PyInit_foo(). You can then write Python "import foo", and Python will search for foo.pyd (as well as foo.py, foo.pyc) and if it finds it, will attempt to call PyInit_foo() to initialize it. Of course you do not link your .exe with foo.lib, as that would cause Windows to require the DLL to be present, we load it dynamically.

First we check our Python installation. Python provides for all user and current user installations. All user installations place the Py dll in the Windows System directory and write registry info to HKEY_LOCAL_MACHINE.
Current user installations place the dll in the install path and the registry info in HKEY_CURRENT_USER version < py 3.5.
So, for current user installations we need to try and find the install path since it may not be on the system path.

```
$IFDEF MSWINDOWS}
function IsPythonVersionRegistered(PythonVersion : string;
  out InstallPath: string; out AllUserInstall: Boolean) : Boolean;
  // The above convention was changed in Python 3.5.  Now even for all user
  // installations the dll is located at the InstallPath.
  // Also from vers.3.5 onwards 32 bit version have a suffix -32 e.g. "3.6-32"
  // See also PEP 514

var
  key: string;
  VersionSuffix: string;
  MajorVersion : integer;
  MinorVersion : integer;
begin
  Result := False;
  InstallPath := '';
  AllUserInstall := False;
  MajorVersion := StrToInt(PythonVersion[1]);
  MinorVersion := StrToInt(PythonVersion[3]);
  VersionSuffix := '';
{$IFDEF CPUX86}
  if (MajorVersion > 3) or ((MajorVersion = 3)  and (MinorVersion >= 5)) then
    VersionSuffix := '-32';
{$ENDIF}
  key:= Format('\Software\Python\PythonCore\%s%s\InstallPath',
                                                 [PythonVersion, VersionSuffix]);

  // First try HKEY_CURRENT_USER as per PEP514
  try
    with TRegistry.Create1(KEY_READ and not KEY_NOTIFY) do
      try
        RootKey := HKEY_CURRENT_USER;
        if OpenKey(Key, False) then begin
          InstallPath := ReadString('');
          Result := True;
          Exit;
        end;
      finally
        Free;
      end;
  except
    writeln(' HKEY_CURRENT_USER except');
  end;

  //Then try for an all user installation
  try
    with TRegistry.Create1(KEY_READ and not KEY_NOTIFY) do
      try
        RootKey := HKEY_LOCAL_MACHINE;
        if OpenKey(Key, False) then begin
          AllUserInstall := True;
          if (MajorVersion > 3) or ((MajorVersion = 3)
                                   and (MinorVersion >= 5)) then
            InstallPath := ReadString('');
          Result := True;
        end;
      finally
        Free;
      end;
  except
    writeln(' HKEY__LOCAL_MACHINE except');
  end;
end;
{$ENDIF}
```

In my case the path is on:
C:\Users\max\AppData\Local\Programs\Python\Python36\Lib\

Then we can simple check a first function or load on runtime the PyRun_SimpleString for our next example:

```
//if fileExistst(PYDLLPATH+ 'python37.dll';
  function getCopyRight: PChar;
      external 'Py_GetCopyright@C:\maXbox\EKON25\python37.dll stdcall';


function pyrun(command : pchar):integer;
      external 'PyRun_SimpleString@C:\maXbox\EKON25\python37.dll cdecl';

procedure pyinit;
      external 'Py_Initialize@C:\maXbox\EKON25\python37.dll cdecl';
procedure pyexit(retval: integer);
      external 'Py_Exit@C:\maXbox\EKON24\python37.dll cdecl';
```

Now we use to invoke a Python script as an embedding **const** and use the dll functionality of *Import('PyRun_SimpleString');*

To run python code direct in a maXbox, Free Pascal or whatever script you need to import just the 3 dll functions[1], above all *PyRun_SimpleStringFlags* or without flags:

```
Const PYDLLPATH = 'C:\maXbox\EKON25\';
      PYDLLNAME = 'python37.dll';
      PSCRIPTNAME = 'initpy.py';
```

This is a simplified interface to PyRun_SimpleString leaving the PyCompilerFlags* argument set to NULL. Normally the Python interpreter is initialized by Py_Initialize() so we use the same interpreter as from a shell or terminal:

*int PyRun_SimpleString(const char *command)*

```
   //function pyrun(command :pChar) :integer;
   //writeln('pyinitback: '+itoa
  pyinit();
  //retp:= 'print("hello low")'
  retp:= 'print()';
  //PyRun_SimpleString:   function( str: PAnsiChar): Integer; cdecl;
  //writeln(itoa(pyrun(retp)));
  writeln(itoa(pyrun('print("this is box")')));
  writeln(itoa(pyrun('import sys')));
  writeln(itoa(pyrun('f=open(r"C:\maXbox\maxbox4\pytest.txt","w")')));
  writeln(itoa(pyrun('f.write("Hello PyWorld_, \n")')));
  writeln(itoa(pyrun('f.write("Data will be written on the file.")')));
  writeln(itoa(pyrun('f.close()')));
```

You do also have helper functions in the unit **PythonEngine.pas** as Global Subroutines to test the environment:

1 Independent from imports and site-packages

- GetPythonEngine (Returns the global TPythonEngine)
- PythonOK
- 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;
```

To run python code integrated in a maXbox, Free Pascal, GNU Pascal or whatever script you need to import just the 3 dll functions[2], above all *PyRun_SimpleStringFlags* or without flags:

```
Const PYDLLPATH = 'C:\maXbox\EKON25\decimals';
      PYDLLNAME = 'python37.dll';
      PSCRIPTNAME = 'initpy.py';
```

This is a simplified interface to PyRun_SimpleString leaving the PyCompilerFlags* argument set to NULL. Normally the Python interpreter is initialized by Py_Initialize() so we use the same interpreter as from a shell, command or terminal.

In P4D you do invoke the mentioned memo with ExeStrings:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
  PythonEngine1.ExecStrings( Memo1.Lines );
end;
```

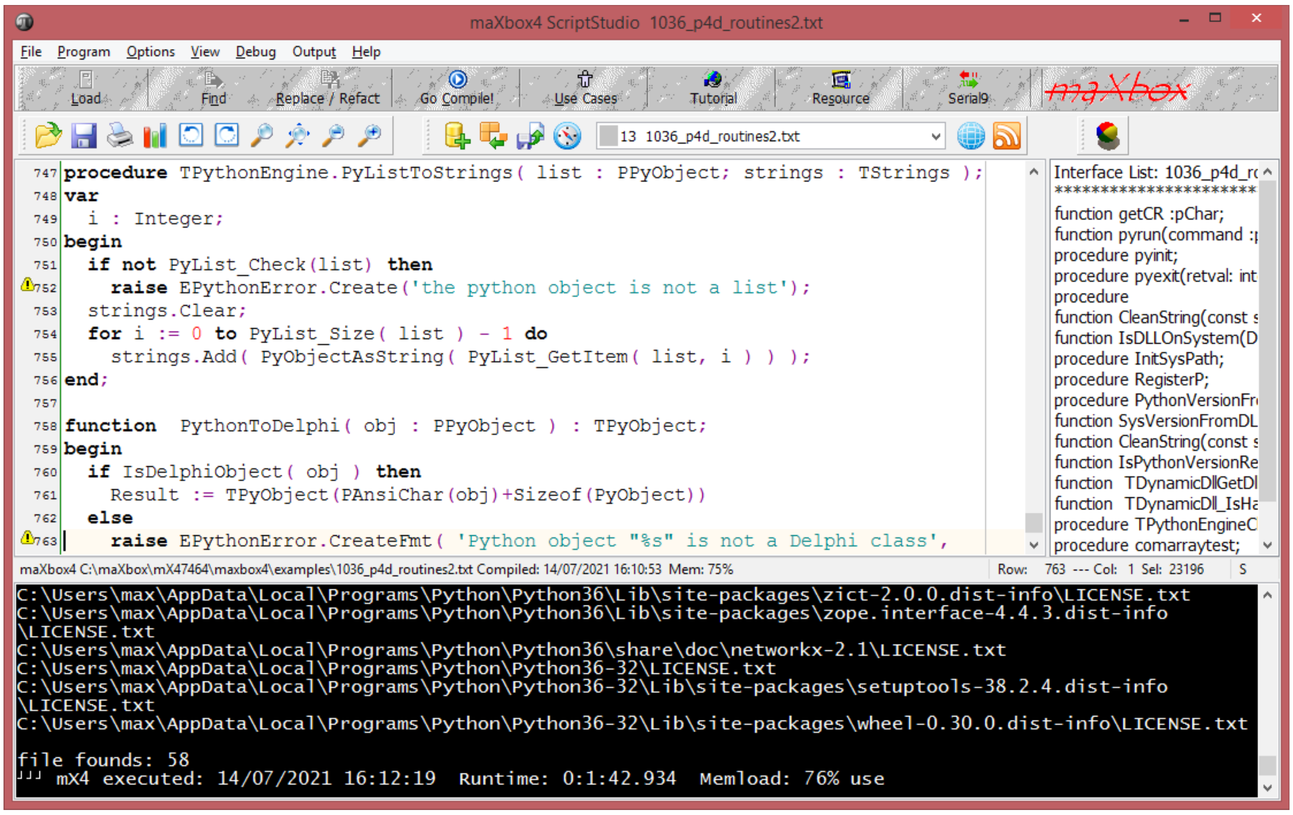
This explains best the code behind, to evaluate, run or execute an internal Python expression.

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

2 Independent from imports and site-packages

```
Println('exec as eval: '+eng.EvalStr('exec("import os as o")'));
Println('exec: '+eng.EvalStr('o.getcwd()'));
>>> exec as eval: None
>>> exec: C:\maXbox\mX47580\maxbox4
writeln('uuid: '+eng.evalstr('exec("import uuid") or
                                           str(uuid.uuid4())'));
>>> uuid: 3b2e10f9-0e31-4961-9246-00852fd508bd
```

See the demo:
http://www.softwareschule.ch/examples/pydemo.txt


maXbox4 ScriptStudio 1036_p4d_routines2.txt
File Program Options View Debug Output Help
Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9
13 1036_p4d_routines2.txt

```
procedure TPythonEngine.PyListToStrings( list : PPyObject; strings : TStrings );
var
  i : Integer;
begin
  if not PyList_Check(list) then
    raise EPythonError.Create('the python object is not a list');
  strings.Clear;
  for i := 0 to PyList_Size( list ) - 1 do
    strings.Add( PyObjectAsString( PyList_GetItem( list, i ) ) );
end;

function  PythonToDelphi( obj : PPyObject ) : TPyObject;
begin
  if IsDelphiObject( obj ) then
    Result := TPyObject(PAnsiChar(obj)+Sizeof(PyObject))
  else
    raise EPythonError.CreateFmt( 'Python object "%s" is not a Delphi class',
```

```
Interface List: 1036_p4d_r
**********************
function getCR :pChar;
function pyrun(command :
procedure pyinit;
procedure pyexit(retval: int
procedure
function CleanString(const s
function IsDLLOnSystem(D
procedure InitSysPath;
procedure RegisterP;
procedure PythonVersionFr
function SysVersionFromDL
function CleanString(const s
function IsPythonVersionRe
function  TDynamicDllGetDl
function  TDynamicDll_IsHa
procedure TPythonEngineC
procedure comarraytest;
```

```
maXbox4 C:\maXbox\mX47464\maxbox4\examples\1036_p4d_routines2.txt Compiled: 14/07/2021 16:10:53  Mem: 75%     Row:  763 --- Col:  1 Sel: 23196
```

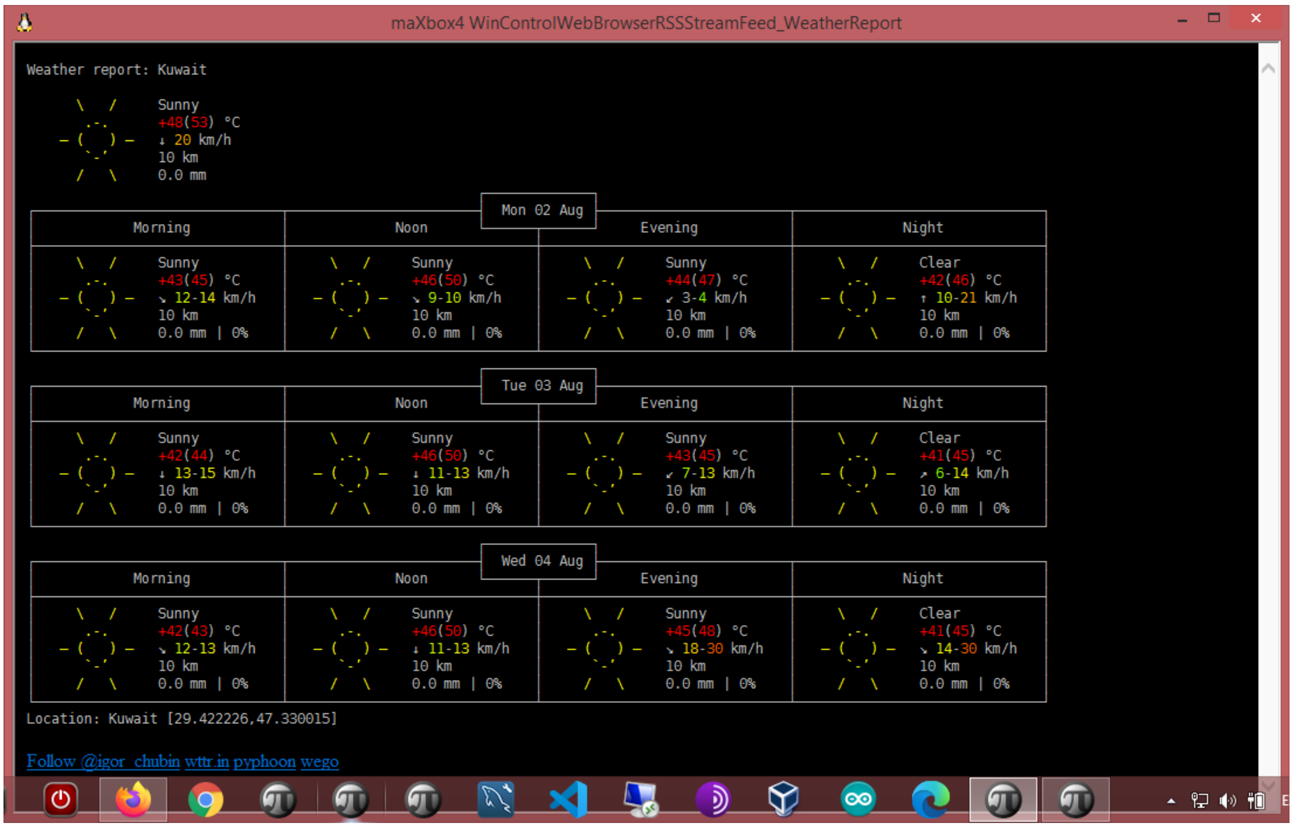
```
C:\Users\max\AppData\Local\Programs\Python\Python36\Lib\site-packages\zict-2.0.0.dist-info\LICENSE.txt
C:\Users\max\AppData\Local\Programs\Python\Python36\Lib\site-packages\zope.interface-4.4.3.dist-info
\LICENSE.txt
C:\Users\max\AppData\Local\Programs\Python\Python36\share\doc\networkx-2.1\LICENSE.txt
C:\Users\max\AppData\Local\Programs\Python\Python36-32\LICENSE.txt
C:\Users\max\AppData\Local\Programs\Python\Python36-32\Lib\site-packages\setuptools-38.2.4.dist-info
\LICENSE.txt
C:\Users\max\AppData\Local\Programs\Python\Python36-32\Lib\site-packages\wheel-0.30.0.dist-info\LICENSE.txt

file founds: 58
 mX4 executed: 14/07/2021 16:12:19  Runtime: 0:1:42.934  Memload: 76% use
```

_PIC: p4d_d10_4_pyengine.png

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 and a clever Dll loader/mapper.

```
 ...
 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');
 ...
```



maXbox4 WinControlWebBrowserRSSStreamFeed_WeatherReport

```
Weather report: Kuwait

     \   /     Sunny
      .-.      +48(53) °C
   ― (   ) ―   ↓ 20 km/h
      `-'      10 km
     /   \     0.0 mm
                                                       ┌─────────────┐
┌──────────────────────────────┬───────────────────────┤  Mon 02 Aug ├───────────────────────┬──────────────────────────────┐
│            Morning           │             Noon      └──────┬──────┘     Evening           │             Night            │
├──────────────────────────────┼──────────────────────────────┼──────────────────────────────┼──────────────────────────────┤
│     \   /     Sunny          │     \   /     Sunny          │     \   /     Sunny          │     \   /     Clear          │
│      .-.      +43(45) °C     │      .-.      +46(50) °C     │      .-.      +44(47) °C     │      .-.      +42(46) °C     │
│   ― (   ) ―   ↘ 12-14 km/h   │   ― (   ) ―   ↘ 9-10 km/h    │   ― (   ) ―   ↙ 3-4 km/h     │   ― (   ) ―   ↑ 10-21 km/h   │
│      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │
│     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │
└──────────────────────────────┴──────────────────────────────┴──────────────────────────────┴──────────────────────────────┘
                                                       ┌─────────────┐
┌──────────────────────────────┬───────────────────────┤  Tue 03 Aug ├───────────────────────┬──────────────────────────────┐
│            Morning           │             Noon      └──────┬──────┘     Evening           │             Night            │
├──────────────────────────────┼──────────────────────────────┼──────────────────────────────┼──────────────────────────────┤
│     \   /     Sunny          │     \   /     Sunny          │     \   /     Sunny          │     \   /     Clear          │
│      .-.      +42(44) °C     │      .-.      +46(50) °C     │      .-.      +43(45) °C     │      .-.      +41(45) °C     │
│   ― (   ) ―   ↓ 13-15 km/h   │   ― (   ) ―   ↓ 11-13 km/h   │   ― (   ) ―   ↙ 7-13 km/h    │   ― (   ) ―   ↗ 6-14 km/h    │
│      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │
│     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │
└──────────────────────────────┴──────────────────────────────┴──────────────────────────────┴──────────────────────────────┘
                                                       ┌─────────────┐
┌──────────────────────────────┬───────────────────────┤  Wed 04 Aug ├───────────────────────┬──────────────────────────────┐
│            Morning           │             Noon      └──────┬──────┘     Evening           │             Night            │
├──────────────────────────────┼──────────────────────────────┼──────────────────────────────┼──────────────────────────────┤
│     \   /     Sunny          │     \   /     Sunny          │     \   /     Sunny          │     \   /     Clear          │
│      .-.      +42(43) °C     │      .-.      +46(50) °C     │      .-.      +45(48) °C     │      .-.      +41(45) °C     │
│   ― (   ) ―   ↘ 12-13 km/h   │   ― (   ) ―   ↓ 11-13 km/h   │   ― (   ) ―   ↘ 18-30 km/h   │   ― (   ) ―   ↘ 14-30 km/h   │
│      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │      `-'      10 km          │
│     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │     /   \     0.0 mm | 0%    │
└──────────────────────────────┴──────────────────────────────┴──────────────────────────────┴──────────────────────────────┘
Location: Kuwait [29.422226,47.330015]
```

Follow @igor_chubin wttr.in pyphoon wego

_PIC: p4d_weathermap_kuwaitcity.png

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