//

DelphiVCL4Python

maXbox Starter92_1 - Code with VCL4Python

People lie, numbers don't.
— unknown.

In the last few Articles we have seen that P4D is a set of free components that wrap up the Python packages into Delphi and Lazarus (FPC). This time we go(t) the other way round. How can the Python World profit from the VCL Components.

We create Python extension modules from Delphi classes, records or functions. It can be the beginning of a long journey to provide Delphi's VCL library as a certain Python module to build powerful Windows GUI out from a Script.
The Python module we take a look at is called: DelphiVCL.pyd
It can be simply installed from the shell via pip:

pip install delphivcl

It supports:

    Win32 & Win64 x86 architectures
    Python cp3.6, cp3.7, cp3.8, cp3.9 and cp3.10

For other platforms, check out DelphiFMX4Python.

Another way to install is explicit with

python.exe -m pip install ***delphivcl***

in case you want to install the 32bit version with the 32bit executable. 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 (G:\Program Files\Python\python.exe "%1" %*). This is enough to

make scripts executable from the command prompt. We can use the python-dll as we use a windows dll. Therefore *.pyd files are dll-libraries, but there are a few differences:
So far you have to know 3 different file types you can import from after installed a known package like *delphivcl*:

1. ***.py**: The norm input source code that we had 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 bee.pyd, then it must have a function *PyInit_bee()*. You can then write Python "import bee", and Python will search for bee.pyd (as well as bee.py, bee.pyc) and if it finds it, will attempt to call PyInit_bee() to initialize it. Of course you don't link your *.exe with bee.lib, as that would cause Windows to require the DLL to be present, we load it dynamically at runtime.

```
import importlib.machinery, importlib.util
def new_import(ext_file):
    loader=importlib.machinery.ExtensionFileLoader("DelphiVCL",ext_file)
    spec = importlib.util.spec_from_file_location("DelphiVCL",ext_file,
        loader=loader, submodule_search_locations=None)
    #print("spec", spec, spec.loader, modulefullpath, __file__)
```

https://github.com/maxkleiner/DelphiVCL4Python/blob/main/tests/___init__.py

The project which we introduce is in the subdirectory Delphi and generates a Python extension module (a DLL with extension "pyd" in Windows) that allows you to create a user interfaces using Delphi from within Python. A part of the VCL or LCL (almost and maybe) is wrapped with a few lines of code!
The small demo *TestApp.py* gives you a flavour of what is possible. The machinery by which this is achieved is the WrapDelphi unit.

The subdirectory *DemoModule* demonstrates how to create Python extension modules using Delphi, that allow you to use in Python, functions defined in Delphi. Compile the project and run test.py from the command prompt (e.g. py test.py).
The generated pyd file should be in the same directory as the Python file. This project should be easily adapted to use with Lazarus and FPC.
After compiled to the *DelphiVCL.pyd* we want to use it in a Python script, which is the main topic of this article:

```
from delphivcl import *
```

Python code in one module gains access to code in another module by the process of importing it. The import statement is the most

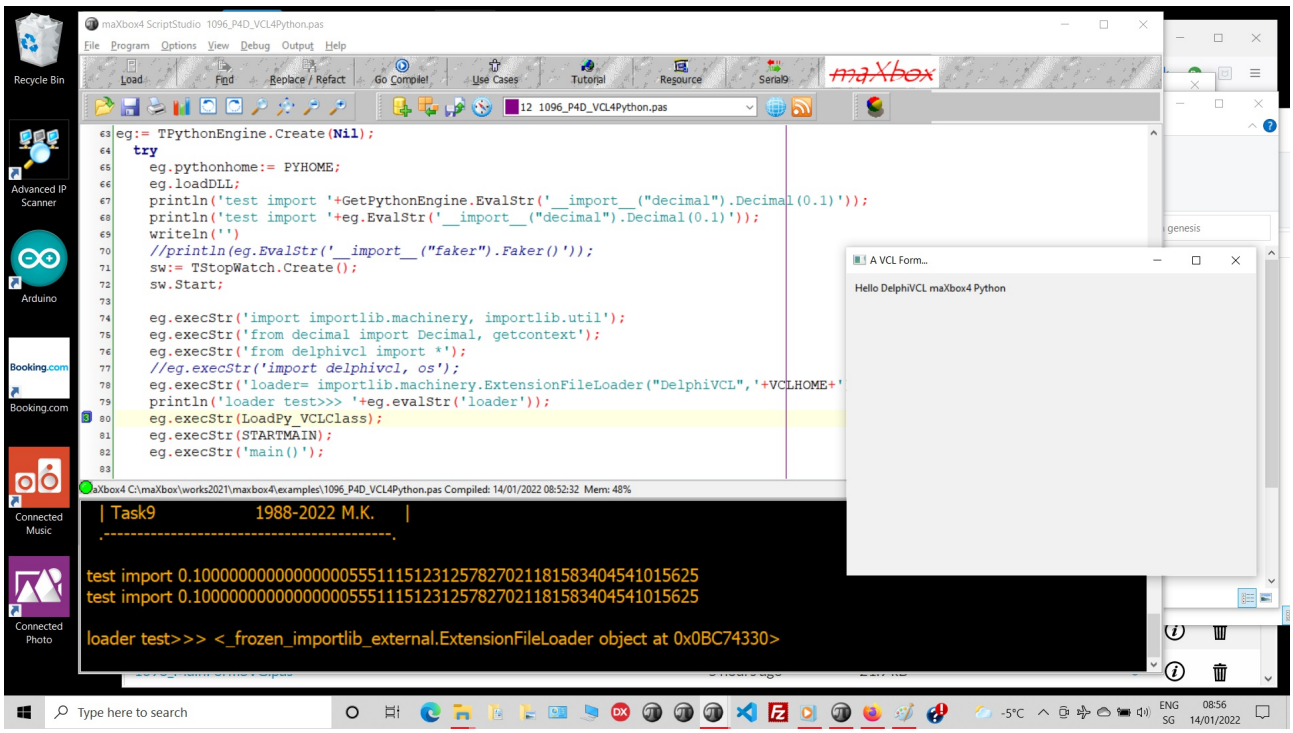
common way of invoking the import machinery, but it is not the only way. First we check our Python installation. Python 3.* provides for all user and current user installations. All user installations place the Py-dll in the Windows System directory and write registry info to HKEY_LOCAL_MACHINE.

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 or package path since it may not be on the system path as an environment var, in our case we set a const to demonstrate:

```
Const PYHOME ='C:\Users\max\AppData\Local\Programs\Python\Python36-32\';
      VCLHOME =
  'r"C:\users\max\appdata\local\programs\python\python36-32\lib\site-
packages\delphivcl\win32\delphivcl.pyd"';
```

So next we load the dll (with or without import statement possible), call the VCL class and start the main procedure:

```
  eg:= TPythonEngine.Create(Nil);
  try
     eg.pythonhome:= PYHOME;
     eg.loadDLL;
     ……..
     eg.execStr(LoadPy_VCLClass);
     eg.execStr(STARTMAIN);
     eg.execStr('main()');
```



Pic: 1096_P4D_VCL4Python.png

We can see the simple VCL-form as it says “Hello”:

https://github.com/maxkleiner/DelphiVCL4Python/tree/main/samples/HelloWorld

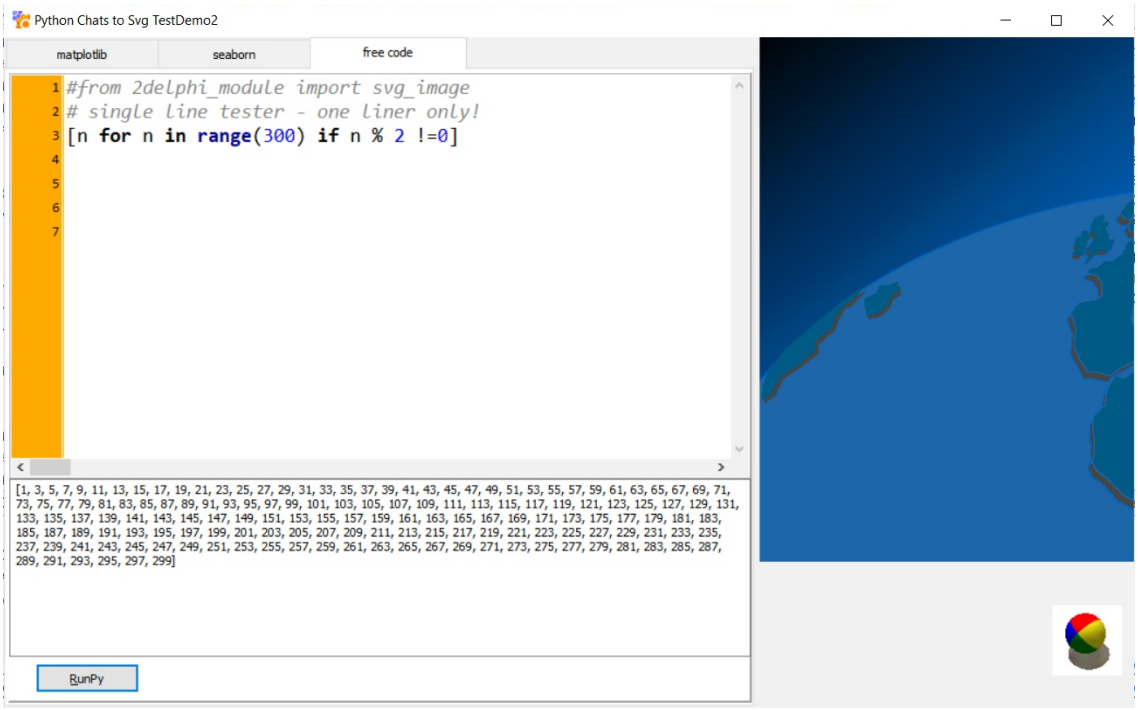
As a special proof of concept I run the hello world sample with P4D in a maXbox script to show the compatibility between the two type- and memory layout systems. But of course normally the script runs in a shell or with PyScripter. As a caveat I can run this “test toggle workaround” only once, could be that a finalizer, dispose or destructor is missing.

```
STARTMAIN =
  'def main():                                  '+LF+
  '  Application.Initialize()                   '+LF+
  '  Application.Title = "Hello Python"         '+LF+
  '  Main = MainForm(Application)               '+LF+
  '  Main.Show()                                '+LF+
  '  FreeConsole()                              '+LF+
  '  Application.Run()                          '+LF+
  '  Main.Destroy()                             ';
```

We pass with the *MainForm()* call our initialized Application to a Python class defined in LoadPy_VCLClass which has the class name 'class MainForm(Form): with two method-functions (def in class):

```
  def __init__(self, owner):
  def __on_form_close(self, sender, action):
```

Imagine on the VCL-form from Python is a SynEdit-control which enables to script in Pascal and Python together, fascinating it:



Pic: 1096_pychats_demo2.png

In P4D you do have the mentioned memo with ExecStrings:

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

This explains best the code behind, to evaluate an internal Python expression or statement. You are responsible for creating one and only one *TPythonEngine* instance.

Conclusion

The VCL/LCL is a mature Windows/Linux native GUI framework with a huge library of included visual components and a robust collection of 3rd party components and classes. It is the finest framework for native Windows applications, and we can use it with Python!

Python has only one type of module object, and all modules are of this type, regardless of whether the module is implemented in Python, Delphi, FreePascal, C, or something else.

VCL4Python topics

-
- https://learndelphi.org/python-native-windows-gui-with-delphi-vcl/
- http://www.softwareschule.ch/examples/weatherbox.txt
- http://www.softwareschule.ch/examples/pydemo37.htm
- https://github.com/maxkleiner/DelphiVCL4Python
- https://t.co/lNhgxqNr7B
-

Author: Max Kleiner, 2022