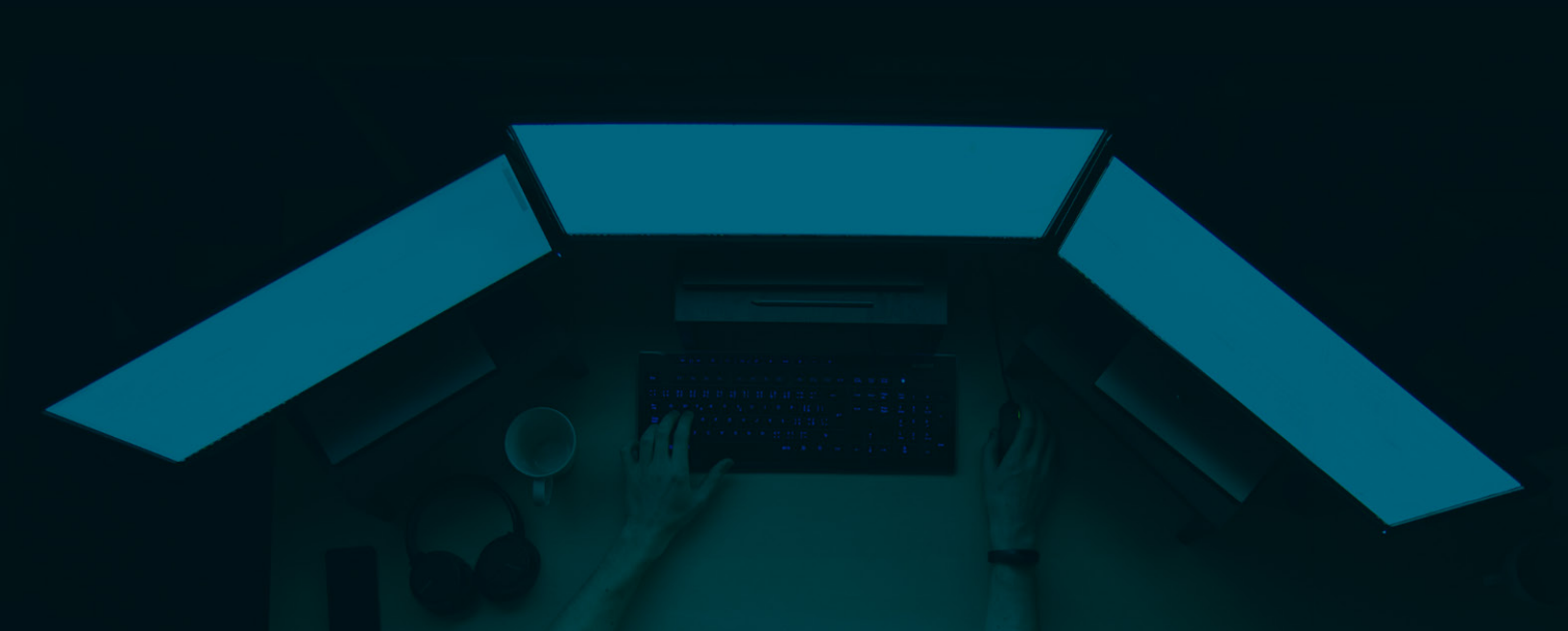


Getting Started with **DelphiVCL for Python**

Rich Windows GUI Framework for Python

embarcadero®

Table of Contents

DelphiVCL for Python ebook
Version 1.0 - March 23, 2022

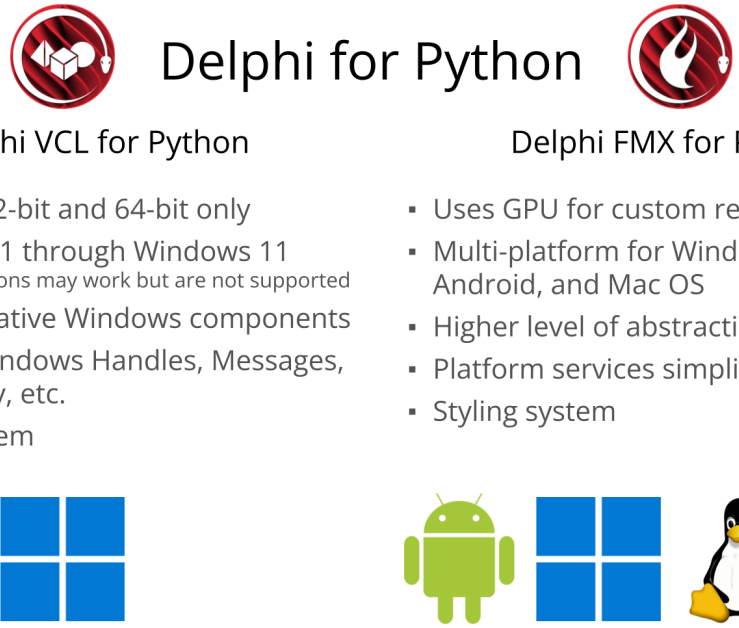
www.embarcadero.com

Introduction

Welcome to Python GUI development with Delphi for Python. In this guide, we will introduce the Delphi VCL environment. Then we're going to look at the architecture and platforms of Delphi for Python. After that, we'll look at installing and using Delphi VCL for Python, demonstrations, and code because that's always the fun part. Finally, we'll explore more about what's possible by mixing Delphi and Python.

Who should read this guide?

The primary target audience is Python domain developers who want a nice GUI, but the guide will also be useful for Delphi developers who want to extend into Python development as well.



There are quite a few Python developers who are curious about a nice GUI or about Delphi. So this guide will be helpful for both Delphi and Python developers.

Python

Growing as a senior developer is about knowing multiple programming languages on the journey. Sure, you may have one or two that are the main programming languages that you're most efficient with, but you use others as well for specific things. The reality is that we all use multiple programming languages. It's about having the right tool for the right job so you're not trying to drive a nail with a wrench or turn a bolt with a hammer.

Recently, Python has emerged as the best starting programming language. The significant characteristics of Python are that:

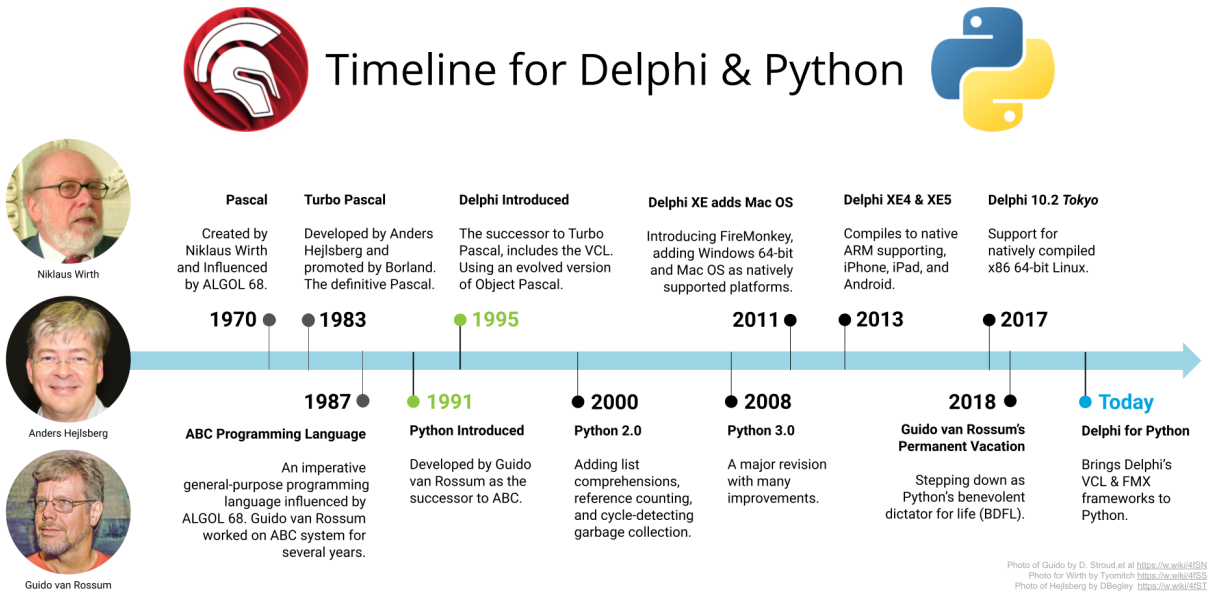
- it is significantly less on syntax
- it is intuitive so it feels like you're just reading codified English
- it automatically handles memory leaks

What Is Delphi for Python?

Delphi for Python's primary focus is to provide free Python modules or bindings of Delphi's GUI libraries to Python developers. If you are a Delphi developer and you're like, "Hey, I've been wanting to use Python, but gosh, there are no good GUI frameworks for it," well, that works too. The idea is that it gives you the ability to take the powerful, mature GUI frameworks of Delphi's VCL and FireMonkey and use them in Python. It's based on the Python for Delphi library, the same technology that powers the popular PyScripter Python IDE. It's available today on GitHub and pip, the Python package manager.

We support both legacy, prebuilt Python's virtual environments and Conda environments. We're still working on polishing it quite a bit, but it's ready to use as of today. So you can go ahead and start using it, playing with it, and filing some issues on GitHub. If you're like, "Hey, it doesn't quite do this, or this doesn't work right," let us know, and we can work on fixing that. So Python for Delphi is a bi-directional bridge between Python and Delphi. And it is kind of the first stage in giving you access to use both.

A Timeline for Delphi and Python



In 1970, Nicklaus Wirth created the Pascal program language based on or influenced by ALGOL 68. It was designed to teach good programming practices, structured programming, and more. In 1983, Anders Hejlsberg created Turbo Pascal, which was initially called Blue Label Pascal. The Borland Software Corporation acquired it, hired Hejlsberg, and promoted the Pascal language. There are several Pascal dialects emerged from the original one. Then in 1987, the ABC programming language was born.

ABC was an imperative general-purpose programming language on which Guido van Rossum worked for a while. He went on in 1991 to make Python, the successor to ABC. In 1995, Delphi, a dialect of Object Pascal, was introduced as the successor to Pascal. This first release included a graphical user interface (GUI) framework VCL along with Delphi. Quite a few languages came out in 1995. In fact, Delphi and Python are about the same age. Later, Python 2.0 and Python 3.0 were released, which moved the Python language forward. And then, in 2011, Delphi introduced Mac support with FireMonkey GUI framework for cross-platform support. And in 2013, we gave support to the ARM compiler with Delphi, adding iOS and Android support.

In 2017, Embarcadero released Delphi Tokyo, which brought Linux support.Then, 2018 was when Guido van Rossum stepped down and took a permanent vacation as Python’s benevolent dictator for life. And that brings us to today, where we're going to talk about Delphi for Python.

Delphi's DNA Compared to Zen of Python

One of the essential considerations of Delphi is developer productivity. To help them achieve productivity, it provides:

- A WYSIWYG visual designer to drag and drop components
- A wide range of both visual and nonvisual components and the properties and events associated with the components
- Backward compatibility to older versions of Delphi
- A rich component ecosystem

Developers can quickly make fast, powerful applications using the above.

We want to contrast Delphi's DNA with the Zen of Python, which has 19 guiding principles that influence the design of the Python programming language. Let's check out a few of those:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

So in a lot of ways, Delphi and Python share the same philosophies.

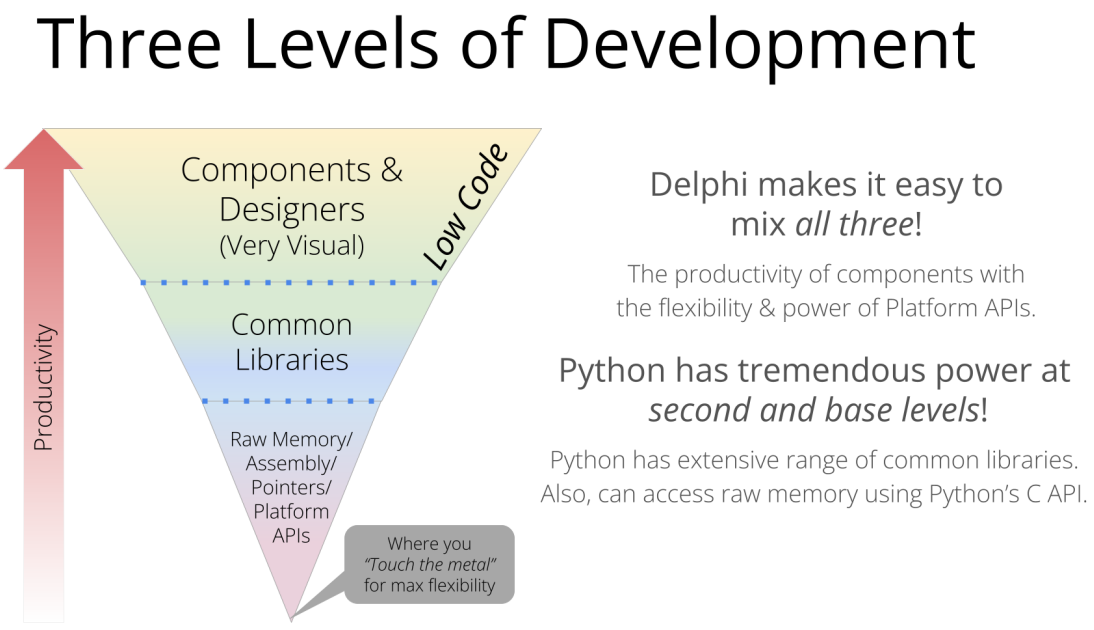
What are the differences between Delphi and Python?

Some key differences are these:

- Although commercial licenses are available, Delphi has a free community edition, whereas Python is open source.
- Python has garbage collection with reference counting and cycle detection. Delphi has manual memory management, but it has an ownership model that simplifies it for most scenarios.
- Delphi is a compiled language, although there are some third-party interpretation options. In contrast, Python is primarily interpreted with some options to compile it.
- People build for business purposes, business-to-business type software, or internal IT development using Delphi. Although Python has grown as a general-purpose language, it's prevalent in research and prototyping machine learning.

Three Levels of Development

Into which levels Do Delphi and Python Fall?



Development, using Delphi or in general, falls into one of these three categories:

1. Low code, with visual components and designers
2. Standard and third-party libraries to implement specific functionalities
3. Raw memory access using pointers or assembly code

Delphi has many low-code tools designed for people who don't want to learn programming and just want to get something done. Those tools are very visual, so very little code is written and

the focus is on productivity. There are limitations on what you can do at this low-code level. At the second level, we use common libraries and write programs. Here we can customize more specific functionality compared to level one. Then you have the lowest level, which is the most flexible and most powerful, but the least productive. Here, you're dealing with raw memory and pointers by writing assembly language code. You're talking directly to the CPU with archaic platforms and APIs.

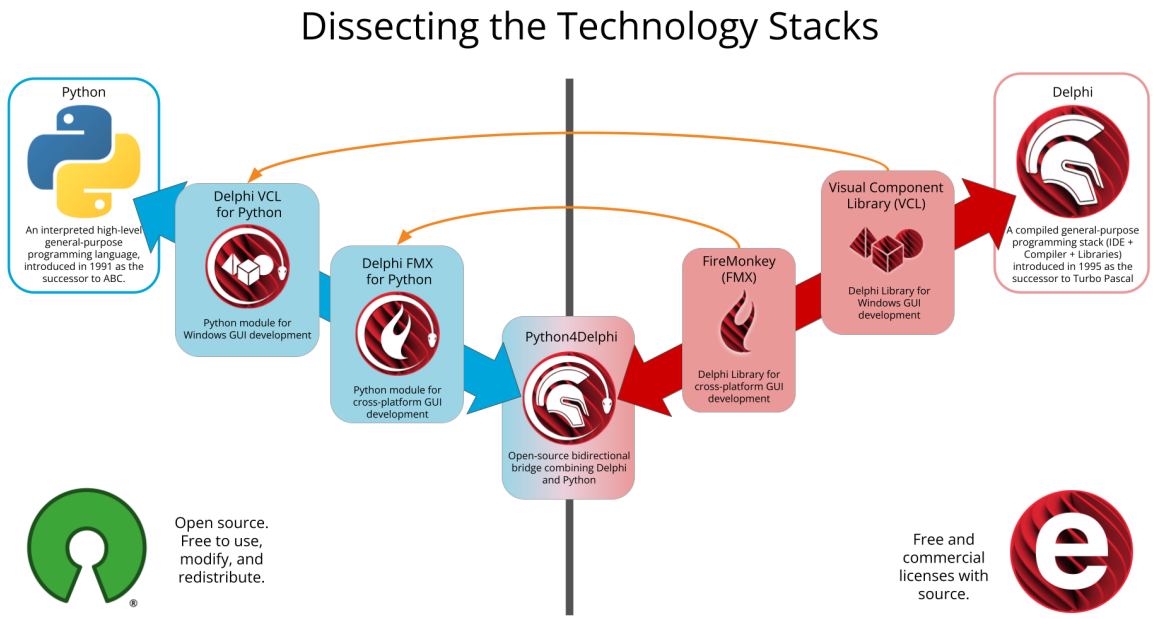
Delphi does an excellent job of combining all three levels of development. Python mainly falls into the middle level, with vast built-in and third-party libraries. Python has automatic garbage collectors, whereas Delphi deals with garbage manually. We can extend Python with the C language. Many third-party Python modules are written in C or C++, which gives you raw memory management and extends Python's reach to low level platform access.

Python's provision to create extension modules in C and offer Python bindings has made it the most used language. Furthermore, we can import those extension modules as libraries, making Python incredibly productive. This is our primary focus area where we also created the extension modules for Python that are written in Delphi.

Delphi VCL and Delphi FireMonkey Libraries to Python

Delphi offers two libraries or frameworks for graphical user interface (GUI) application development:

- Visual Component Library (VCL)
- FireMonkey (FMX)



VCL was initially released along with the first Delphi back in 1995, focused on Windows. Even though Delphi is not open source, it shipped with source code, making a huge difference in its adoption. Delphi is built with the component model for component reuse. It gave all the developers access to the source code to see how Delphi did that and create their custom components. So it created a rich ecosystem of components for developers to use. VCL is based on Windows components, and all of them have handles to intercept messages. A significant part of the VCL framework wraps standard Windows components. So you have automatic access to everything that comes with a Windows component, but it adds a layer of abstraction. VCL simplifies their usage, and you don't have to think about handles, messages, or other things.

FireMonkey framework was released in 2011, as you can see in the timeline above. FireMonkey is designed from the ground up to be a cross-platform GUI framework. It takes advantage of GPUs. So hardware acceleration uses DirectX on Windows and OpenGL on other platforms to create very fast, great-looking UIs across platforms. It supports Windows, Mac, iOS, Android, and Linux. If you know VCL, you can shift to FireMonkey quickly, but FireMonkey is not hindered by trying to be completely backward compatible with the VCL. It includes platform services, which help with the abstraction moving across platforms. So the behaviors and the look and feel automatically adapt to the platform you're running on.

With the recent advancements, Delphi is as fast as C++. We created Python extension modules for VCL and FireMonkey using the Python for Delphi bridge. So these two libraries make up Delphi for Python, essentially creating:

- Delphi VCL for Python
- Delphi FMX for Python

Getting Started with DelphiVCL for Python

Delphi VCL for Python is a Python binding of Delphi's legacy GUI framework—VCL. As mentioned, it's a GUI framework for the Windows operating system platform. You don't need to install or know Delphi to develop GUI applications using Delphi VCL for Python. It's Windows-focused and lightweight because it uses pre-existing Windows controls. The only requirement is installing the **`delphivcl`** package in Python. In contrast, Delphi FMX for Python is a cross-platform GUI application development library. If you want to develop GUI applications for Linux, Mac, Android, or iOS using Python, you need the **`delphifmx`** package. Applications built using DelphiVCL will be faster than those built on DelphiFMX because DelphiVCL speaks to Windows directly. So you have the advantage of using DelphiVCL when developing GUI applications for Windows.

As you learn and start developing with DelphiVCL, you can shift into DelphiFMX in no time to create cross-platform applications, as they are very similar.

Installation

DelphiVCL for Python is a native Python module. It is available via PyPi or by downloading the source via GitHub. It is natively compiled for Win32 and Win64 and should work on Microsoft Windows 8 or newer. While it does not include a Windows ARM binary, it should work via the x86 ARM interop included in Windows for ARM. All Python versions from 3.6 to 3.10 are supported, including Conda.

The easiest way to install is via PIP:

```
pip install delphivcl
```

You can also install manually by downloading or cloning the repository from GitHub: github.com/Embarcadero/DelphiVCL4Python. After cloning or downloading, enter the root **DelphiVCL4Python** folder/directory and open the command prompt or Anaconda prompt with that path. Now install the package using:

```
python setup.py install
```

Testing the Installation

After installing **`delphivcl`** library using **`pip`**, let's enter the Python REPL to understand a few essential things. Python has a predefined **`dir()`** function that lists available names in the local scope. So, before importing anything, let's check the available names using the **`dir()`** function.

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__']
```

Now let's import the installed **`delphivcl`** module to validate its installation and check for the output of the **`dir()`** function:

```
>>> import delphivcl
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__', 'delphivcl']
```

In the above output list, we have **`delphivcl`** as part of the available names in the local scope. In this case, if we need to use any classes or functions available in the **`delphivcl`** module, we should use a dot (**`.`**) operator after it. Now let's import everything using **`*`** from the **`delphivcl`** library and check for the available classes, functions, objects, constants, and so forth.

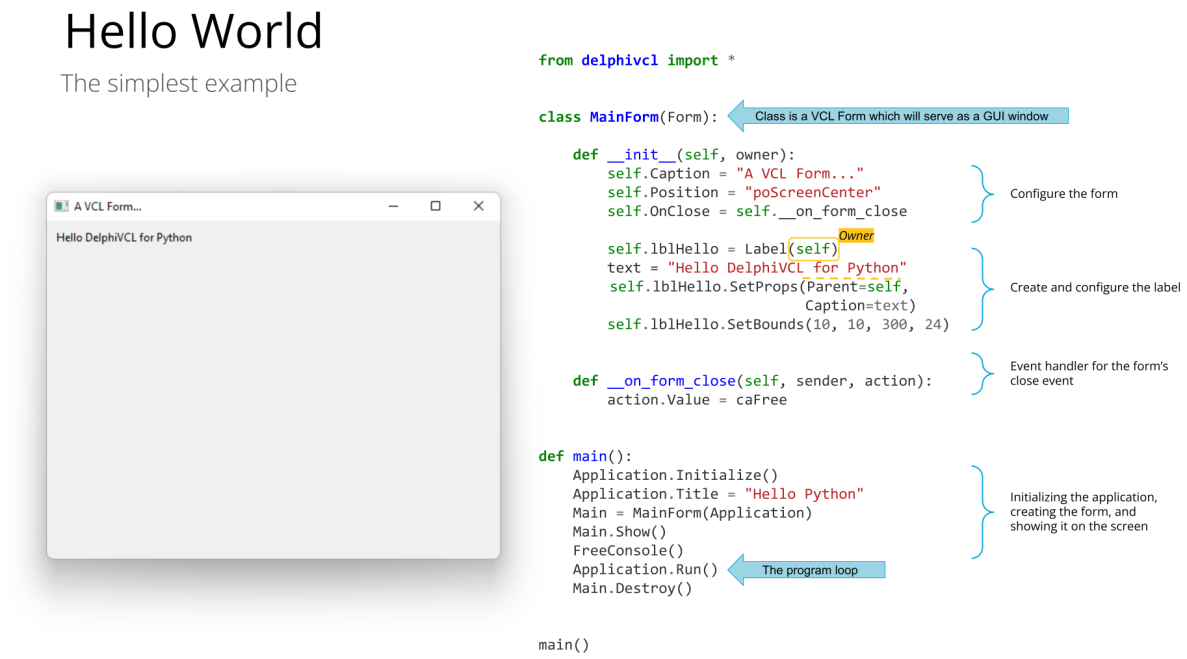
```
>>> from delphivcl import *
>>> dir()[0:45]
['Abort', 'Action', 'ActionList', 'ActivityIndicator', 'Application',
'BasicAction', 'Bevel', 'BitBtn', 'Bitmap', 'BoundLabel', 'Button',
'Canvas', 'CheckBox', 'Collection', 'ColorBox', 'ComboBox', 'Component',
'ContainedAction', 'ContainedActionList', 'Control', 'ControlBar',
'CreateComponent', 'CustomAction', 'CustomActionList',
'CustomActivityIndicator', 'CustomControl', 'CustomDrawGrid', 'CustomEdit',
'CustomForm', 'CustomGrid', 'CustomMemo', 'CustomStyleServices',
'CustomTabControl', 'CustomToggleSwitch', 'DateTimePicker',
'DelphiDefaultContainer', 'DelphiDefaultIterator', 'DelphiMethod',
'DrawGrid', 'Edit', 'FileOpenDialog', 'Form', 'FreeConsole', 'Graphic',
'GroupBox']
```

To avoid an enormous list of names, we checked for the first 46 elements only using **`dir()[0:45]`**. Let's check for a few available classes, functions, and objects.

```
>>> CreateComponent
<built-in function CreateComponent>
>>> Button
<class 'Button'>
>>> Form
<class 'Form'>
>>> Application
<Delphi object of type TApplication at 2033DFE9C30>
```

We need to create an object instance for other classes like **`Button`** and **`Form`**. There are many other classes and functions but only one object **`Application`** instance, which is an existing singleton instance, ready for use with the dot (**`.`**) operator. It is the source of all GUI applications that we create.

Hello Delphi VCL for Python



The code on the right-hand side of the above image for you to copy and run is:

```
from delphivcl import *

class MainForm(Form):

    def __init__(self, Owner):
        self.Caption = "A VCL Form…"
        self.Position = "poScreenCenter"
        self.OnClose = self.__on_form_close

        self.lblHello = Label(self)
        text = "Hello DelphiVCL for Python"
        self.lblHello.SetProps(Parent=self,
                               Caption=text)
        self.lblHello.SetBounds(10, 10, 300, 24)

    def __on_form_close(self, sender, action):
        action.Value = caFree

def main():
    Application.Initialize()
    Application.Title = "Hello Python"
```

```
    Main = MainForm(Application)
    Main.Show()
    FreeConsole()
    Application.Run()
    Main.Destroy()

main()
```

A breakdown of the concepts in the above code is:

```
from delphivcl import *
```

At first, we imported everything from **delphivcl**. Later, we will create the GUI application window using the above code in the main function.

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

main()
```

Then, we initialized the application and set a title for it. We can refer to all the classes that are part of the import from the **delphivcl** library as components. The **Form** is special and different as it creates the GUI window containing all other components. We instantiated the **Form** with **Application** as the owner parameter, using **Main = MainForm(Application)**. The **MainForm** class is used to create all the GUI components and the event-handling functions.

```
class MainForm(Form):

    def __init__(self, Owner):
        self.Caption = "A VCL Form…"
        self.Position = "poScreenCenter"
        self.OnClose = self.__on_form_close

        self.lblHello = Label(self)
```

```
        text = "Hello DelphiVCL for Python"
        self.lblHello.SetProps(Parent=self,
                               Caption=text)
        self.lblHello.SetBounds(10, 10, 300, 24)

    def __on_form_close(self, sender, action):
        action.Value = caFree
```

Here we've set a **Caption** that appears on the title bar of the Form window. Please check the left side of the above image, where you can observe the title of the GUI window. The line **self.Position = "poScreenCenter"** sets our GUI window's position at the center of the screen. All the component classes, including **Form**, have many event-handling methods to handle various possible events. Here, we used the **OnClose** event of **Form**. The events are assigned with the methods to perform the required and necessary actions. For example, we used the **__on_form_close()** method to handle the event or action of closing the GUI window where we're freeing up or releasing the **Main** object of **MainForm**.

The **Label** class is used to display text on the GUI window. All the components, including **Form**, have a

- **SetProps()** method to set their properties, and a
- **SetBounds()** method to set the position and size of the component.

Here, we're setting the **Parent** and **Caption** properties. The **Parent** of the **lblHello** object is the **MainForm** instance, and we assigned the label's text. The first two parameters of the **SetBounds()** method set the origin position of the component relative to the **Form**, whereas the last two set its size (width and height).

As we create the application and set its properties, we will show it on the screen using the **Main.show()** code snippet. GUI applications run in interaction with the command window (console). To make the GUI perform better without lags, we use **FreeConsole()** to give primary control to the GUI interface. **Application.Run()** starts the GUI interaction loop between the GUI and the user of the GUI application. When we close the GUI application, **Main.Destroy()** takes care of not crashing it.

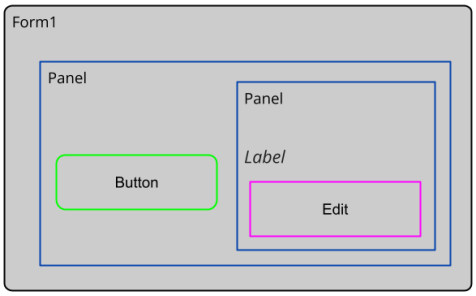
Delphi has an ownership model, where all components can have an owner. Every component has an owner and a parent. In the above sample script, **Application** is the owner of the form, and this works differently with the **Parent**.

Owner
(Lifecycle)

- Application
  - Form1
    - Panel
    - Label
    - Edit
    - Button
    - Panel
    - Components
  - Form2
    - Components

Parent
(Drawing)

- Form1
  - Panel
    - Button
    - Panel
      - Label
      - Edit
- Form2
  - Nested Components
    - Nested Components
      - Nested Components



All visual components (except the **`Form`**) must have a **`Parent`** defined. The parent is used when drawing the components. A component is drawn on its parent. Both visual and non-visual components have an **`Owner`** property, but the value is optional. The owner is used for object lifecycle and memory management. The application owns the forms, and then the forms own the components on it.

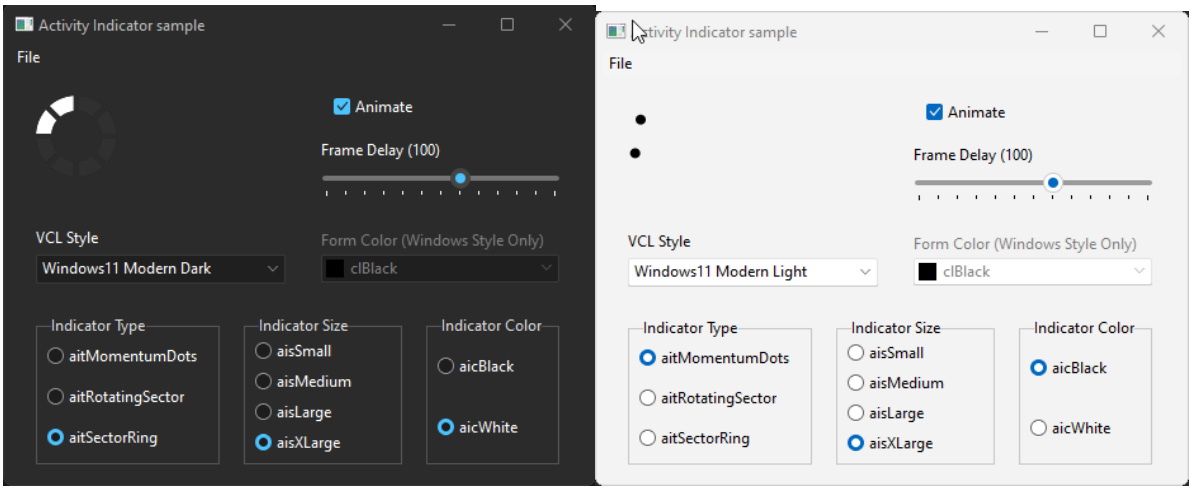
In our example, we have a single **`Form`**, but using the DelphiVCL library, you can create multiple forms too. If you have multiple forms, each form owns all of its components, and then those forms are owned by the **`Application`** object. The parent, on the other hand, is associated with the visual scope of the components. A **`Panel`** is a composite component in that it can act as a parent for other components inside it.

The parent can also be the owner, but that isn't required. From the above image, we can understand the illustrated concepts of Application Life Cycle and Owner-Parent-Child interaction.

Introducing VCL Styles

Think of VCL styles as themes in your mobile phone. They change the whole look and feel of the GUI application. We're bringing the capability to add VCL styles to Python with **`delphivcl`**. Creating a GUI application with Delphi VCL for Python uses the default "**Windows**" VCL style. Please check the below images, where an activity indicator application takes advantage of two VCL styles:

- Windows11 Modern Dark
- Windows11 Modern Light



Summary

The Python bindings for Delphi GUI libraries are essential for the Python GUI developer community. We're bringing the decades of research and development of the mature Delphi VCL GUI library to Python. Some of the professional-grade GUI applications built using Delphi are:

- FL Studio/Fruity Loops Digital Audio Workstation
- KMPlayer Media player
- PyScripter

For more case studies, please check Embarcadero success stories.

Quick Start Guide

We developed a dedicated quick-start guide covering DelphiVCL for Python that includes more details and examples, along with a bundle of VCL Styles. It is available as a free download.

lp.embarcadero.com/QuickStartDelphiVCLPython

About Embarcadero Technologies

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster, and run them better regardless of platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance, and accelerate innovation. Founded in 1993, Embarcadero is headquartered in Austin, Texas, with offices located around the world.

10801 North Mopac Expressway, Building 1, Suite 100
Austin, TX, 78759
www.embarcadero.com/company/contact-us
US: 1 (512) 226-8080 - info@embarcadero.com

About PyScripter

PyScripter is an open-source and feature-rich lightweight Python IDE.

PyScripter has all the features expected in a modern Python IDE in a lightweight package. It's also natively compiled for Windows to use minimal memory with maximum performance. The IDE is open-source and fully developed in Delphi with extensibility via Python scripts.
www.embarcadero.com/free-tools/pyscripter/free-download

About Delphi

Delphi is Embarcadero's flagship development tool supporting native application and server development for Windows, macOS, Linux, Android, and iOS. It includes a variety of libraries, including a robust framework for database applications, REST services, and visual application development. It is available in multiple editions, including a free Community Edition, an Academic Edition, and Professional, Enterprise, and Architect Editions.
www.embarcadero.com/products/delphi

Download a free trial today!