DelphiVCL for Python

Quick Start Guide

embarcadero®

Introduction to DelphiVCL for Python

DelphiVCL for Python is a Python GUI library built on the Delphi VCL framework.

Delphi is a general-purpose programming language that emerged as a dialect of the Object Pascal programming language. Unlike Python, Delphi is a compiled language that is very fast. In addition, Delphi has a mature and robust visual component library (VCL) framework to support native Windows GUI applications development. The VCL framework covers all the native Windows controls and includes many other rich visual components.

Many powerful and feature-rich Windows GUI applications are built using the VCL framework. DelphiVCL for Python brings the power of Delphi's VCL to Python. Most of the high-performance scientific computing libraries like NumPy and Matplotlib are C/C++ bindings for Python. Similarly, we're wrapping Delphi's VCL framework as a Python module to achieve a great combination of high-performance and extensive features of VCL.

If you are interested in cross-platform GUI development, then be sure to check out Delphi FMX for Python with support for Windows, MacOS, Linux, and Android.

Table of Contents

DelphiVCL for Python Quick Start Guide
Version 1.0 - March 3rd, 2022

www.embarcadero.com
www.ideracorp.com

Installation

DelphiVCL for Python is a native Python module. It is available via PyPi or download 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
```

DelphiVCL for Python is freely redistributable via its freeware license agreement. To redistribute the Python application created using DelphiVCL for Python, you can either

- recommend that your users run **`pip`** to install it on the target system, or
- if you know the system configuration of the target system and want more control, use any one of the following deploy everything necessary in one package:
    - InnoSetup is one of the most popular and powerful Windows installation systems.
    - pyInstaller bundles a Python script and all dependencies into a single package.
    - cx_Freeze creates standalone executables from Python scripts.
    - py2exe converts Python scripts into executable Windows programs.
    - pyship ships Python desktop apps to end users.
    - Pynsist builds Windows installers for your Python applications.

We recommend using pip as it uses system configuration (architecture and Python version) to install the package that supports the current system configuration.

Quick Start Guide 1: Testing the Install

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.

Let's now create a simple GUI application. The code for it is:

```
from delphivcl import *

Application.Initialize()
Application.Title = "Hello Delphi VCL"
app = Form(Application)
app.SetProps(Caption = "Welcome")
app.Show()
FreeConsole()
Application.Run()
app.Destroy()
```

Using the above code, we just created an empty GUI app. Please save the above code and run it to see the following output:

Let's explore and understand the functionality of the code.

```
from delphivcl import *

Application.Initialize()
Application.Title = "Hello Delphi VCL"
```

At first, we imported everything from **`delphivcl`**. Then, we initialized the application and set a title for it. Later, we will create the GUI application window using the following code:

```
app = Form(Application)
app.SetProps(Caption = "Welcome")
```

We can refer to all the classes that are part of the import 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 in the above code. All the components, including **`Form`**, have a method **`setProps()`** to set their properties. Here we've set **`Caption`** that appears on the title bar of the Form window.

Let's look at the following few lines of the code:

```
app.Show()
FreeConsole()
Application.Run()
app.Destroy()
```

As we created the application and set its properties, we will show it on the screen using the **`app.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, **`app.Destroy()`** takes care of not crashing it.

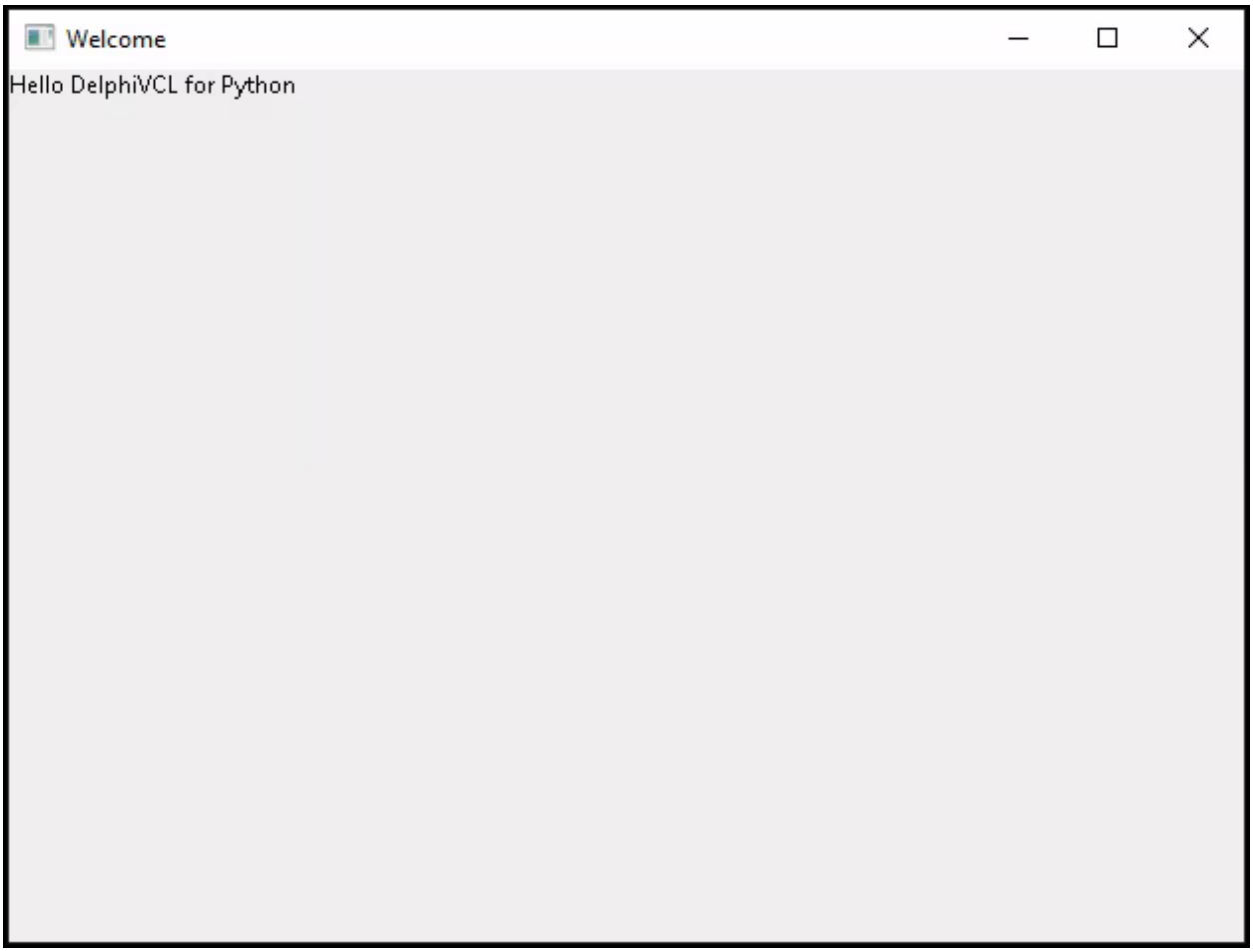
Quick Start Guide 2: Hello, World

We discussed the most basic ideas about the **`delphivcl`** library in the first, simplest quick start. We created an empty GUI application without displaying anything on the Form/GUI window. Also, we did not use an object-oriented approach to create the GUI application. So let's expand on those ideas, develop an object-oriented version displaying a text message.

First, let's look at the code. You might be able to guess what the below code does if you understood the basics from the first guide.

```
from delphivcl import *

class GUIApp(Form):

    def __init__(self, owner):
        self.SetProps(Caption = "Welcome")

        self.lblHello = Label(self)
        self.lblHello.SetProps(
            Parent=self,
            Caption="Hello DelphiVCL for Python")

def main():
    Application.Initialize()
    Application.Title = "Hello Delphi VCL"
    app = GUIApp(Application)
    app.Show()
    FreeConsole()
    Application.Run()
    app.Destroy()

main()
```

As you save the above code in a Python file and run it, you'll get the following GUI window with a text message as follows:



In the main function, let's look at the following line of code:

```
    app = GUIApp(Application)
```

Instead of instantiating the **Form** directly, we instantiate a class - **GUIApp** that extends the **Form** class. Let's investigate the code in the **GUIApp** class:

```
class GUIApp(Form):

    def __init__(self, owner):
        self.SetProps(Caption = "Welcome")

        self.lblHello = Label(self)
        self.lblHello.SetProps(
            Parent=self,
            Caption="Hello DelphiVCL for Python")
```

As we instantiated the **GUIApp** using **app = GUIApp(Application)**, the **owner** argument was assigned with the **Application** object. After that, **Form** uses the **owner** in its initialization and creates an empty Form/GUI window. This **owner** variable can have any other name as it's just a placeholder of the **Application** object. In the first line of the **GUIApp** initialization, we've set the **Caption** property of the **Form**.

Then we instantiated the **Label** component/class with the instance/object of the **Form** as its parameter using the **self.lblHello = Label(self)** code snippet. We use **Label** to display any single-line text messages. Every component other than **Form** will have a parent and is set using the **Parent** property. The parent holds the child component in it.

In our code, we're setting **Label**'s parent as **Form** using **Parent=self**. So now the **Form** object - **app** holds the **Label** object - **lblHello**. Next, the text of the **Label** is set using its **Caption** property. So the Form/GUI window gets populated by a text message: - **Hello DelphiVCL for Python**.

We used all the default positions and sizes of the **Form** and **Label** and didn't handle any events in this guide. However, we will implement them and introduce some new components in the following advanced quick start guide.

Quick Start Guide 3: TODO Task App

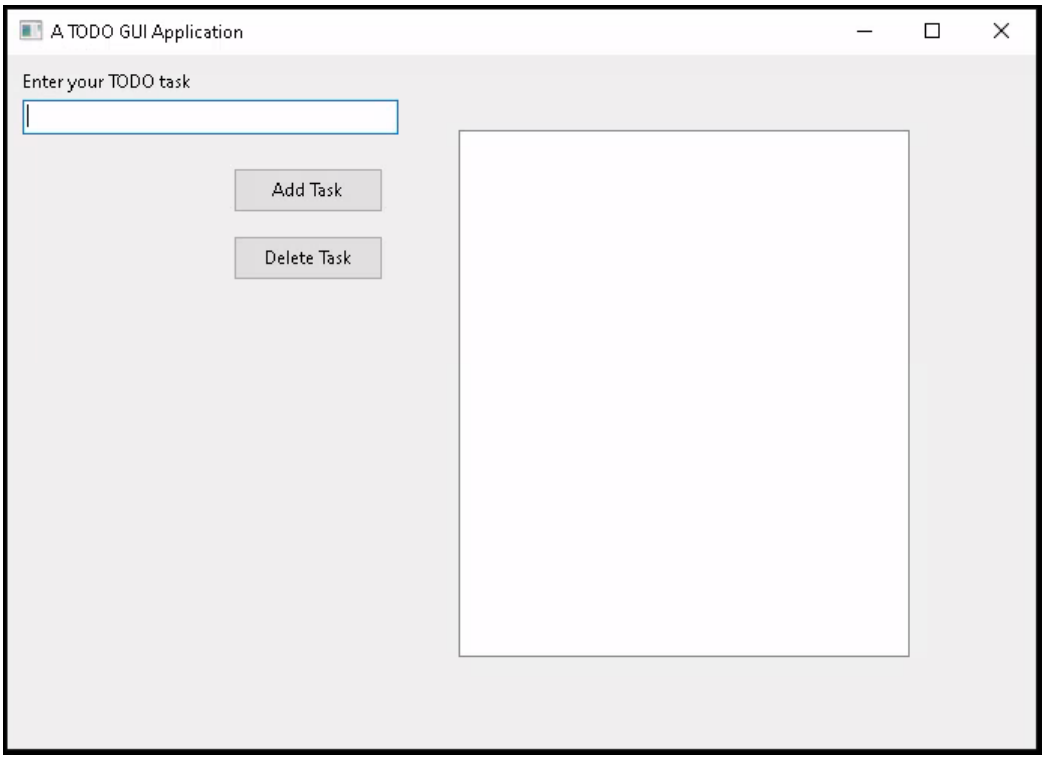
Here we will create a TODO task application to understand some components of GUI applications.

Let's take a look at the code to achieve that:

```
from delphivcl import *

class TodoApp(Form):

    def __init__(self, Owner):
        self.Caption = "A TODO GUI Application"
        self.SetBounds(100, 100, 700, 500)

        self.task_lbl = Label(self)
        self.task_lbl.SetProps(Parent=self, Caption="Enter your TODO task")
        self.task_lbl.SetBounds(10, 10, 125, 25)

        self.task_text_box = Edit(self)
        self.task_text_box.SetProps(Parent=self)
        self.task_text_box.SetBounds(10, 30, 250, 20)

        self.add_task_btn = Button(self)
        self.add_task_btn.Parent = self
        self.add_task_btn.SetBounds(150, 75, 100, 30)
        self.add_task_btn.Caption = "Add Task"
        self.add_task_btn.OnClick = self.__add_task_on_click

        self.del_task_btn = Button(self)
        self.del_task_btn.SetProps(Parent = self, Caption = "Delete Task")
        self.del_task_btn.SetBounds(150, 120, 100, 30)
        self.del_task_btn.OnClick = self.__del_task_on_click

        self.list_of_tasks = ListBox(self)
        self.list_of_tasks.Parent = self
        self.list_of_tasks.SetBounds(300, 50, 300, 350)

        self.OnClose = self.__on_form_close

    def __on_form_close(self, Sender, Action):
        Action.Value = caFree
```

```
    def __add_task_on_click(self, Sender):
        self.list_of_tasks.Items.Add(self.task_text_box.Text)
        self.task_text_box.Text = ""

    def __del_task_on_click(self, Sender):
        self.list_of_tasks.Items.Delete(0)

def main():
    Application.Initialize()
    Application.Title = "TODO App"
    app = TodoApp(Application)
    app.Show()
    FreeConsole()
    Application.Run()
    app.Destroy()

main()
```

As you save and run the above code, you should get the following GUI as a result:



Let's get to the details of what our code does behind the scenes. First, take a look at the **`main()`** function:

```
def main():
    Application.Initialize()
    Application.Title = "TODO App"
    app = TodoApp(Application)
    app.Show()
    FreeConsole()
    Application.Run()
    app.Destroy()
```

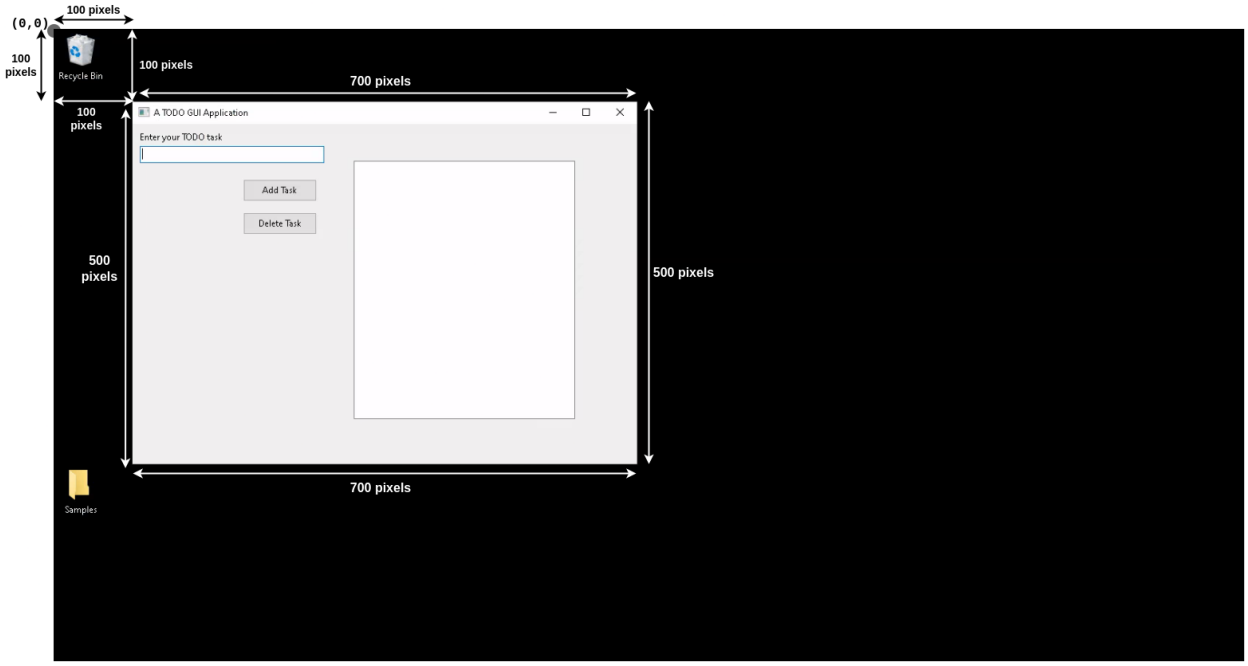
Above, we instantiated the **`TodoApp`** class with **`Application`** as the **`Owner`**. As we instantiate the GUI using **`app = TodoApp(Application)`**, the following code runs:

```
class TodoApp(Form):

    def __init__(self, Owner):
        self.Caption = "A TODO GUI Application"
        self.SetBounds(100, 100, 700, 500)
```

We inherit the **`Form`** class from the **`delphivcl`** library to create our GUI. In Delphi VCL, all the GUIs are treated as forms. The name of the GUI pop-up window is set using the **`Caption`** property/attribute. The line **`self.SetBounds(100, 100, 700, 500)`** is used to set:

- the GUI window's origin position comparable to screen's origin position = **`(100, 100);`**
- width of the GUI window = **`700`** pixels; and
- height of the GUI window = **`500`** pixels.

The upper-left corner of the screen is treated as the **`(0, 0)`** coordinate, with the left side as positive width and down as positive height. We can visualize it as shown below:

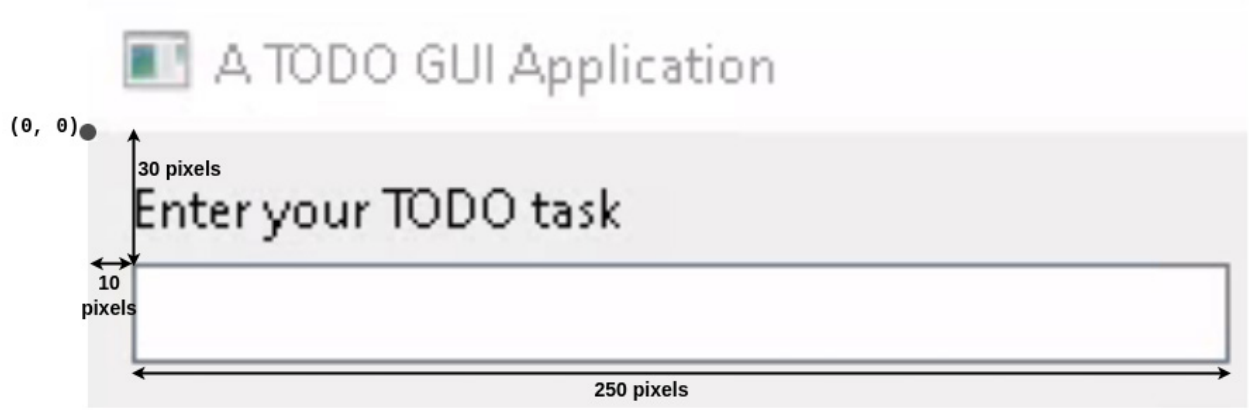


Let's look at the following few lines of code:

```
        self.task_lbl = Label(self)
        self.task_lbl.SetProps(Parent=self, Caption="Enter your TODO task")
        self.task_lbl.SetBounds(10, 10, 125, 25)

        self.task_text_box = Edit(self)
        self.task_text_box.Parent = self
        self.task_text_box.SetBounds(10, 30, 250, 20)
```

The first three lines of code above will create the text “- Enter your TODO task” that you see on the GUI app. It does so by instantiating the **`Label`** class of the **`delphivcl`** library. Every component (**`Label`** here) has a **`SetProps()`** method to set its properties. Every component will have a scope that is set using its **`Parent`** property/attribute, which is set to **`self`** here. The **`Caption`** property sets the string of the text label. Similar to the Form/GUI app, every component needs to be placed inside the Form/GUI using the **`SetBounds()`** method. For components, the top-left corner of their parent (GUI window here) is considered as the origin - **`(0, 0)`**.

The next 3 lines of code create the edit box using the **`Edit`** class. We can also set the properties/attributes directly without using the **`SetProps()`** method, like we did here using the code **`self.task_text_box.Parent = self`**. With the Form/GUI window as the parent of the Edit box, we can visualize its position and size as shown in the below figure. The height of the Edit box is automatically set to the default value.



Let's look at the next few lines of code:

```
        self.add_task_btn = Button(self)
        self.add_task_btn.Parent = self
        self.add_task_btn.SetBounds(150, 75,100,30)
        self.add_task_btn.Caption = "Add Task"
        self.add_task_btn.OnClick = self.__add_task_on_click

        self.del_task_btn = Button(self)
        self.del_task_btn.SetProps(Parent = self, Caption = "Delete Task")
        self.del_task_btn.SetBounds(150,120,100,30)
        self.del_task_btn.OnClick = self.__del_task_on_click
```

The above lines of code create two buttons, **`Add Task`** and **`Delete Task`**, using the **`Button`** instance of the **`delphivcl`** package. For the buttons, one extra thing you'll find is an event handling using **`self.add_task_btn.OnClick = self.__add_task_on_click`** and **`self.del_task_btn.OnClick = self.__del_task_on_click`** for **`Add Task`** and **`Delete Task`** buttons, respectively. We will look at this in just a while.

Let's look at the next few lines of code:

```
        self.list_of_tasks = ListBox(self)
        self.list_of_tasks.Parent = self
        self.list_of_tasks.SetBounds(300,50,300,350)

        self.OnClose = self.__on_form_close
```

In the above lines of code, we created a list box using the **ListBox** instance. At the last line of code, **self.OnClose = self.__on_form_close**, we're handling the event of closing the GUI window using the **__on_form_close()** method, which is mentioned below:

```
    def __on_form_close(self, Sender, Action):
        Action.Value = caFree
```

When we trigger the **OnClose** event, the Form automatically sends two arguments, - **Sender** and **Action**, to the event-handling method. These **Sender** and **Action** can have different names. The line **Action.Value = caFree** releases the created instance (**app** here) using the **caFree** constant.

Let's now look at the event-handling methods for the **Add Task** and **Delete Task** buttons:

```
    def __add_task_on_click(self, Sender):
        self.list_of_tasks.Items.Add(self.task_text_box.Text)
        self.task_text_box.Text = ""

     def __del_task_on_click(self, Sender):
        self.list_of_tasks.Items.Delete(0)
```

For all the events other than **OnClick**, the Form automatically sends a single argument (**Sender** here, but this can be any name). We can add a task to the list box by typing anything into the text box and pressing the **Add Task** button. Delphi VCL library-based GUIs support tab controls too, where you can also navigate from one component to another using the tab key. So you can press `Tab` on the keyboard, and when the **Add Task** button gets highlighted, you can press `Enter/Return` to fire its event. We add text from the text box to the list box using the **Add()** method under **Items** under **ListBox** instance. We delete the earlier added events on a first-come, first-serve basis by pressing the **Delete Task** button.

Adding Style

Think 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's with **`delphivcl`**. Creating a GUI application with Delphi VCL for Python uses the default "**Windows**" VCL style. We can load any other style using the **`StyleManager`** class. The VCL Style files end with a **`.vsf`** extension. By just adding five more lines of code, we give our TODO application a stunning look.

Note: You need a VCL style file **`<style-file>.vsf`** to follow the below demonstration. The Quick Start Guide bundle includes some styles. Additional styles are available with the Delphi IDE, and you can find other styles online, such as at delphistyles.com.

Now let's add the below two lines at the starting of the TODO app's Python code:

```
import os

FILE_DIR = os.path.dirname(os.path.abspath(__file__))
```

We import the **`os`** module to get the current Python file folder path and to load the VCL style file into the GUI application. The global variable **`FILE_DIR`** stores the path of the directory of the TODO app's Python file.

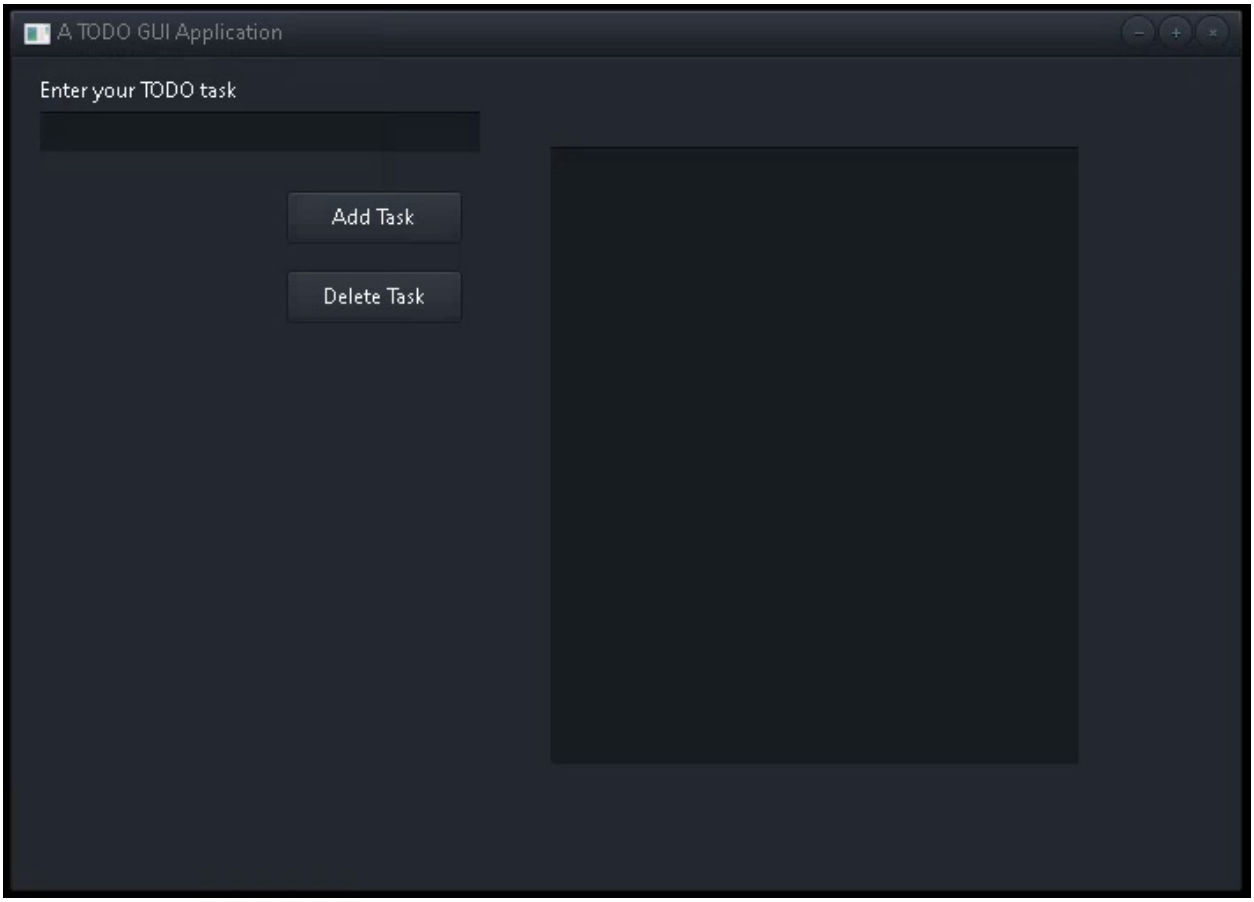
Then add the following three lines of code anywhere inside the **`__init__()`** method's scope of the **`TodoApp`** class.

```
        self.__sm = StyleManager()
        self.__sm.LoadFromFile(os.path.join(FILE_DIR, "Glow.vsf"))
        self.__sm.SetStyle(self.__sm.StyleNames[1])
```

In the above code, we're instantiating **`StyleManager`** in the first line. The second line loads a **`Glow.vsf`** style file that exists in the TODO app's Python file's directory. As mentioned, you need a VCL style file to implement the second line in the above code. Finally, the third line of the above code sets the loaded style for the GUI application.

The **`self.__sm.StyleNames`** stores the list of styles that are available for the GUI application. We already have a default **`Windows`** style at **`0`** index. So the newly loaded style attaches to it at **`1`** index.

As you update the mentioned additions to the TODO app's Python code and run it, you'll see the GUI application's different look and feel. Here, it changes as per the **`Glow.vsf`** style as follows:

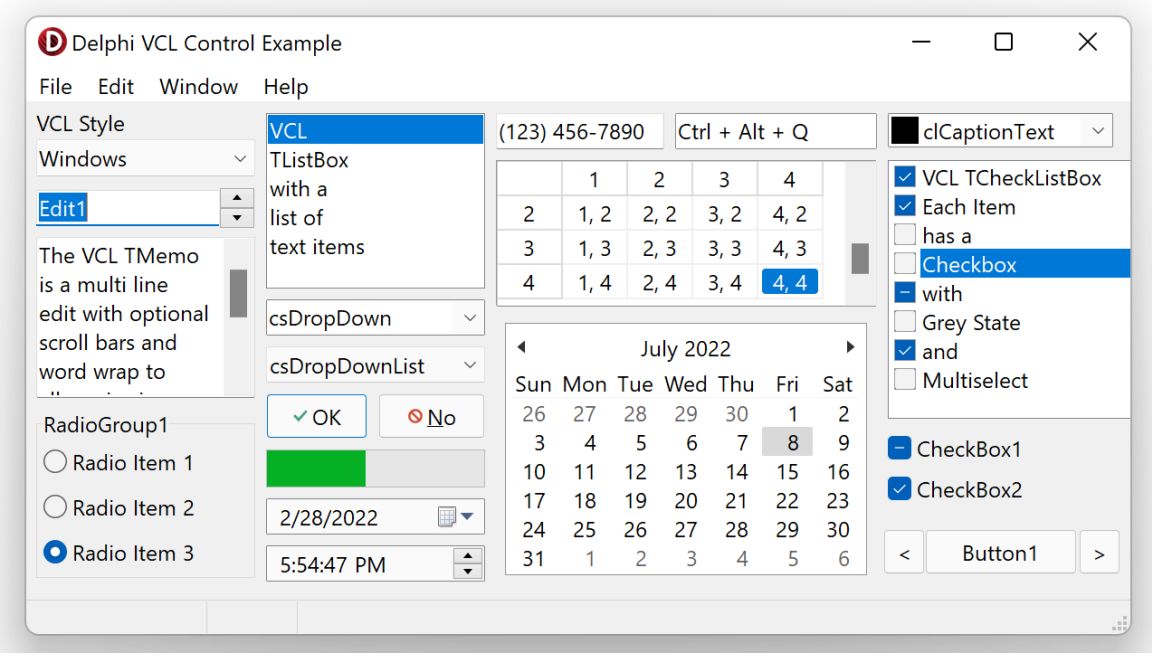


List of Included VCL Styles

The following VCL Styles are included with this quick start guide and may be distributed with the applications you build with DelphiVCL. The styles all support high-DPI resolutions.

Windows Default

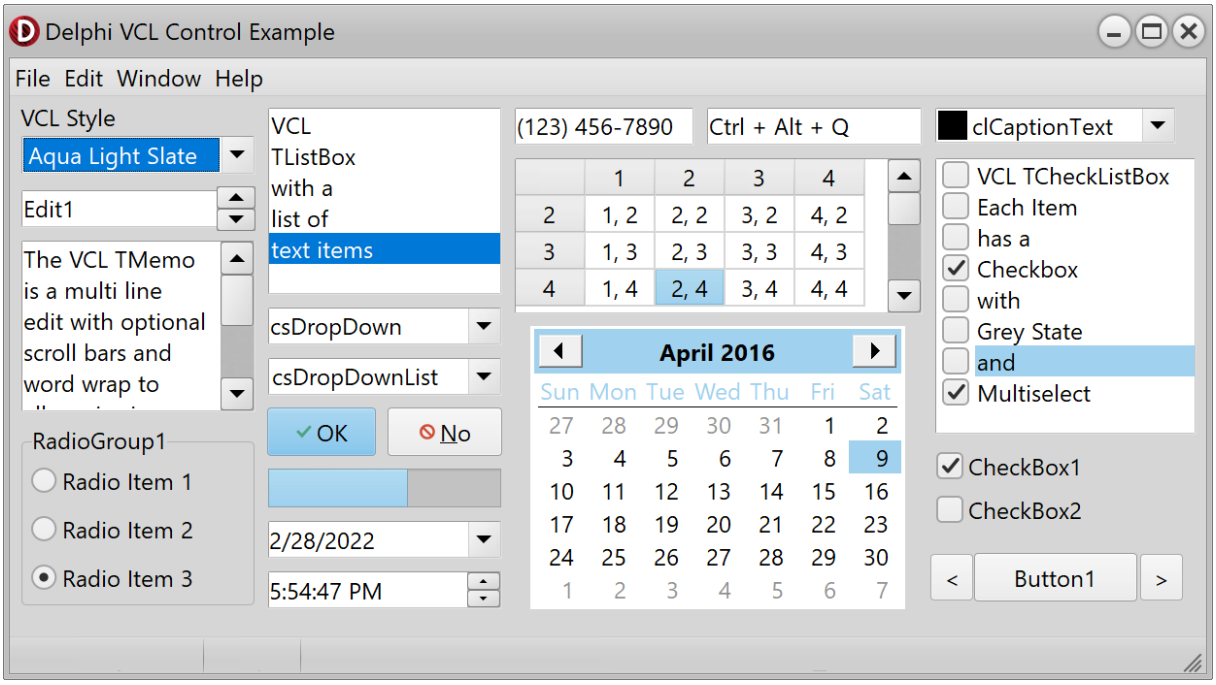
When no style file is included, your DelphiVCL application will use the default style.



This Quick Start Guide bundle includes the following styles. If you install Delphi, it comes with additional styles and the Bitmap Style Designer that allows you to create and customize styles. You can find out more about Embarcadero Delphi and its different editions at embarcadero.com/products/delphi

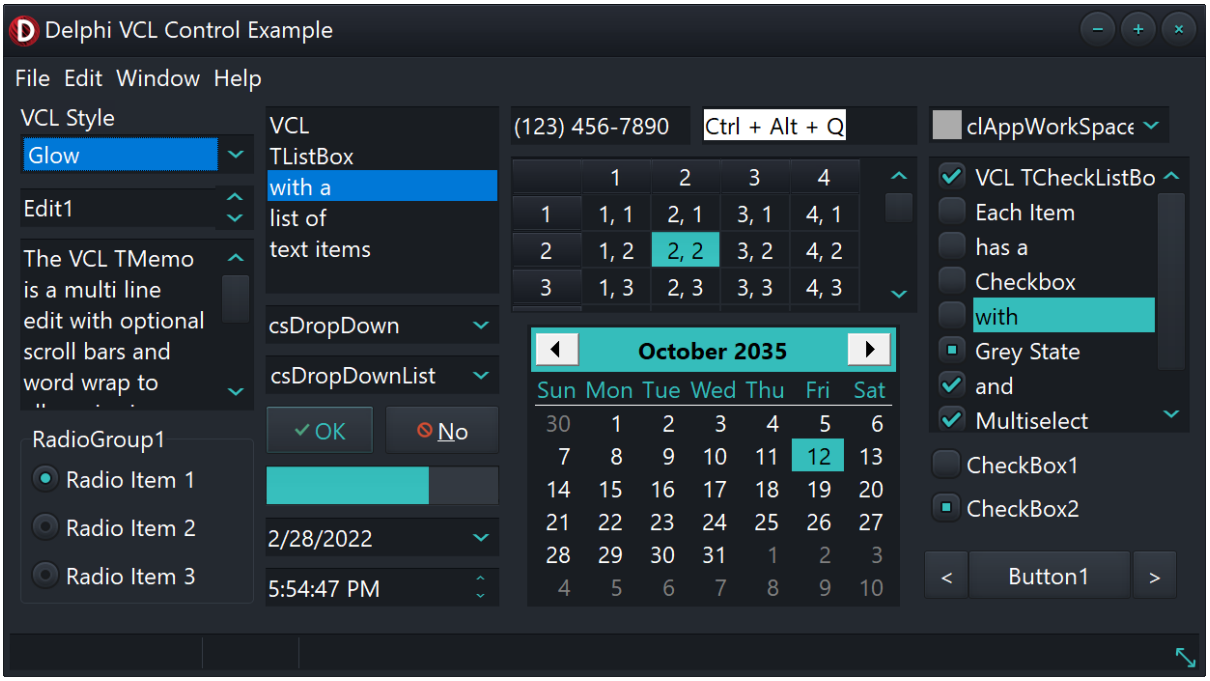
Aqua Light Slate

AquaLightSlate.vsf



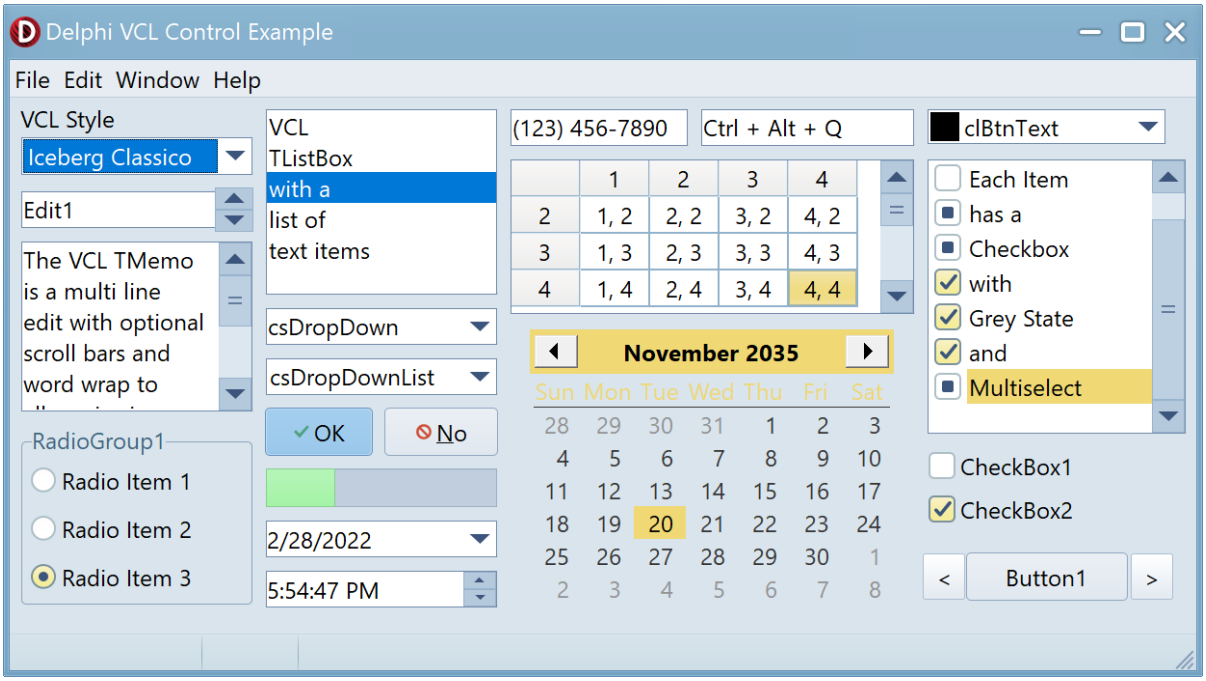
Glow

Glow.vsf



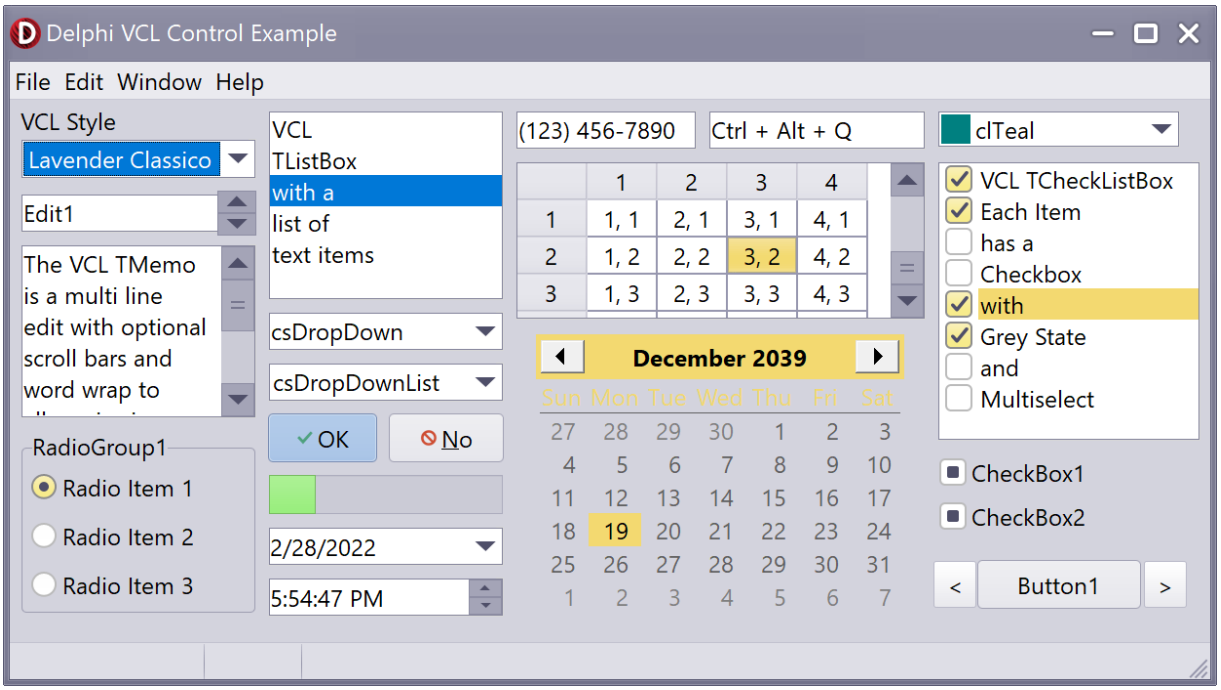
Iceberg Classico

IcebergClassico.vsf



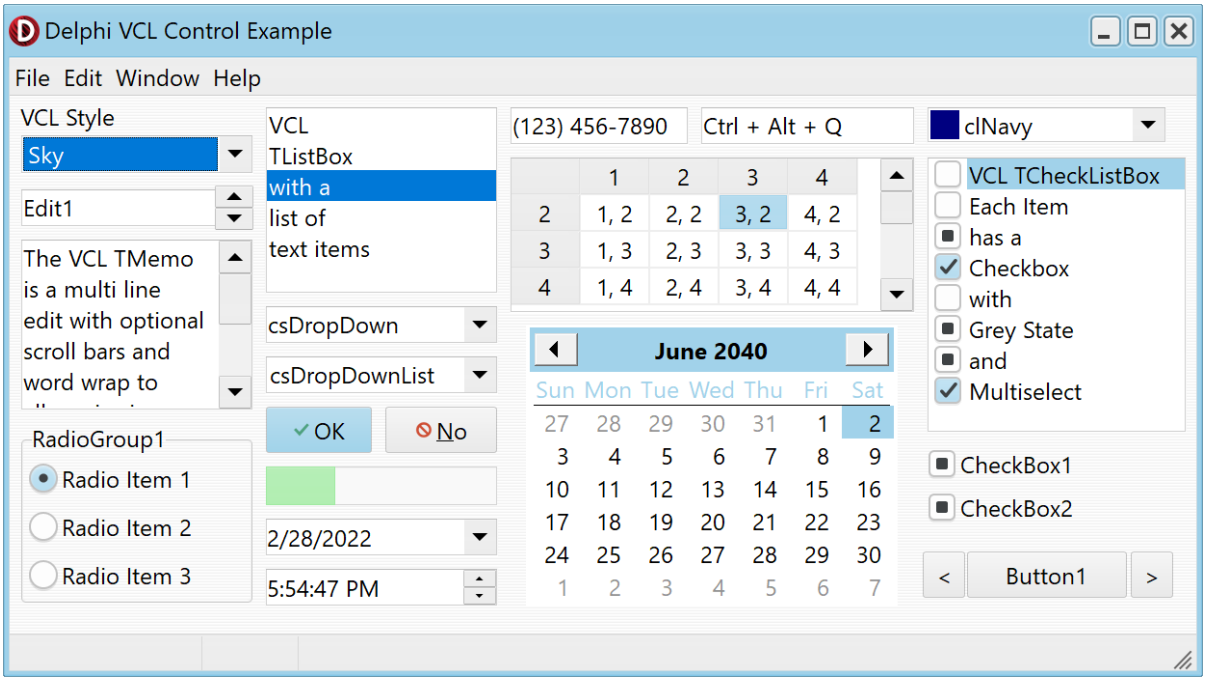
Lavender Classico

LavenderClassico.vsf



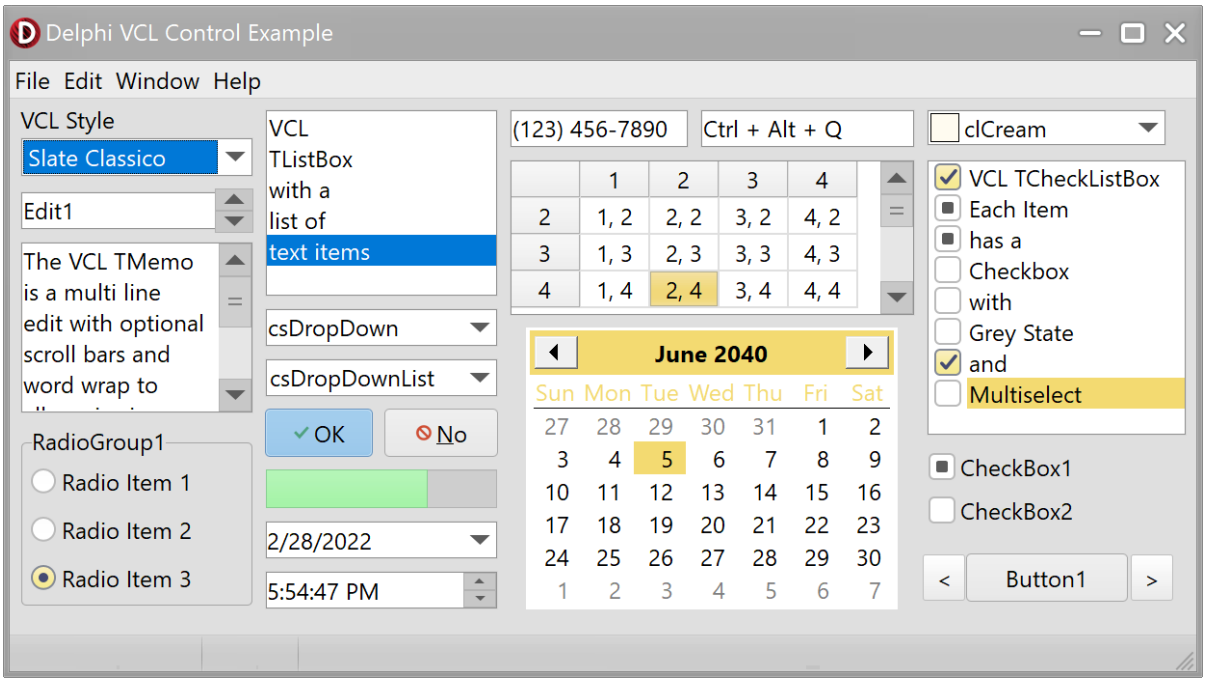
Sky

Sky.vsf



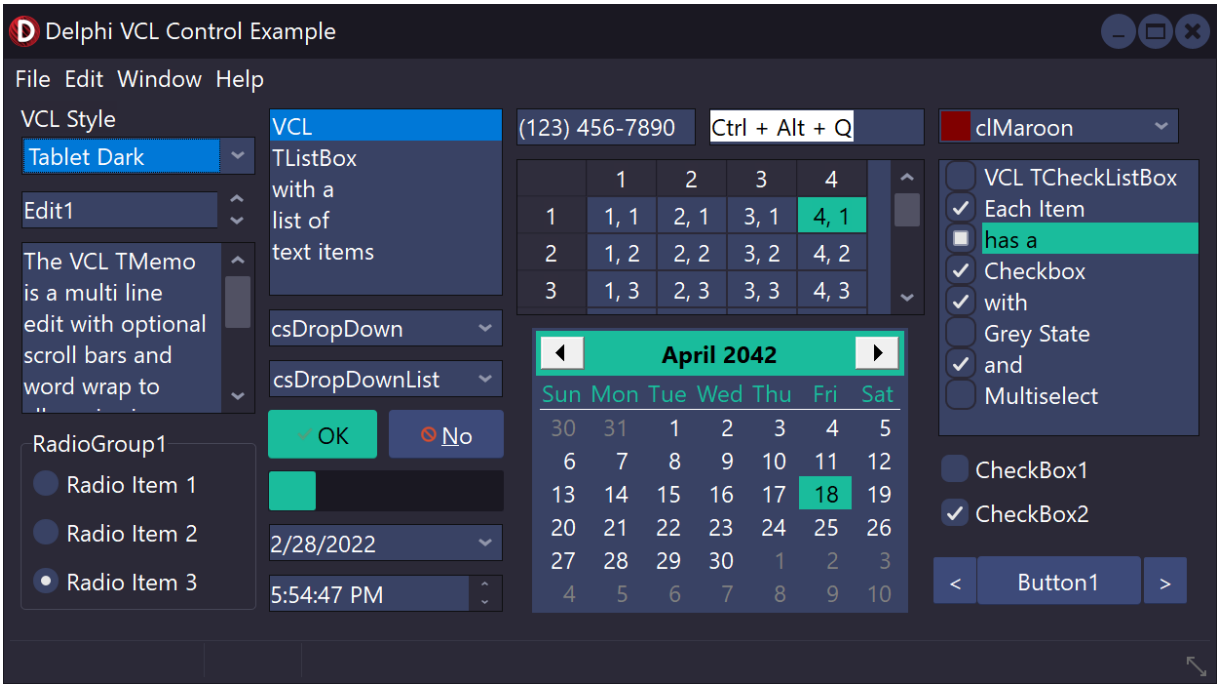
Slate Classico

SlateClassico.vsf



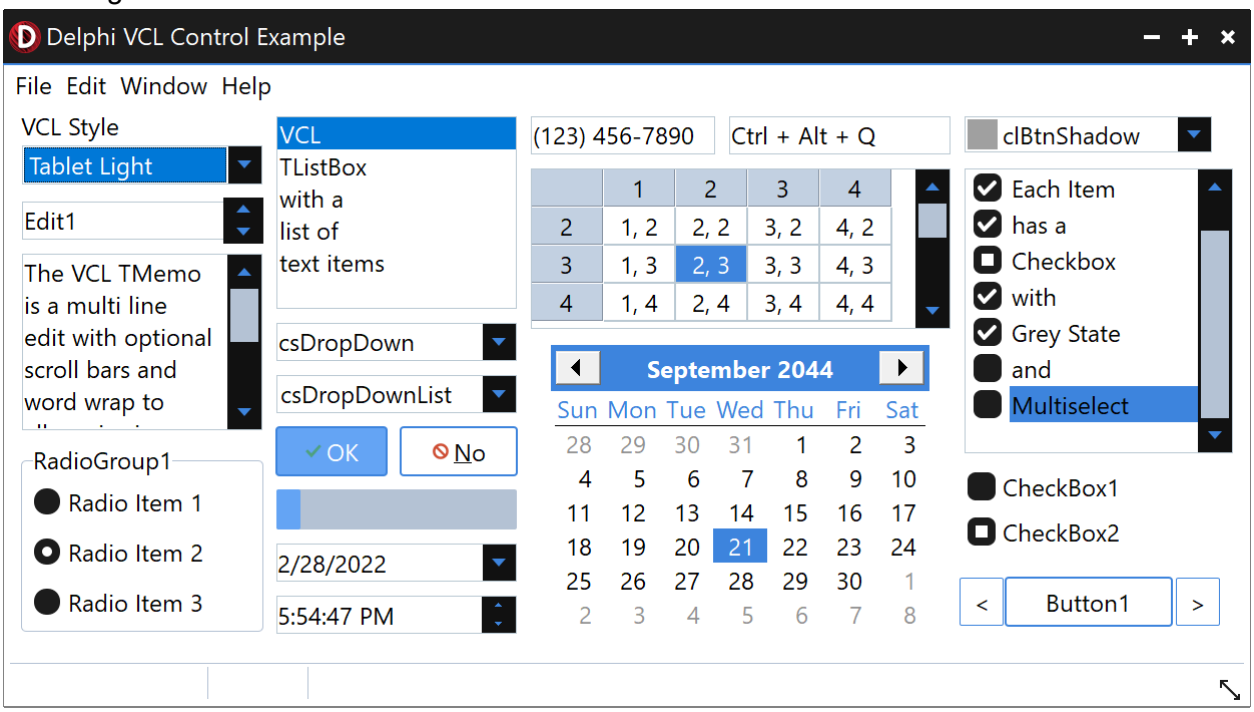
Tablet Dark

TabletDark.vsf



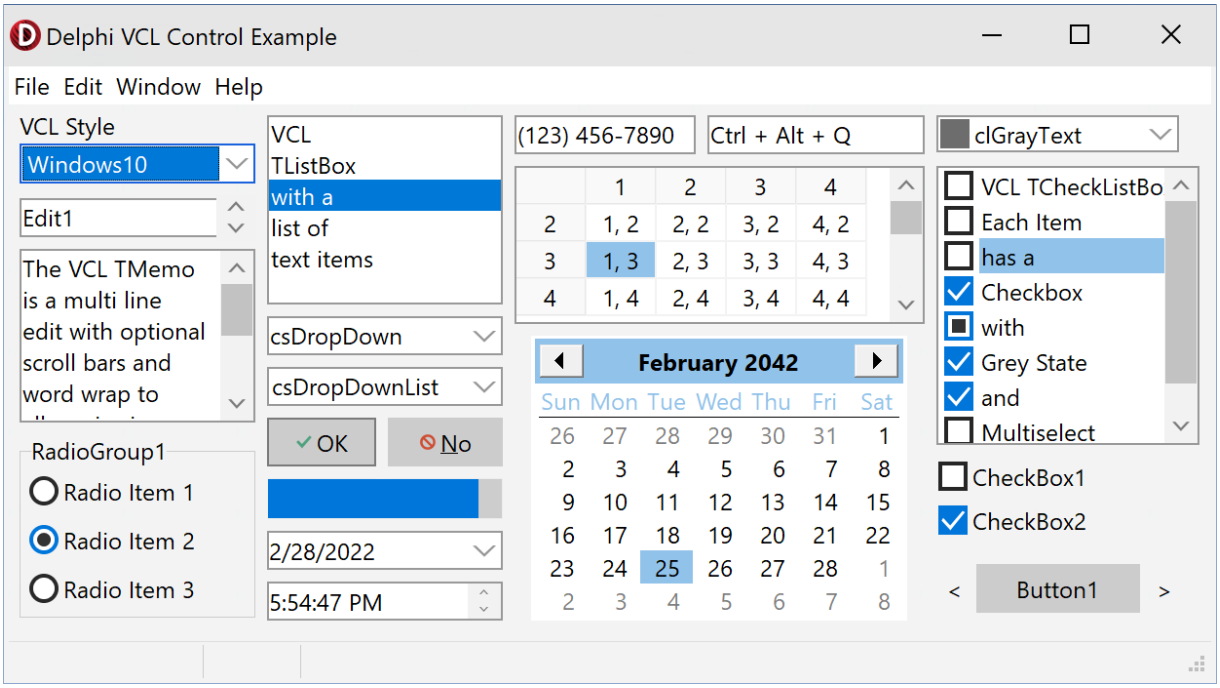
Tablet Light

TabletLight.vsf



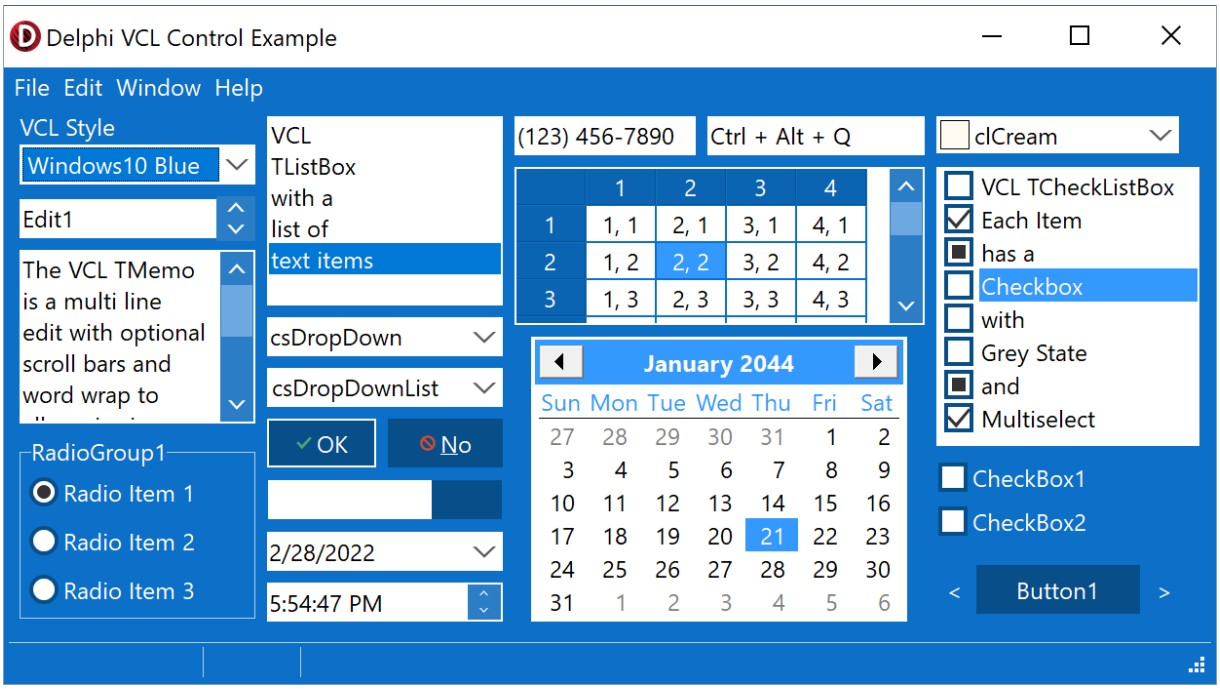
Windows 10

Windows10.vsf



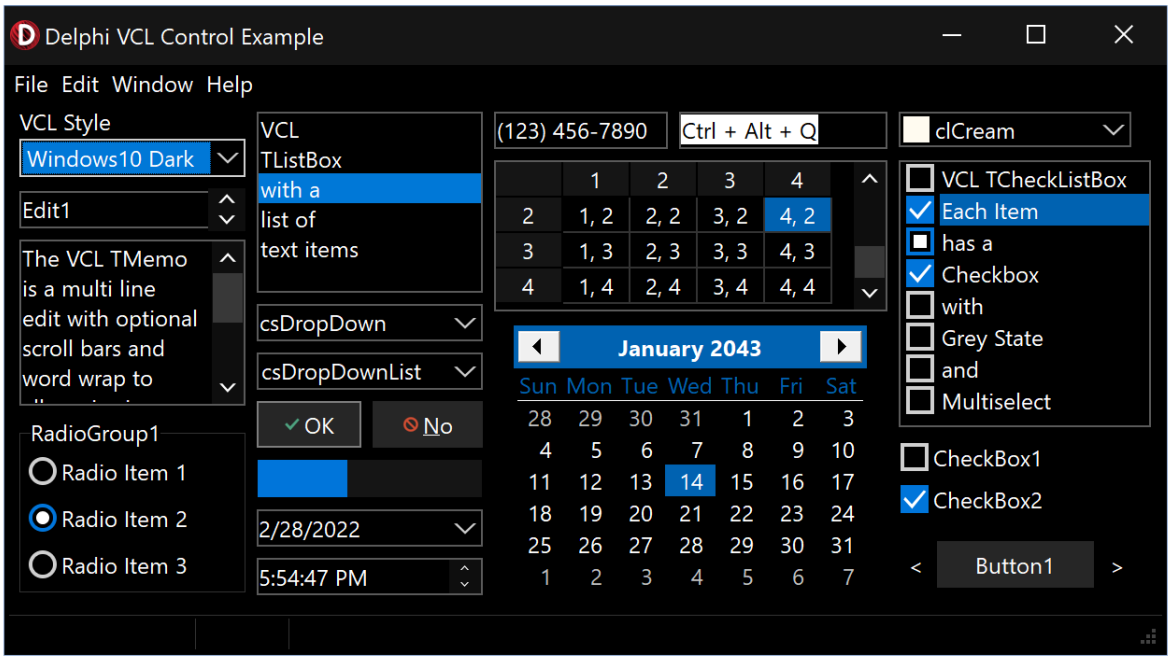
Windows 10 (Blue)

Windows10Blue.vsf



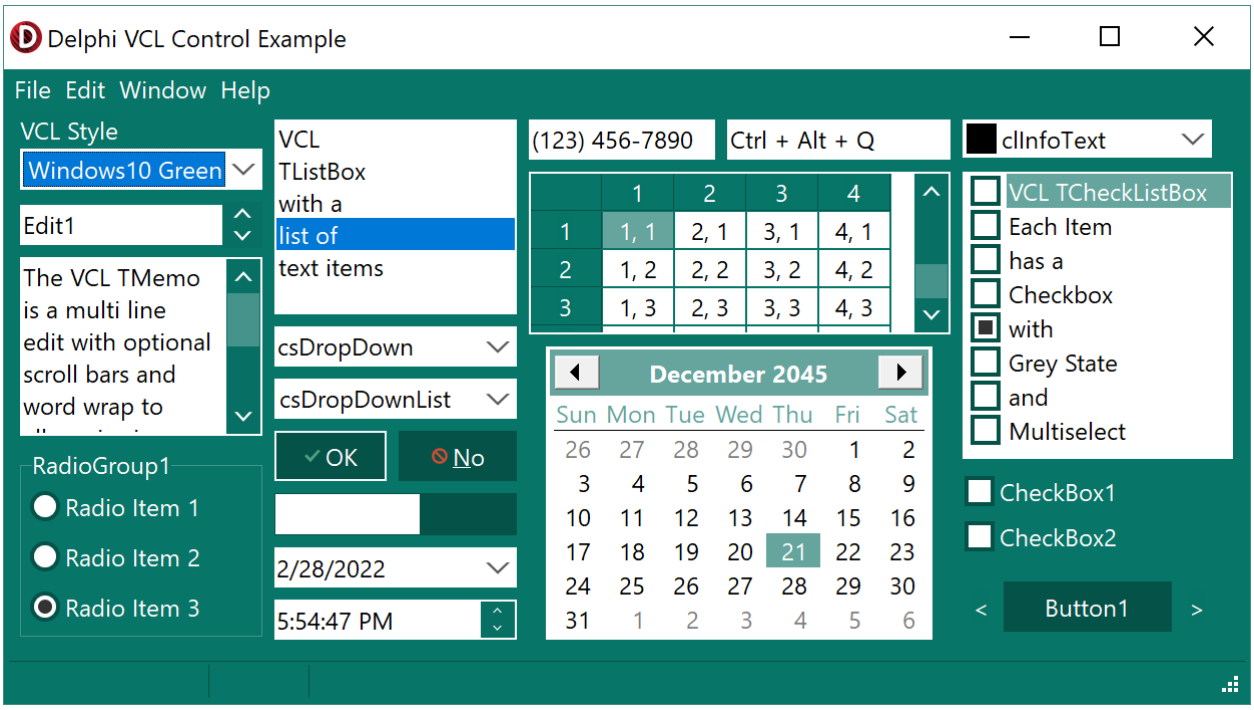
Windows 10 (Dark)

Windows10Dark.vsf



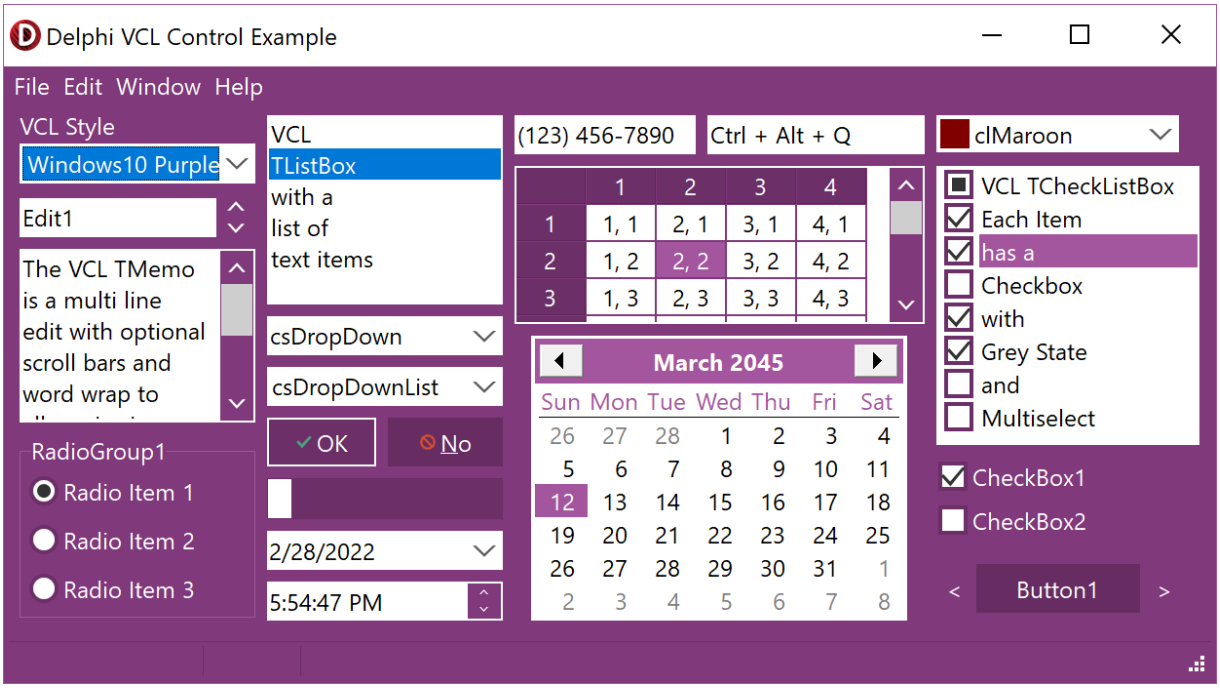
Windows 10 (Green)

Windows10Green.vsf



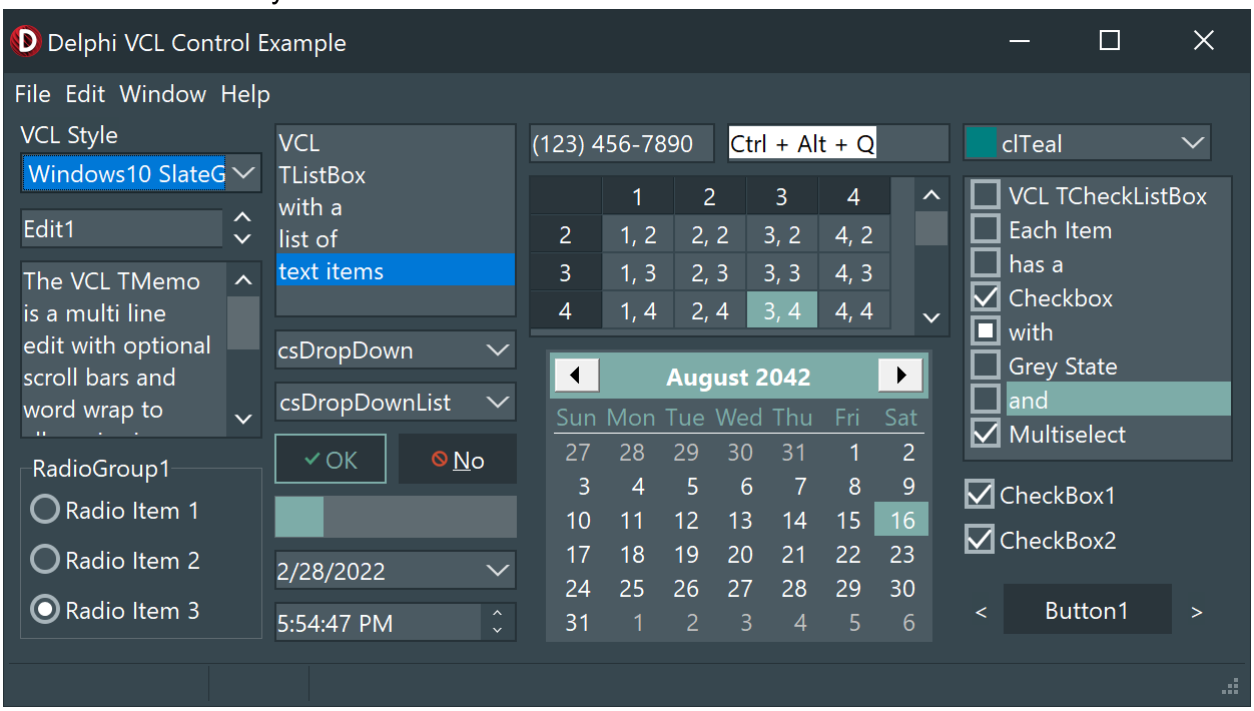
Windows 10 (Purple)

Windows10Purple.vsf



Windows 10 (Slate Gray)

Windows10SlateGray.vsf



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.

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.
More information embarcadero.com/products/delphi
Download a free trial embarcadero.com/products/delphi/start-for-free