PySide GUI Application Development
Second Edition

Elegantly-built GUI applications are always a massive hit among users. PySide is an open source software project that provides Python bindings for the Qt cross-platform UI framework. Combining the power of Qt and Python, PySide provides easy access to the Qt framework for developers and also acts as an excellent application development platform.

This book will take you through everything you need to know to develop UI applications. You will learn about installing and building with PySide in various major operating systems as well as the basics of GUI programming. The book will then move on to discuss event management, signals and slots, and the widgets and dialogs available with PySide. Database interaction and manipulation is also covered.

By the end of this book, you will be able to program GUI applications easily and will have mastered developing your own applications and running them across platforms.

Who this book is written for
This book is written for Python programmers who want to learn about GUI programming. It is also suitable for those who are new to Python but are familiar with object-oriented programming.

What you will learn from this book
- Program GUI applications in an easy and efficient way
- Download and install PySide, a cross-platform GUI development toolkit for Python
- Create menus, toolbars, status bars, and child windows
- Develop a text editor application on your own
- Connect your GUI to a database and manage it
- Execute SQL queries by handling databases
- Learn dialogs and widgets to build a complete and usable GUI application
- Discover how to create windows and modify them accordingly

Develop more dynamic and robust GUI applications using PySide, an open source cross-platform UI framework.
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Getting Started with PySide'
- A synopsis of the book’s content
- More information on *PySide GUI Application Development Second Edition*
About the Authors

**Gopinath Jaganmohan** is an Internet of Things evangelist and open source distributed-computing architect. He has 14 years of experience in various industries and has architected and lead implementation in Internet of Things platforms and solutions for enterprise in Telematics, Healthcare, and Wearables. He has worked on various technologies starting from C, Python, Lua, to Node.js, and implemented Big Data technologies like Hbase, Couchbase and ZooKeeper for various clients. He is passionate about device programming and device integration. He is lately working on deep learning technologies like Keras, scikit, Torch7 and Pandas in machine learning for device data.

**Venkateshwaran Loganathan** is an eminent software developer who has been involved in the design, development, and testing of software products for more than five years now. He was introduced to computer programming at an early age of 11 with FoxPro, and he then started to learn and master various computer languages, such as C, C++, Perl, Python, Node.js, and Unix shell scripting. Fascinated by open source development, he has involved himself in contributing to various open source technologies.

He is now working for Cognizant Technology Solutions as a technology specialist where he has involved himself in research and development for the Internet of Things domain. He is now actively involved in using RFID devices, Drones, and Google Glass to evolve Future of Technology concepts. Before joining with Cognizant, he worked with few of the IT majors, such as Infosys, Virtusa, and NuVeda. Starting his career as a network developer, he gained expertise in various domains, such as Networking, E-Learning, and HealthCare. He has won various awards and accolades to his merit in the companies he has worked for.
Venkateshwaran holds a bachelor’s degree in Computer Science and Engineering from Anna University and an M.S in software systems from BITS, Pilani. Apart from programming, he is actively involved in handling various technical and soft skills classes for budding engineers and college students. His hobbies include singing and trekking. He likes to get involved with social servicing and move with people a lot. You can write to him at anandvenkat4@gmail.com.
Preface

The aim of this book is to introduce you to developing GUI applications in an easy way. Python is easy to learn and use, and its programs are relatively short compared to those written in any other programming languages, such as C++, and Java. It is supported by a large set of dynamic libraries and bindings that make it efficient to develop very complex applications in an efficient manner. This book will introduce you to user interface programming and its components. You will be able to develop real-time applications in a shorter time after reading this book. The second edition.

What this book covers

Chapter 1, Getting Started with PySide, introduces you to GUI programming in general. This chapter takes you through the introduction of PySide and its installation in various major operating systems, followed by a short introduction to exception handling in programming. By the end of this chapter, users will know how to install and use PySide to create GUI applications in Python.

Chapter 2, Entering through Windows, introduces you to all the GUI programming that revolves around Windows. This chapter explains the basic methods of creating windows and adding some functions to them. By the end of this chapter, users will be familiar with how to create windows and modify them accordingly.

Chapter 3, Main Windows and Layout Management, elaborates further on the previous chapter by explaining how to create menus and tool bars for a windowed application. This also explains layout management policies. A simple text editor is given as an example at the end of the chapter. By the end of this chapter, readers have an experience of creating a real-time application in PySide.

Chapter 4, Events and Signals, this chapter goes on to explain the signals, various text and graphic effects, drag and drop, and a few geometrical diagram shapes. By the end of this chapter, readers will learn about managing events and various other text and graphical effects.
Preface

Chapter 5, *Dialogs and Widgets*, details the built-in dialog boxes for applications, introduces how to create customized dialogs, and then takes a look at the various widgets that are available in PySide. By the end of this chapter, you will learn about creating your own customized widgets and dialogs.

Chapter 6, *Database Handling*, explains how connecting to a database is evident for almost all applications. This chapter is dedicated to explaining how to connect to a database and execute queries on it. It also deals with the presentation of data in table and form views. By the end of this chapter, you will know more about interacting with databases and viewing data from them.
Getting Started with PySide

Python is a general-purpose, interpreted, object-oriented, and high-level programming language with dynamic semantics. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. It is one of the most preferred programming languages by software developers due to its interpreted nature and its elegant syntax.

The success of Python lies in its simple and easy-to-learn syntax and the support of a wide variety of modules and packages that encourage program modularity and code reuse. Being an interpreted language, there is no compilation step, which makes the edit-test-debug cycle incredibly fast, paving the way to Rapid Application Development, the need of the hour. The support of object-oriented features and high-level data structures, such as generators and list comprehensions, makes Python a superior language for coding small scripting programs to more advanced game programming.

This book assumes that you have been acquainted with Python and want to test its capability in creating GUI applications. However, Python is easy to learn in just a week. If you already know programming, then learning Python will be like walking in the park for you. There are many resources available online and offline covering a wide range of topics. Being an open source language, Python is also supported by many programmers around the globe in the IRC system under the tag #python.

Python is named after the BBC show Monty Python's Flying Circus and has nothing to do with reptiles. Thus, making references to Monty Python skits in documentation is practiced and encouraged.

The Python newsgroup, comp.lang.python, and mailing list python-list at https://mail.python.org/mailman/listinfo/python-list will help you learn and explore Python.
Introducing PySide

Many of the modern programming languages are backed up by a set of libraries (commonly referred to as toolkits) to create GUI applications, such as Qt, Tcl/Tk, and so on. PySide is a Python binding of the cross-platform GUI toolkit Qt, and it runs on all platforms that are supported by Qt, including Windows, Mac OS X, and Linux. It is one of the alternatives to toolkits such as Tkinter for GUI programming in Python.

PySide combines the advantages of Qt and Python. A PySide programmer has all the power of Qt, but it is able to exploit it with the simplicity of Python. PySide is licensed under the LGPL version 2.1 license, allowing both Free/Open Source software and proprietary software development. PySide is evolving continuously, like any other open source product, and you are free to contribute to its development. Some of the applications, such as matplotlib, PhotoGrabber, QBitTorrent, Lucas Chess, Fminer and so on, certify the wide spread usage of PySide in the software industry.

The IRC channel for PySide is #pyside at Freenode.

PySide has also become an enabler of mobile development. Qt Mobility is a project that is creating a new suite of Qt APIs for mobile device functionality. The project Pyside Mobility is a set of bindings that allows Python to access the Qt Mobility API. The Qt Mobility API enables the developer to access the bread and butter of services provided by the underlying operating system that are essential for any mobile application. Learning PySide, you learn this for free. Without further ado, let’s get hacking!

Hello, GUI

In computing terms, GUI (pronounced as gooey, or Graphical User Interface) is used to denote a set of interfaces with computing systems that involves user-friendly images rather than boring text commands. GUI comes to the rescue of the numerous command-line interfaces that have always been coupled with a steep learning curve because learning and mastering commands requires a lot of effort due to their nonintuitive nature. Moreover, GUI layers make it easy for the end users to fulfill their needs without knowing much about the underlying implementation, which is unnecessary for them.
Every other application in the modern world is designed with interactive graphics
to attract the end users. Simplicity and usability are the two main ingredients for
a successful GUI system. The demanding feature of a GUI is to allow the user to
concentrate on the task at hand. To achieve this, it must serve the interaction between
the human and the computer, and make it no less than seamless and flowing.
Therefore, learning to create GUls will not only make you a successful developer, but
it will also help in getting some revenue for yourself.

At a very basic level, a GUI is seen as a window (visibly noticeable or not) consisting
of the following parts: controls, menu, layout, and interaction. A GUI is represented
as a window on the screen and contains a number of different controls, as follows:

- **Controls**: These can, for example, be labels, buttons or text boxes.
- **Menu**: This is usually situated under the top frame of the GUI window and
  presents to the users some choices to control the application. The top frame
can also have buttons to hide, resize, or destroy the windows, which are,
  again, controls.
- **Layout**: This is the way that the controls are positioned, which is very
  important in good GUI design.
- **Interaction**: This happens in the way of I/O devices, such as a mouse and
  keyboard.

Development of a GUI application revolves around defining and controlling these
components, and designing the area of interaction is the most challenging part of
all. The correct exploitation of events, listeners, and handlers will help in developing
better GUI applications. Many frameworks have been developed to support GUI
development, such as the Model-View-Controller framework that is used in many
web-based applications. Using some of these frameworks can make the GUI
programming easier and will come in handy for future implementations. A good
user-interface design relates to the user, not to the system architecture.

Usually, GUls are characterized by 2W’s, namely WIMP and WYSIWYG. They are
acronyms for Windows, Icons, Menus, Pointing devices (mouse, joystick, and so on)
Setting up PySide
This is your first step in this series of learning. PySide is compatible with Python 2.6 or later and Qt 4.6 or better. So, before getting to install PySide, we must make sure that minimum version compatibility is achieved. This section will teach you two ways of installing PySide. One, being the most common and easiest way, is using simple point and click installers and package managers. This will install the most stable version of PySide on your system, which you can comfortably use without worrying too much about the stability. However, if you are an advanced programmer, you may prefer to build PySide from scratch from the latest builds that are available when you are reading this book. Both these methods are explained here for Windows, Mac OS X, and Linux systems, and you are free to choose your own setup style.

Installing PySide using Windows
Installation of PySide on Windows is pretty much easy with the help of an installer. Perform the following steps for setup:

1. Get the latest stable package matching your Operating System architecture and the Python version installed from the releases page at http://qt-project.org/wiki/PySide_Binaries_Windows
2. Run the downloaded installer executable, which will automatically detect the Python installation from your system
3. You are given an option to install PySide on the default path or at the path of your choice
4. On clicking Next in the subsequent windows, and finally clicking Finish, PySide is installed successfully on your system

Installing PySide using Mac OS X
The binaries for MAC OS X installers of PySide are available at:
http://qt-project.org/wiki/PySide_Binaries_MacOSX
Download the latest version that is compatible with your system and perform a similar installation as explained in the previous section.
You can also choose to install PySide from the command line with the help of Homebrew or using MacPorts. The commands, respectively, are as follows:

```bash
brew install pyside
port-install pyXX-pyside
```

Replace `XX` with your Python version.

## Installing PySide using Linux

Installing PySide on a Debian-based system is much easier with the synaptic package manager. Issuing the following command will fetch and install the latest stable version available in the aptitude distribution:

```bash
sudo apt-get install python-pyside
```

On an RPM-based system, you can use the RPM-based distribution, yum, as follows:

```bash
yum install python-pyside pyside-tools
```

If you want to make sure that PySide is installed properly on your system, issue the following commands in the Python shell environment, as shown in Figure 1. The `import pyside` command should not return any errors.

PySide.__version__ should output something similar to 1.1.2:

![Python Shell](image_url)

Let's move on to see how we can build PySide from scratch.
Building PySide on Windows

Before starting to build PySide on Windows, ensure that the following prerequisites are installed:

- Visual Studio Express 2008 (Python 2.6, 2.7, or 3.2) / Visual Studio Express 2010 (Python 3.3) [http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products]
- CMake [http://www.cmake.org/cmake/resources/software.html]
- Git [http://git-scm.com/download/win]
- Python 2.6, 2.7, 3.2, or 3.3 [http://www.python.org/download/]
- OpenSSL [http://slproweb.com/products/Win32OpenSSL.html] (Optional)

Make sure that the Git and cmake executables are set in your system path. Now, perform the following steps to start building PySide:

1. Git Clone the PySide repository from GitHub, as follows:
   c:\> git clone https://github.com/PySide/pyside-setup.git pyside-setup

2. Change your working directory to pyside-setup, as follows:
   c:\> cd pyside-setup

3. Build the installer:
   c:\> c:\Python27\python.exe setup.py bdist_wininst --msvc-version=9.0 --make=c:\Qt\4.8.4\bin\qmake.exe --openssl=c:\OpenSSL32bit\bin

4. Upon successful installation, the binaries can be found in the dist sub-folder:
   c:\pyside-setup\dist

On completion of these steps, the PySide should have been successfully built on your system.
Building PySide on Linux

The following are the prerequisites to build PySide in Linux:

Prerequisites

- CMake version 2.6.0 or higher [http://www.cmake.org/cmake/resources/software.html]
- Qt libraries and development headers version 4.6 or higher [http://origin.releases.qt-project.org/qt4/source/qt-everywhere-opensource-src-4.8.4.tar.gz]
- libxml2 and development headers version 2.6.32 or higher [http://www.xmlsoft.org/downloads.html]
- libxslt and development headers version 1.1.19 or higher [http://xmlsoft.org/XSLT/downloads.html]
- Python libraries and development headers version 2.5 or higher [http://www.python.org/download/]

Building PySide

PySide is a collection of four interdependent packages, namely API Extractor, Generator Runner, Shiboken Generator, and Pyside Qt bindings. In order to build PySide, you have to download and install these packages in that order:

- **API Extractor**: This is a set of libraries that is used by the binding generator to parse the header and type system files to create an internal representation of the API [https://distfiles.macports.org/apiextractor/].
- **Generator Runner**: This is the program that controls the bindings generation process according to the rules given by the user through headers, type system files, and generator frontends. It is dependent on the API Extractor [https://distfiles.macports.org/generatorrunner/].
- **Shiboken Generator**: This is the plugin that creates the PySide bindings source files from Qt headers and auxiliary files (type systems, global.h, and glue files). It is dependent on Generator Runner and API Extractor [https://distfiles.macports.org/py-shiboken/].
- **PySide Qt Bindings**: This is a set of type system definitions and glue codes that allows generation of Python Qt binding modules using the PySide tool chain. It is dependent on Shiboken and Generator Runner [https://distfiles.macports.org/py-pyside/].
Getting Started with PySide

Always, make sure that you have downloaded and built these packages in this order because each of these packages is interdependent. The build steps for each of these are:

1. Unzip the downloaded packages and change into the package directory:
   
   ```
   tar -xvf <package_name>
   cd <package_directory>
   ```

2. Create a build directory under the package directory and enter that directory:
   
   ```
   mkdir build && cd build
   ```

3. Make the build using cmake:
   
   ```
   cmake .. && make
   ```

4. On a successful make, build and install the package:
   
   ```
   sudo make install
   ```
   Please note that you require sudo permissions to install the packages.

5. To update the runtime linker cache, issue the following command:
   
   ```
   sudo ldconfig
   ```

Once you complete these steps in this order for each of these packages, PySide should be successfully built on your system.

Mac OS X

Building PySide on a Mac system follows the same procedure as the Linux system except that Mac needs Xcode-Developer Tools to be installed as a prerequisite.

If you are installing the libraries in a nondefault system directory (other than /usr/local), you may have to update the DYLD_LIBRARY_PATH by typing the following command:

```
export DYLD_LIBRARY_PATH=/path/to/lib
```

Importing PySide objects

Congratulations on setting up PySide successfully on your system. Now, it's time to do some real work using PySide. We have set up PySide and now we want to use it in our application. To do this, you have to import the PySide modules in your program to access the PySide data and functions. Here, let's learn some basics of importing modules in your Python program.
There are basically two ways that are widely followed when importing modules in Python. The first is to use a direct `import <module>` statement. This statement will import the module and creates a reference to the `module` in the current namespace. If you have to refer to entities (functions and data) that are defined in module, you can use `module.function`. The second is to use `from module import*`. This statement will import all of the entities that the module provides and set up references in the current namespace to all the public objects defined by that module. In this case, referencing an object within the module will boil down to simply stating its literal name in code.

Therefore, in order to use PySide functions and data in your program, you have to import it by saying either `import PySide` or `from PySide import*`. In the former case, if you have to refer to some function from PySide you have to prefix it with PySide, such as `PySide.<function_name>`. In the latter, you can simply call the function by `<function_name>`. Also, please note that in the latter statement, `*` can be replaced by specific functions or objects. The use of `*` denotes that we are trying to import all the available functions from that module. Throughout this book, I would prefer to use the latter format as I do not have to prefix the module name every time when I have to refer to something inside that module.

**First PySide application**

It's time to roll up our sleeves and get our hands dirty with some real coding now. We are going to learn how to create our first and the traditional Hello World application. Have a look at the code first, and we will dissect the program line by line for a complete explanation of what it does. The code may look a little strange to you at first but you will gain understanding as we move through:

```python
# Import the necessary modules required
import sys
from PySide.QtCore import *
from PySide.QtGui import *

# Main Function
if __name__ == '__main__':

    # Create the main application
    myApp = QApplication(sys.argv)

    # Create a Label and set its properties
    appLabel = QLabel()
    appLabel.setText("Hello, World!!!\nLook at my first app using PySide")
```

```
Getting Started with PySide

appLabel.setAlignment(Qt.AlignCenter)
appLabel.setWindowTitle("My First Application")
appLabel.setGeometry(300, 300, 250, 175)

# Show the Label
appLabel.show()

# Execute the Application and Exit
myApp.exec_()
sys.exit()

On interpretation, you will get an output window, as shown in the figure:

On interpretation, you will get an output window, as shown in the figure:

Now, let's get into the working of the code. We start with importing the necessary objects into the program.

Lines 1, 2 and 3 imports the necessary modules that are required for the program. Python is supported with a library of standard modules that are built into the interpreter and provide access to operations that are not a part of the core language. One such standard module is `sys`, which provides access to some variables and functions that are used closely by the interpreter. In the preceding program, we need the `sys` module to pass command-line arguments `sys.argv` as a parameter to the `QApplication` class. It contains the list of command-line arguments that are passed to a Python script. Any basic GUI application that uses PySide should have two classes imported for basic functionality. They are `QtCore` and `QtGui`. The `QtCore` module contains functions that handle signals and slots and overall control of the application, whereas `QtGui` contains methods to create and modify various GUI window components and widgets.
In the main program, we are creating an instance of the `QApplication` class. `QApplication` creates the main event loop, where all events from the window system and other sources are processed and dispatched. This class is responsible for an application's initialization, finalization, and session management. It also handles the events and sets the application's look and feel. It parses the command-line arguments (`sys.argv`) and sets its internal state, accordingly. There should be only one `QApplication` object in the whole application even though the application creates one or many windows at any point in time.

Once the main application instance is created, we move on by creating a `QLabel` instance that will display the required message on the screen. This class is used to display a text or an image. The appearance of the text or image can be controlled in many ways by the functions provided by this class. The next two lines that follow the instantiation of this class set the text to be displayed and align it in a way that is centered on the application window.

As Python is an object-oriented programming language, we take the advantage of many object-oriented features, such as polymorphism, inheritance, object initialization, and so on. The complete Qt modules are designed in an object-oriented paradigm that supports these features. `QLabel` is a base class that is inherited from the `QFrame` super class whose parent class is `QWidget` (the details will be covered in forthcoming chapters). So, the functions that are available in `QWidget` and `QFrame` are inherited to `QLabel`. The two functions, `setWindowTitle` and `setGeometry`, are functions of `QWidget`, which are inherited by the `QLabel` class. These are used to set the title of the window and position it on the screen.

Now that all the instantiation and setup is done, we are calling the show function of the `QLabel` object to present the label on the screen. At this point only, the label becomes visible to the user and they are able to view it on the screen. Finally, we call the `exec_()` function of the `QApplication` object, which will enter the Qt main loop and start executing the Qt code. In reality, this is where the label will be shown to the user but the details can be safely ignored as of now. Finally, we exit the program by calling `sys.exit()`.
Getting Started with PySide

Exception handling as a practice

It is not always possible to foresee all the errors in your programs and deal with them. Python comes with an excellent feature called exception handling to deal with all runtime errors. The aim of the book is not to explain this feature in detail but to give you some basic ideas so that you can implement it in the code that you write.

In general, the exceptions that are captured while executing a program are handled by saving the current state of the execution in a predefined place and switching the execution to a specific subroutine known as exception handler. Once they are handled successfully, the program takes the normal execution flow using the saved information. Sometimes, the normal flow may be hindered due to some exceptions that could not be resolved transparently. In any case, exception handling provides a mechanism for smooth flow of the program altogether.

In Python, the exception handling is carried out in a set of try and except statements. The try statements consist of a set of suspicious code that we think may cause an exception. On hitting an exception, the statement control is transferred to the except block where we can have a set of statements that handles the exception and resolves it for a normal execution of a program. The syntax for the same is as follows:

```
try : suite
except exception <, target> : suite
except : suite
```

Here, suite is an indented block of statements. We can also have a set of try, except block in a try suite. The former except statement provides a specific exception class that can be matched with the exception that is raised. The latter except statement is a general clause that is used to handle a catch-all version. It is always advisable to write our code in the exception encapsulation.

In the previous example, consider that we have missed instantiating the appLabel object. This might cause an exception confronting to a class of exception called NameError. If we did not encapsulate our code within the try block, this raises a runtime error. However, if we had put our code in a try block, an exception can be raised and handled separately, which will not cause any hindrance to the normal execution of the program. The following set of code explains this with the possible output:

```
# Import the necessary modules required
import sys
from PySide.QtCore import *
from PySide.QtGui import *
```
# Main Function

```python
if __name__ == '__main__':

    # Create the main application
    myApp = QApplication(sys.argv)

    # Create a Label and set its properties
    try:
        appLabel = QLabel()
        appLabel.setText("Hello, World!!!\nLook at my first app using PySide")
        appLabel.setAlignment(Qt.AlignCenter)
        appLabel.setWindowTitle("My First Application")
        appLabel.setGeometry(300, 300, 250, 175)

        # Show the Label
        appLabel.show()

        # Execute the Application and Exit
        myApp.exec_()
        sys.exit()
    except NameError:
        print("Name Error:", sys.exc_info()[1])
        pass
```

In the preceding program, if we did not handle the exceptions, the output would be as shown in the figure:
Conversely, if we execute the preceding code, we will not run into any of the errors shown in the preceding figure. Instead, we will have captured the exception and given some information about it to the user, as follows:

Hence, it is always advised to implement exception handling as a good practice in your code.

**Summary**

The combination of Qt with Python provides the flexibility of Qt developers, develops GUI programs in a more robust language, and presents a rapid application development platform available on all major operating systems. We introduced to you the basics of PySide and its installation procedure on Windows, Linux, and Mac systems. We went on to create our first application, which introduced the main components of creating a GUI application and the event loop. We have concluded this chapter with an awareness on how to introduce exception handling as a best practice. Moving on, we are set to create some real-time applications in PySide.
Where to buy this book

You can buy PySide GUI Application Development Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.