



Programmation Orientée Objet (Python)

Mini-Projet instrumentation en Python : partie 2

[A.Martin](#)

2026-2027



Visual Studio Code



Objectif

Ce second volet du mini-projet a pour objectif de prolonger le travail de programmation orientée objet réalisé autour du Rigol DHO924S en construisant une **interface graphique** qui s'appuie sur les classes déjà développées, puis en livrant l'application sous forme d'**exécutable autonome**.

À la fin, vous saurez :

- Concevoir l'interface d'une application avec **Qt Widgets Designer** (fenêtres, widgets, layouts, signaux et slots) ;
- Convertir le fichier d'interface (.ui) en code Python ;
- Séparer clairement la **couche interface** de la **couche métier** (les classes d'appareils : Oscillo, GBF, etc.) et relier les deux par des signaux/slots ;
- Piloter l'appareil depuis l'interface (connexion VISA, réglage des paramètres, acquisition, affichage des mesures) ;
- Gérer les erreurs et l'état de la connexion (appareil absent, commande refusée, déconnexion) de façon lisible pour l'utilisateur ;
- Tester l'application avec l'appareil réel ;
- Générer un **exécutable** (avec PyInstaller) et le déployer sur un poste sans environnement Python.

Durée estimée : 4 heures

1. Partie théorique

1.1 Qu'est-ce que PySide6 ?

PySide6 est la bibliothèque Python officielle du framework Qt 6. Un **framework** est un ensemble structuré de bibliothèques, d'outils et de conventions qui fournit l'architecture de base d'une application, que le développeur complète avec son propre code. PySide6 est développée et maintenue par The Qt Company dans le cadre du projet **Qt for Python**. Elle donne accès en Python à presque toutes les classes de Qt : fenêtres, widgets, gestion des événements, graphiques, communication série, réseau, etc.

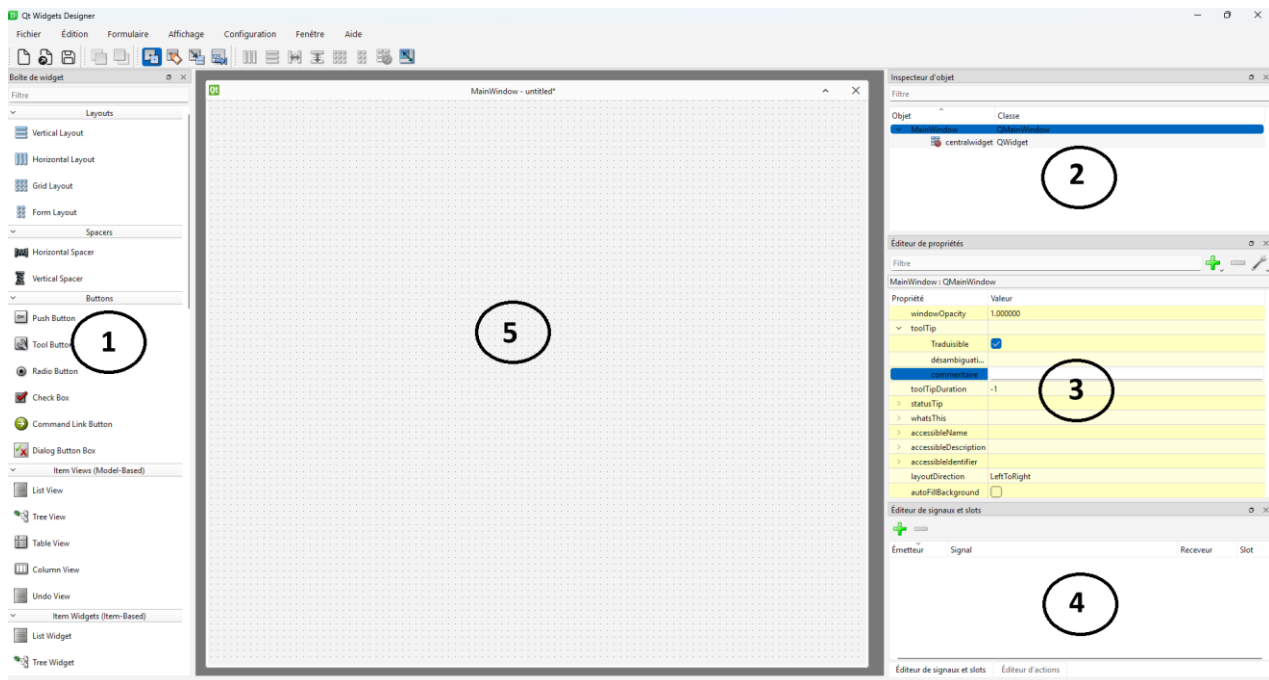
Qt étant écrit en C++, PySide6 sert de passerelle entre les deux langages : on écrit du code Python, et les composants graphiques sont ceux de Qt, avec leur rapidité d'exécution.

1.2 Qu'est-ce que Qt Widgets Designer ?

Qt Widgets Designer est l'outil de conception graphique du framework Qt. Il permet de construire l'interface d'une application en plaçant les composants à la souris (glisser-déposer), sans écrire de code pour la mise en page. Le résultat est visible immédiatement, ce qui accélère la création et facilite les modifications.

On compose la fenêtre à partir de **widgets** (boutons, champs de saisie, listes déroulantes, zones d'affichage, étiquettes, etc.) placés dans une fenêtre principale (classe **QMainWindow**, qui hérite de **QWidget**, lui-même héritant de **QObject**) ou dans une boîte de dialogue (classe **QDialog**). Les widgets sont organisés par des **layouts** (horizontal, vertical, grille, formulaire), qui adaptent automatiquement leur taille et leur position quand la fenêtre est redimensionnée.

Qt Widgets Designer est organisé en plusieurs panneaux (voir la capture ci-dessous) :



1. La boîte de widgets, qui regroupe les composants disponibles ;
2. L'inspecteur d'objets, qui montre l'arborescence des widgets de la fenêtre ;
3. L'éditeur de propriétés, qui modifie les **attributs** de l'objet sélectionné (nom, texte, taille, état, etc.) ;
4. L'éditeur de **signaux/slots**, pour relier un événement (clic sur un bouton, changement de valeur) à une action ;
5. La zone de conception : la fenêtre en cours de création.

Le travail est enregistré dans un fichier **.ui** (User Interface) au format XML. Ce fichier décrit uniquement l'interface (les widgets, leurs propriétés, leur organisation) et ne contient aucune logique métier. Cette séparation est au cœur du projet : l'aspect visuel est défini dans le fichier **.ui**, tandis que le comportement (le pilotage de l'oscilloscope et du GBF) reste dans les classes Python déjà écrites.

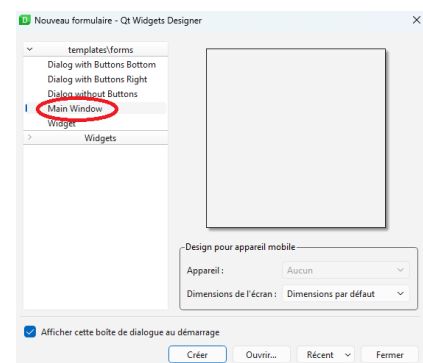
2. Travail à réaliser

2.1 Étape préparatoire

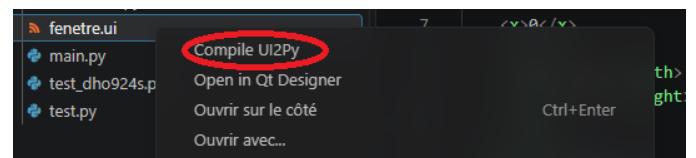
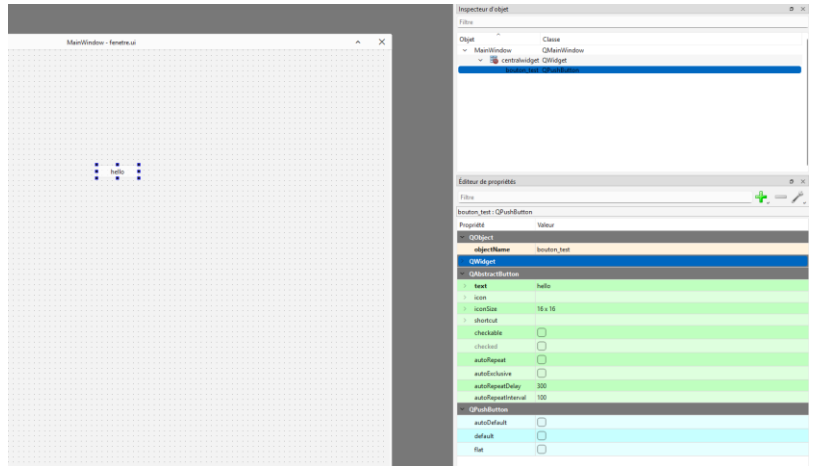
1. Dans l'environnement virtuel de votre projet, installer **PySide6** : `pip install PySide6`.
2. Dans VS Code, installer l'extension **PySide6 UI2Py** : elle convertit un fichier **.ui** en **.py** par un clic droit, sans ligne de commande (`pyside6-uic`).

2.2 Tutoriel : créer une première fenêtre

1. Ouvrir un terminal dans le dossier de votre projet.
2. Activer votre environnement virtuel.
3. Lancer Qt Widgets Designer avec la commande **pyside6-designer**.
4. Choisir le modèle (template) **Main Window**.



5. Dans l'inspecteur d'objets ou dans la zone de conception, sélectionner la **menubar** et la **statusbar**, puis les supprimer (touche Suppr) afin d'obtenir une fenêtre la plus vide possible. L'objet **centralwidget** ne peut pas être supprimé : c'est le conteneur dans lequel vous placerez l'ensemble de vos widgets.
6. Placer un **Push Button** dans la fenêtre (glisser-déposer depuis la boîte de widgets).
7. Dans l'éditeur de propriétés, identifier les classes dont hérite un objet **QPushButton** (les propriétés sont regroupées par classe).
8. Renommer l'objet en **bouton_test** (propriété `objectName`) et changer le texte du bouton en « hello » (propriété `text`).
9. Vérifier dans l'inspecteur d'objets que le nom a bien été modifié.
10. Sauvegarder votre interface sous le nom **fenetre.ui** dans le dossier **src** de votre projet, puis ouvrir ce projet dans VS Code.
11. Faire un clic droit sur le fichier **fenetre.ui** et lancer la conversion en Python proposée par l'extension installée précédemment. Le fichier **fenetre_ui.py** est alors créé.
12. Ouvrir le code généré et repérer l'attribut **bouton_test**, qui est un objet de classe **QPushButton**, ainsi que les méthodes appelées sur cet objet.
13. Créer un nouveau fichier **main.py** contenant le code ci-dessous, puis l'exécuter : un clic sur le bouton doit afficher « Hello ENSGTI! » dans le terminal.



```
main.py > ...
import sys

from PySide6.QtWidgets import QApplication, QMainWindow
from fenetre_ui import Ui_MainWindow

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        # Connexion du bouton_test à une fonction
        self.ui.bouton_test.clicked.connect(self.fonction_bouton_test)

    def fonction_bouton_test(self):
        print("Hello ENSGTI!")

app = QApplication(sys.argv) #Création et initialisation de l'application Qt

window = MainWindow() #Création de la fenêtre principale
window.show() #Affichage de la fenêtre principale

sys.exit(app.exec()) #Lancement de la boucle d'événements de l'application Qt et sortie propre à la fermeture de l'application
```

clicked est un **signal** fourni par le bouton (**QPushButton**). Ce signal est émis automatiquement par Qt lorsque l'utilisateur clique sur le bouton.

Le **slot** est la fonction (ou méthode) exécutée lorsque le signal est émis (**fonction_bouton_test** ici). La méthode **connect** relie le signal au slot.

Il existe une autre façon de connecter un slot à un signal : utiliser le décorateur **@Slot()** et nommer la méthode **on_<nom_objet>_<signal>** (ici **on_bouton_test_clicked**). Qt réalise alors la connexion automatiquement, et la ligne **connect** n'est plus nécessaire :

```
main.py > ...
import sys

from PySide6.QtWidgets import QApplication, QMainWindow
from fenetre_ui import Ui_MainWindow

from PySide6.QtCore import Slot

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        @Slot()
        def on_bouton_test_clicked(self): # Slot appelé automatiquement lors du clic sur "bouton_test"
            print("Hello ENSGTI!")

app = QApplication(sys.argv) #Création et initialisation de l'application Qt

window = MainWindow() #Création de la fenêtre principale
window.show() #Affichage de la fenêtre principale

sys.exit(app.exec()) #Lancement de la boucle d'événements de l'application Qt et sortie propre à la fermeture de l'application
```

- Ajouter la méthode suivante dans la classe **MainWindow** : elle est appelée automatiquement à la fermeture de la fenêtre et permet d'exécuter une action à ce moment-là (ici, afficher un message). L'instruction `event.accept()` autorise la fermeture, alors que `event.ignore()` l'annulerait.

```
def closeEvent(self, event):
    print("Fermeture de la fenêtre principale")
    event.accept() # Accepter l'événement de fermeture
```

2.3 Développement de l'interface et fin du projet

À partir des éléments précédents, créer une interface esthétique et simple d'utilisation qui permet de piloter les différentes fonctions réalisées dans la première partie de ce mini-projet. Les classes d'appareils (**Oscillo**, **GBF**, etc.) ne doivent contenir aucun code d'interface : seule la classe **MainWindow** communique avec elles.

À la fin du projet, suivre les étapes ci-dessous pour créer un exécutable Windows de votre application :

- Installer le paquet **pyinstaller** dans votre environnement virtuel (`pip install pyinstaller`).
- Exécuter la commande suivante depuis le dossier du projet (adapter le chemin `src/main.py` au fichier principal de votre application) :

```
pyinstaller --onefile --name application --collect-all pyvisa_py src/main.py
```

- Tester l'application générée, qui se trouve dans le dossier **dist/** du projet.

Résumé

Dans ce TP, vous avez appris à :

- Concevoir une interface graphique avec Qt Widgets Designer ;
- Relier les widgets de l'interface aux actions du programme grâce aux signaux et aux slots ;
- Séparer la couche interface de la couche métier (les classes d'appareils **Oscillo**, **GBF**, etc.) ;
- Piloter l'appareil réel depuis l'interface et informer l'utilisateur, de façon lisible, des erreurs et de l'état de la connexion ;
- Générer avec PyInstaller un exécutable Windows autonome, utilisable sur un poste sans environnement Python.