ADR-003: PyQt6 as Primary GUI Framework

- Architecture Diagrams
- Context and Problem Statement
- Decision Drivers
- Considered Options
- Decision Outcome
  - Positive Consequences
  - Negative Consequences
- Pros and Cons of the Options
  - PyQt6
  - PySide6
  - Tkinter
  - Kivy
  - wxPython
- Links
- Implementation Notes
- Review History

Architecture Diagrams

Status: Accepted **Date:** 2025-09-15 (estimated) **Deciders:** System Architect, Development Team **Technical Story:** Initial GUI framework selection for TOSCA v0.1

Context and Problem Statement

TOSCA requires a robust, modern GUI framework for medical device laser control. The framework must support: - Cross-platform deployment (Windows 10/11 primary target) - Real-time data visualization (camera feeds, graphs, status displays) - Complex widget layouts with dynamic updates - Thread-safe signal/slot architecture for hardware communication - Long-term commercial licensing compatibility

Which GUI framework should TOSCA use?

Decision Drivers

- Medical device compliance (21 CFR Part 11, IEC 62304)
- Real-time performance requirements (30 FPS camera, 100Hz data logging)
- Thread safety for hardware communication (serial, USB devices)
- Mature ecosystem with extensive documentation
- Commercial licensing compatibility (no LGPL restrictions for static linking)
- Active maintenance and Python 3.12+ support
- Built-in support for graphics, plotting, and video display

Considered Options

- PyQt6 (Qt 6.x with GPLv3/Commercial dual licensing)
- PySide6 (Qt 6.x with LGPL/Commercial dual licensing)
- Tkinter (Python standard library, limited features)
- Kivy (modern, mobile-focused, less mature for desktop)
- wxPython (native widgets, older architecture)

Decision Outcome

Chosen option: “PyQt6”, because it provides the most mature, performant solution with excellent real-time capabilities, comprehensive documentation, and proven track record in medical device applications. The GPLv3 license is acceptable for research-phase development, with commercial licensing option available for production deployment.

Positive Consequences

- **Signal/Slot Architecture:** Built-in thread-safe communication perfect for hardware controllers
- **Real-Time Performance:** Proven 30+ FPS camera streaming, smooth UI updates
- **Mature Ecosystem:** Extensive documentation, large community, abundant examples
- **pyqtgraph Integration:** Excellent real-time plotting library designed for PyQt
- **Model-View Architecture:** Clean separation of UI and business logic
- **Designer Tool:** Qt Designer for visual UI layout (`.ui` files)
- **Future-Proof:** Qt 6.x is modern, actively developed, long-term support

Negative Consequences

- **GPLv3 License:** Requires open-source release OR commercial Qt license ($5,000-15,000/year) for proprietary distribution
- **Learning Curve:** Qt framework is comprehensive but complex for new developers
- **Binary Size:** PyQt6 adds ~100 MB to deployment package
- **Platform-Specific Quirks:** Some minor differences between Windows/Linux (font rendering, file dialogs)

Pros and Cons of the Options

PyQt6

- Good, because signal/slot architecture is thread-safe by design (critical for hardware communication)
- Good, because proven real-time performance (medical imaging, oscilloscopes, DAQ systems)
- Good, because pyqtgraph provides production-ready real-time plotting
- Good, because Qt Designer allows rapid UI prototyping
- Good, because QPainter provides low-level graphics control (overlays, annotations)
- Bad, because GPLv3 requires open-source release or commercial license
- Bad, because learning curve is steep for Qt beginners
- Bad, because deployment size is larger than alternatives

PySide6

- Good, because LGPL license is more permissive (dynamic linking allowed)
- Good, because functionally equivalent to PyQt6 (same Qt 6.x backend)
- Good, because officially supported by Qt Company
- Bad, because smaller community than PyQt (fewer examples, less StackOverflow content)
- Bad, because pyqtgraph is optimized for PyQt, not PySide (compatibility issues)
- Bad, because slightly slower signal/slot performance in benchmarks

Tkinter

- Good, because part of Python standard library (no external dependencies)
- Good, because simple API, easy to learn
- Good, because cross-platform with native look-and-feel
- Bad, because limited real-time capabilities (not designed for video streaming)
- Bad, because no built-in threading support (manual queue management required)
- Bad, because limited plotting/graphing libraries (matplotlib is slow for real-time)
- Bad, because outdated aesthetics, difficult to customize

Kivy

- Good, because modern, GPU-accelerated rendering
- Good, because excellent for touch interfaces and mobile
- Bad, because primarily mobile-focused (desktop as secondary target)
- Bad, because smaller ecosystem for desktop applications
- Bad, because custom widget toolkit (non-native look-and-feel)
- Bad, because less mature for medical device applications

wxPython

- Good, because uses native OS widgets (platform-consistent UI)
- Good, because mature, stable, large community
- Bad, because older architecture (predates modern signal/slot patterns)
- Bad, because manual threading management (no built-in thread-safe communication)
- Bad, because limited real-time plotting options
- Bad, because slower development than Qt (smaller team)

Links

- PyQt6 Documentation
- Qt for Medical Devices
- Related: [ADR-010-threading-model.md] (thread safety with PyQt6 signals)

Implementation Notes

Migration Path from PyQt5 → PyQt6 (October 2025): - Changed import: `from PyQt5` → `from PyQt6` - Updated enum access: `Qt.AlignCenter` → `Qt.AlignmentFlag.AlignCenter` - Updated signal connections: `connect(self.slot)` (no change) - Verified camera streaming at 30 FPS stable - Verified thread-safe hardware controller signal emissions

Licensing Decision: - Research phase: GPLv3 acceptable (open-source project) - Phase 6+ (Pre-Clinical): Evaluate commercial Qt license if proprietary deployment required - Alternative: Consider PySide6 migration if LGPL dynamic linking sufficient

Thread Safety Pattern Established:

```python
from PyQt6.QtCore import QObject, pyqtSignal
import threading

class HardwareController(QObject):
    value_changed = pyqtSignal(float)

    def __init__(self):
        super().__init__()
        self._lock = threading.RLock()

    def set_value(self, value: float):
        with self._lock:
            self.hardware.set(value)
            self.value_changed.emit(value)  # Thread-safe signal
```

This pattern is used consistently across all hardware controllers.

Review History

Date	Reviewer	Notes
2025-09-15	System Architect	Initial PyQt6 selection
		Confirmed after 1 month of

2025-10-15	Development Team	development
2025-11-04	Documentation Review	Formalized as ADR-003

Template Version: 1.0 **Last Updated:** 2025-11-04