HardwareControllerBase Usage Guide

Architecture Diagrams

Last Updated: 2025-11-04

Overview

`HardwareControllerBase` is an abstract base class that enforces a consistent interface across all hardware controllers in the TOSCA project. It combines PyQt6's `QObject` (for signals/slots) with Python's `ABC` (for interface enforcement).

File Location

`src/hardware/hardware_controller_base.py`

Design Philosophy

1. **Minimal Interface** - Only enforces what's truly common across all hardware
2. **PyQt6 Integration** - Seamless signal/slot support for thread-safe communication
3. **Type Safety** - Comprehensive type hints for IDE support and static analysis
4. **Optional Event Logging** - Integration with TOSCA's event logging system

Required Interface

All hardware controllers MUST implement:

1. `connect(**kwargs) -> bool` - Connect to hardware device
2. `disconnect() -> None` - Disconnect and cleanup resources
3. `get_status() -> dict[str, Any]` - Get current hardware status

All hardware controllers MUST have:

1. `is_connected: bool` attribute - Current connection state
2. `connection_changed` signal - Emitted when connection state changes
3. `error_occurred` signal - Emitted when errors occur

Example: Creating a New Hardware Controller

```python
from typing import Any
from src.hardware.hardware_controller_base import HardwareControllerBase


class MyHardwareController(HardwareControllerBase):
    """Example hardware controller implementation."""

    def __init__(self, event_logger=None):
        super().__init__(event_logger)
        self.device = None

    def connect(self, port: str = "COM3", **kwargs) -> bool:
        """
        Connect to hardware device.

        Args:
            port: Serial port for device
            **kwargs: Additional device-specific parameters

        Returns:
            True if connected successfully
        """
        try:
            # Connect to hardware
            self.device = SomeHardwareAPI(port)

            # Update state
            self.is_connected = True

            # Emit signal
            self.connection_changed.emit(True)

            # Log event (optional)
            if self.event_logger:
                from src.core.event_logger import EventType
                self.event_logger.log_hardware_event(
                    event_type=EventType.HARDWARE_CONNECT,
                    description=f"Device connected on {port}",
                    device_name="My Hardware Device"
                )

            return True

        except Exception as e:
            # Emit error signal
            self.error_occurred.emit(f"Connection failed: {e}")
            return False

    def disconnect(self) -> None:
        """Disconnect from hardware device."""
        # Stop any operations
        self.stop_operations()

        # Close device
        if self.device:
            try:
                self.device.close()
            except Exception as e:
                # Don't raise - log but continue cleanup
                logger.warning(f"Cleanup error: {e}")
            self.device = None

        # Update state
        self.is_connected = False
        self.connection_changed.emit(False)

        # Log event (optional)
        if self.event_logger:
            from src.core.event_logger import EventType
            self.event_logger.log_hardware_event(
                event_type=EventType.HARDWARE_DISCONNECT,
                description="Device disconnected",
                device_name="My Hardware Device"
            )
```

```
    def get_status(self) -> dict[str, Any]:
        """
        Get current hardware status.

        Returns:
            Dictionary with status information (must include 'connected' key)
        """
        if not self.is_connected or not self.device:
            return {"connected": False}

        return {
            "connected": True,
            "device_id": self.device.get_id(),
            "temperature": self.device.get_temperature(),
            "status": "ready"
        }
```

Metaclass Conflict Resolution

The base class uses a combined metaclass (`QObjectABCMeta`) to resolve the conflict between PyQt6's metaclass and Python's ABC metaclass:

```
class QObjectABCMeta(type(QObject), type(ABC)):
    """Combines QObject and ABC metaclasses."""
    pass

class HardwareControllerBase(QObject, ABC, metaclass=QObjectABCMeta):
    # ... implementation ...
```

This allows the class to: - Support PyQt6 signals/slots (from `QObject`) - Enforce abstract methods (from `ABC`)

Signal/Slot Integration

All controllers automatically get two signals:

```
# Emitted when connection state changes
connection_changed = pyqtSignal(bool)  # True=connected, False=disconnected

# Emitted when errors occur
error_occurred = pyqtSignal(str)  # Error message
```

Usage Example:

```
# Create controller
controller = MyHardwareController()

# Connect signals to slots
controller.connection_changed.connect(on_connection_changed)
controller.error_occurred.connect(on_error_occurred)

# Connect to device (signals will be emitted)
controller.connect(port="COM3")
```

Compatibility with Existing Controllers

All existing TOSCA hardware controllers already implement the required interface:

- `CameraController` ✓
- `ActuatorController` ✓
- `LaserController` ✓
- `GPIOController` ✓

They can be updated to inherit from `HardwareControllerBase` in a future refactor.

Benefits

1. **Interface Enforcement** - Python will raise `TypeError` if abstract methods not implemented
2. **Consistency** - All hardware controllers follow the same pattern
3. **Type Safety** - IDEs and static analyzers can verify usage
4. **Documentation** - Clear contract for what controllers must provide
5. **Refactoring Safety** - Changes to base class affect all controllers
6. **Testing** - Easy to create mock controllers for unit tests

Testing

Example unit test using the base class:

```
from src.hardware.hardware_controller_base import HardwareControllerBase

class MockController(HardwareControllerBase):
    def connect(self, **kwargs):
        self.is_connected = True
        self.connection_changed.emit(True)
        return True

    def disconnect(self):
        self.is_connected = False
        self.connection_changed.emit(False)

    def get_status(self):
        return {"connected": self.is_connected}

def test_controller():
    controller = MockController()
    assert controller.connect() is True
    assert controller.is_connected is True
    assert controller.get_status()["connected"] is True
```

Future Enhancements

Potential additions to the base class:

1. Optional `is_hardware_available()` method (already implemented)
2. Standard error handling decorator
3. Connection timeout management
4. Automatic reconnection logic
5. Health check interface

See Also

- `src/hardware/camera_controller.py` - Example implementation
- `src/hardware/actuator_controller.py` - Example implementation
- `src/hardware/laser_controller.py` - Example implementation
- `src/hardware/gpio_controller.py` - Example implementation