Qt for Embedded Systems - Master Exam

Embedded Qt Academy

Core UI Classes (QWidget Framework)

1. **Basic Widget Creation** Create a Qt application that:
   - Uses `QMainWindow` as base
   - Contains a `QTabWidget` with 3 tabs
   - Each tab should have:
     - Tab 1: `QPushButton` and `QLabel`
     - Tab 2: `QSlider` connected to a `QProgressBar`
     - Tab 3: `QGraphicsView` displaying a simple rectangle
2. **Dynamic UI Generation** Write code that:
   - Parses this XML using `QDomDocument`:

```
<UI>
  <Button text="Start" geometry="10,10,100,30"/>
  <CheckBox text="Enable Logging" checked="true"/>
</UI>
```

   - Dynamically creates the widgets described in the XML
   - Places them in a `QVBoxLayout`

File and Process Management

3. **QFile Operations** Create a logger class that:
   - Uses `QFile` to write timestamped messages
   - Implements log rotation when file exceeds 1MB
   - Uses `QTextStream` for UTF-8 formatted output

4. **QProcess Implementation** Write a process manager that:
    - Uses `QProcess` to execute system commands
    - Captures both stdout and stderr separately
    - Implements a timeout (kill process after 30 seconds)
    - Displays output in a `QPlainTextEdit`

Hardware Communication

5. **Serial Port Monitor** Create an application that:
    - Uses `QSerialPort` to list available ports
    - Connects to a selected port at 115200 baud
    - Displays incoming data in hex and ASCII formats
    - Implements basic flow control (RTS/CTS)
6. **CAN Bus Analyzer** Implement a CAN monitor using:
    - `QCanBusDevice` (with SocketCAN backend)
    - `QCanBusFrame` for message processing
    - A `QTableView` to display messages with timestamp, ID, and payload
    - Statistics using `QLCDNumber` widgets

Data Management

7. **XML Configuration Parser** Create a system that:
    - Uses `QDomDocument` to parse:

```
<ECU>
  <Parameter name="max_rpm" value="8000"/>
  <Parameter name="temp_warn" value="90"/>
</ECU>
```

    - Validates against XSD schema using `QXmlSchemaValidator`
    - Converts parameters to `QVariantMap` for easy access
8. **Database Integration** Implement a SQLite interface that:

- Uses `QSqlDatabase` for connection
- Creates tables from XML definition
- Provides CRUD operations through `QSqlQuery`
- Shows data in `QTableView` with `QSqlTableModel`

Advanced Topics

9. **Multithreaded Worker** Create a system with:
   - Worker class inheriting from `QObject`
   - Controller using `QThread` for execution
   - `QMutex` protected data access
   - `QWaitCondition` for task synchronization
   - Progress reporting via signals/slots
10. **Plugin Architecture** Design a plugin system where:
    - Core application uses `QPluginLoader`
    - Plugins implement a defined interface
    - Each plugin can register its own `QWidget`
    - Configuration handled via `QSettings`
11. **Deployment Package** Create a build system that:
    - Uses `QMake` or `CMake`
    - Bundles dependencies with `windeployqt` or equivalent
    - Generates firmware image using `dd` and `QProcess`
    - Signs packages with `QProcess` and OpenSSL

Real-world Challenges

12. **Low-Memory Optimization** Optimize an existing Qt application to:
    - Reduce RAM usage below 50MB
    - Implement lazy loading with `QStackedWidget`
    - Use `QScopedPointer` for dynamic widgets

- Profile with `QMemoryInfo`

13. **Crash Recovery System** Implement a watchdog that:
    - Uses `QSharedMemory` for state tracking
    - Employs `QProcess` to restart on failure
    - Logs crashes via `QFile`
    - Recovers unsaved data from `QTemporaryFile`
14. **Secure Communication** Build a TLS secured channel using:
    - `QSslSocket` for encrypted connections
    - Certificate management with `QSslCertificate`
    - Key storage in `QFile` with `QSaveFile` atomic writes
    - Verification using `QSslConfiguration`

Bonus: Embedded-specific Problems

15. **Framebuffer Display** Create a direct framebuffer application that:
    - Uses `QGuiApplication` (no X11/Wayland)
    - Implements touch input via `QEvdevTouchHandler`
    - Rotates display with `QScreen` properties
    - Adjusts DPI settings programmatically
16. **Power Management** Implement a power-aware system with:
    - `QBatteryInfo` monitoring
    - `QTimer` for sleep mode activation
    - Wake-on-LAN using `QUdpSocket`
    - Low-power state detection via `QProcess` and sysfs