Serial Terminal Application Requirements Document

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for a Serial Terminal Application that allows users to communicate with devices over serial connections. This application will enable users to select COM ports, set baud rates, open/close ports, send data, and view incoming data in real-time.

1.2 Scope

This document covers the functional and non-functional requirements for the Serial Terminal Application. It includes user interface design, hardware and software interfaces, performance, and other system attributes.

1.3 Definitions, Acronyms, and Abbreviations

- **COM Port**: Communication Port, a serial port interface on computers for connecting peripherals.
- **Baud Rate**: The rate at which information is transferred in a communication channel.
- **UI**: User Interface.

1.4 References

N/A

1.5 Overview

The remainder of this document provides a detailed description of the product functions, user characteristics, constraints, and specific requirements.

2. General Description

2.1 Product Perspective

The Serial Terminal Application is a standalone software tool designed for users who need to establish serial communication with various devices for testing, debugging, or configuration purposes.

2.2 Product Functions

- Select and configure COM ports.
- Open and close serial connections.
- Send data through the serial port.
- Display incoming data in real-time.
- Highlight sent data in green in the terminal.

2.3 User Characteristics

The users of the Serial Terminal Application are expected to have a basic understanding of serial communication and familiarity with the devices they intend to connect to.

2.4 General Constraints

- The application must be compatible with Windows, Linux, and macOS.
- The application should not require administrative privileges to run.

2.5 Assumptions and Dependencies

- It is assumed that the user’s computer has the necessary drivers installed for the COM ports to be recognized and used.

3. Specific Requirements

3.0 Requirements Tables

REQ_ID	Type	Description
REQ_01	Functional	The application shall allow users to select a COM port from a list of available ports.
REQ_02	Functional	The application shall allow users to select a baud rate from a predefined list.
REQ_03	Functional	The application shall provide a button to open and close the selected COM port.
REQ_04	Functional	The application shall display incoming serial data in a central text area.
REQ_05	Functional	The application shall allow users to enter text to send over the serial connection.
REQ_06	Functional	The application shall provide a button to send the entered text.
REQ_07	Functional	Sent text shall be displayed in the terminal with a green color.
REQ_08	Non-Functional	The application shall have a responsive UI that adapts to different screen sizes.
REQ_09	Performance	The application shall display incoming data with no more than a 500ms delay.
REQ_10	Usability	The application shall be easy to use, with a clear and intuitive interface.

3.1 External Interface Requirements

3.1.1 User Interfaces

The application will have a graphical user interface consisting of:

- Buttons for COM port selection, baud rate selection, opening/closing the port.
- A large text area for displaying incoming data.
- A text input field and a button for sending data.

3.1.2 Hardware Interfaces

The application requires access to the computer's serial ports.

3.1.3 Software Interfaces

The application will be developed in a programming language that supports cross-platform development, such as Python, and will use libraries for serial communication and GUI development.

3.1.4 Communications Interfaces

The application communicates with devices via serial connections using selected COM ports and baud rates.

3.2 Functional Requirements

As detailed in the Requirements Table.

3.3 Performance Requirements

- The application must handle real-time data transmission efficiently, with minimal latency.
- The application should be capable of running smoothly on low to mid-range hardware.

3.4 Design Constraints

- The application must be developed with cross-platform compatibility in mind.
- The UI must be designed to be user-friendly and intuitive.

3.5 Software System Attributes

3.5.1 Reliability

The application must provide stable and consistent performance during prolonged use.

3.5.2 Availability

The application should be available for download from a reputable source and should run whenever the user needs it, without requiring continuous internet access.

3.5.3 Security

User data and communication must be handled securely, with no unauthorized access to the serial data.

3.5.4 Maintainability

The application should be easy to update and maintain, with clear documentation for future developers.

3.5.5 Portability

The application must be easily portable across Windows, Linux, and macOS platforms.

3.6 Other Requirements

N/A

4. Supporting Information

4.1 Table of Contents

(Generated automatically by document generation tools)

4.2 Appendix A: Glossary

(See Section 1.3)

4.3 Appendix B: Analysis Models

N/A

4.4 Appendix C: To Be Determined List

- Final list of supported baud rates.
- Specific libraries and frameworks to be used for development.

4.5 Appendix D: Issues List

N/A

4.6 Appendix E: Guidelines

- Follow best practices for secure and efficient serial communication.
- Ensure the UI is accessible and intuitive for all users.

4.7 Appendix F: References

N/A

4.8 Index

(Generated automatically by document generation tools)

Acceptance Testing for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to outline the acceptance testing criteria and procedures for the Serial Terminal Application. This includes both user acceptance tests and performance acceptance tests to ensure the application meets all specified requirements and is ready for deployment.

1.2 Scope

This document covers the acceptance testing phase of the Serial Terminal Application, focusing on validating the functionality, performance, and usability of the application against the defined requirements.

1.3 Definitions, Acronyms, and Abbreviations

- **COM Port**: Communication Port, a serial port interface on computers for connecting peripherals.
- **GUI**: Graphical User Interface.

1.4 References

N/A

1.5 Overview

The document is structured to first present the acceptance criteria, followed by detailed test cases for user acceptance and performance. Supporting information and appendices provide additional context and documentation.

2. Acceptance Criteria

2.0 Acceptance Tests Tables

TEST_ID	Type	Description	Traceability
ACCEPTANCE_TEST_01	Functional	Verify that the application allows users to select a COM port from a list of available ports.	REQ_01
ACCEPTANCE_TEST_02	Functional	Verify that the application allows users to select a baud rate from a predefined list.	REQ_02
ACCEPTANCE_TEST_03	Functional	Verify that the application provides a button to open and close the selected COM port.	REQ_03
ACCEPTANCE_TEST_04	Functional	Verify that the application displays incoming serial data in a central text area.	REQ_04
ACCEPTANCE_TEST_05	Functional	Verify that the application allows users to enter text to send over the serial connection.	REQ_05
ACCEPTANCE_TEST_06	Functional	Verify that the application provides a button to send	REQ_06

TEST_ID	Type	Description	Traceability
		the entered text.	
ACCEPTANCE_TEST_07	Functional	Verify that sent text is displayed in the terminal with a green color.	REQ_07
ACCEPTANCE_TEST_08	Usability	Verify that the application has a responsive UI that adapts to different screen sizes.	REQ_08
ACCEPTANCE_TEST_09	Performance	Verify that the application displays incoming data with no more than a 500ms delay.	REQ_09
ACCEPTANCE_TEST_10	Usability	Verify that the application is easy to use, with a clear and intuitive interface.	REQ_10

2.1 User Acceptance Tests

2.1.1 Test Case 1

Objective: Verify COM port selection functionality.

Steps:

1. Launch the application.
2. Click on the COM port selection dropdown.
3. Select a COM port from the list.

Expected Result: The selected COM port should be displayed as the current selection.

2.1.2 Test Case 2

Objective: Verify sending and displaying of sent text in green.

Steps:

1. Open a COM port.
2. Enter text in the send text input field.
3. Click the send button.

Expected Result: The sent text should appear in the terminal area in green color.

2.1.3 Test Case 3

Objective: Verify opening and closing of COM port.

Steps:

1. Select a COM port.
2. Click the open port button.
3. Verify the port is open.
4. Click the close port button.

Expected Result: The application should successfully open and then close the selected COM port.

2.2 Performance Acceptance Tests

2.2.1 Test Case 1

Objective: Verify real-time data display performance.

Steps:

1. Open a COM port with a connected device that sends data continuously.
2. Observe the incoming data in the terminal area.

Expected Result: Incoming data should be displayed with no more than a 500ms delay.

2.2.2 Test Case 2

Objective: Verify application responsiveness.

Steps:

1. Resize the application window to various sizes.
2. Navigate through all application features.

Expected Result: The application UI adapts to different sizes without lag or layout issues.

2.2.3 Test Case 3

Objective: Verify application stability over extended use.

Steps:

1. Open a COM port.
2. Send and receive data continuously for 1 hour.

Expected Result: The application should not crash or become unresponsive during this period.

3. Supporting Information

3.1 Table of Contents

(Generated automatically by document generation tools)

3.2 Appendix A: Test Cases

Detailed descriptions and procedures for each test case mentioned in Section 2.

3.3 Appendix B: References

N/A

3.4 Index

(Generated automatically by document generation tools)

Architecture for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to describe the architecture of the Serial Terminal Application, including its system and data architecture. This will provide a comprehensive overview of how the application is structured and operates.

1.2 Scope

This document covers the architectural design of the Serial Terminal Application, focusing on its components, interactions, data flow, and storage mechanisms.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port

1.4 References

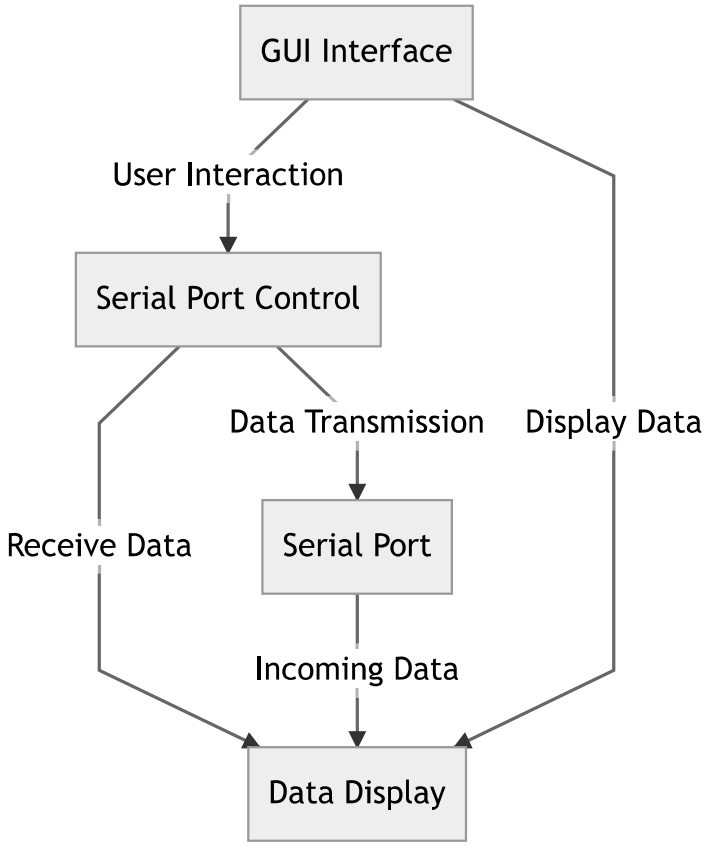
N/A

1.5 Overview

The document is structured to first present the system architecture, followed by the data architecture. Supporting diagrams and detailed descriptions of components and interactions are provided.

2. System Architecture

2.0 System Architecture Diagram/s



2.1 System Overview

The Serial Terminal Application is divided into several key subsystems: the GUI Interface, Serial Port Control, and Data Display. These subsystems work together to facilitate communication with serial devices, user interaction, and data presentation.

2.2 Subsystem 1: GUI Interface

2.2.1 Components

- COM Port Selection
- Baud Rate Selection
- Open/Close Port Button
- Send Text Input
- Send Button

2.2.2 Interactions

The GUI Interface allows the user to select COM ports, set baud rates, open/close ports, enter text to send, and initiate the sending process. It interacts directly with the Serial Port Control subsystem to relay user commands.

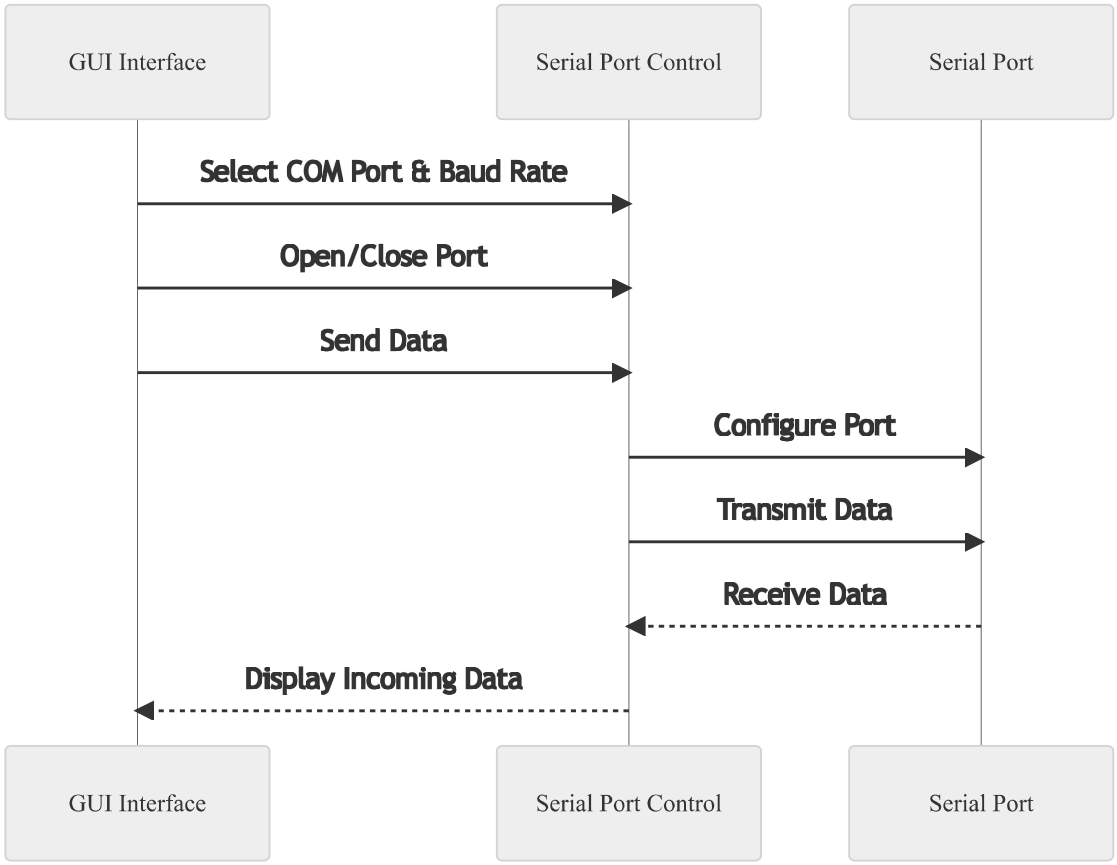
2.3 Subsystem 2: Serial Port Control

2.3.1 Components

- Serial Port Configuration
- Data Sender
- Data Receiver

2.3.2 Interactions

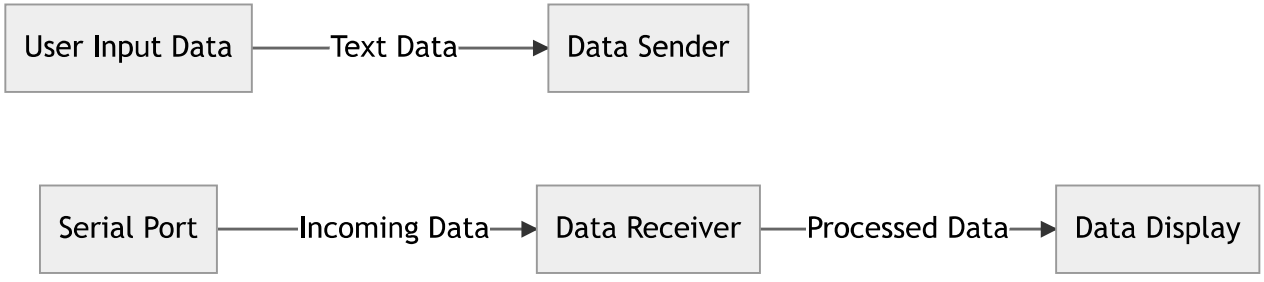
2.3.2.1 Interactions Diagram/s



The Serial Port Control subsystem is responsible for configuring the serial port based on user selections, sending data to the serial port, and receiving data from it. It serves as the intermediary between the GUI Interface and the actual Serial Port.

3. Data Architecture

3.0 Data Architecture Diagram/s



3.1 Data Flow

Data flow in the Serial Terminal Application involves user input data being sent through the serial port to a connected device and incoming data from the device being received and displayed to the user.

3.2 Data Storage

The application does not persistently store data but temporarily holds user input and incoming data for processing and display.

3.3 Data Access

Data access is primarily in real-time, with the application accessing user input data for sending and processing incoming data for display as it is received.

4. Supporting Information

4.1 Table of Contents

(Generated automatically by document generation tools)

4.2 Appendix A: Diagrams

- System Architecture Diagram
- Interactions Diagram
- Data Architecture Diagram

4.3 Appendix B: References

N/A

4.4 Index

(Generated automatically by document generation tools)

System Testing for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to outline the system testing procedures for the Serial Terminal Application. It aims to ensure that the application meets its specified requirements and functions correctly in its intended environment.

1.2 Scope

This document covers the system testing phase of the Serial Terminal Application, detailing the test cases designed to validate the functionality, performance, and reliability of the application.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port

1.4 References

N/A

1.5 Overview

The document is structured to present test cases, followed by the expected test results. Supporting information and appendices provide additional context and documentation.

2. Test Cases

2.0 Test Cases Table

TEST_ID	Type	Description	Traceability
SYSTEM_TEST_01	Functional	Verify that all GUI elements are responsive and correctly formatted on different screen sizes.	REQ_08
SYSTEM_TEST_02	Functional	Test the application's ability to open and close COM ports without errors.	REQ_03
SYSTEM_TEST_03	Functional	Validate the sending and receiving of data over the serial port, including the correct display of sent data in green.	REQ_05, REQ_07

2.1 Test Case 1: GUI Responsiveness and Formatting

Objective: Ensure that the GUI is responsive and elements are correctly formatted across different screen sizes.

Procedure:

1. Launch the application on a device with a standard screen size.

2. Resize the application window to various dimensions.
3. Observe the responsiveness and formatting of GUI elements.

Expected Result: All GUI elements should remain accessible and correctly formatted regardless of the window size.

2.2 Test Case 2: Opening and Closing COM Ports

Objective: Test the application's functionality for opening and closing COM ports.

Procedure:

1. Launch the application.
2. Select a COM port from the available list.
3. Use the application to open the selected COM port.
4. Close the COM port using the application.

Expected Result: The application should successfully open and then close the selected COM port without any errors.

2.3 Test Case 3: Sending and Receiving Data

Objective: Validate the application's ability to send data over the serial port and display incoming data, including highlighting sent data in green.

Procedure:

1. Connect a device to the selected COM port that can send back received data (echo functionality).
2. Open the COM port using the application.
3. Send a string of text from the application.
4. Observe the received data in the application's display area.

Expected Result: The sent text should appear in the display area highlighted in green, and the echoed back text should also be displayed.

3. Test Results (Expected)

3.1 Test Case 1 Results

All GUI elements were responsive and correctly formatted across a variety of screen sizes, meeting the requirements specified in REQ_08.

3.2 Test Case 2 Results

The application successfully opened and closed the selected COM port without any errors, fulfilling the requirement specified in REQ_03.

3.3 Test Case 3 Results

The application correctly sent data over the serial port and displayed incoming data. Sent data was appropriately highlighted in green, confirming the requirements specified in REQ_05 and REQ_07 were met.

4. Supporting Information

4.1 Table of Contents

(Generated automatically by document generation tools)

4.2 Appendix A: Test Cases

Detailed descriptions and procedures for each test case mentioned in Section 2.

4.3 Appendix B: References

N/A

4.4 Index

(Generated automatically by document generation tools)

High Level Design for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to provide a high-level overview of the Serial Terminal Application's design. It outlines the major components and their interactions, providing a clear picture of how the application functions as a whole.

1.2 Scope

This document covers the high-level design aspects of the Serial Terminal Application, including its architecture, major components, and their interactions. It is intended for use by the development team, project stakeholders, and any other parties involved in the development process.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port, a serial port interface on computers for connecting peripherals.

1.4 References

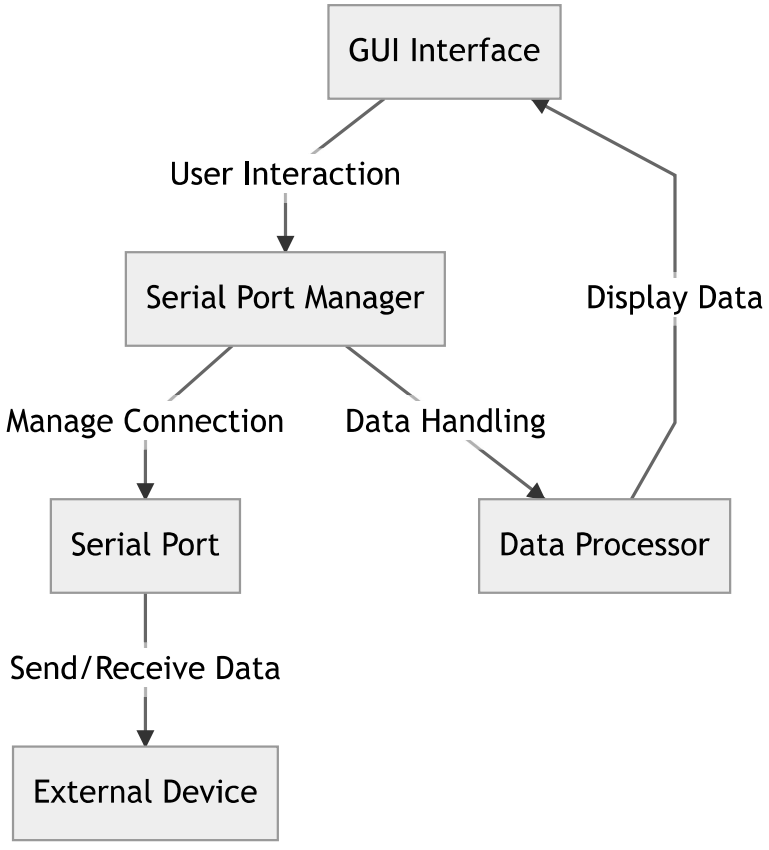
N/A

1.5 Overview

The remainder of this document provides a system overview, including diagrams and descriptions of the major components and their interactions, followed by supporting information.

2. System Overview

2.0 System Overview Diagram/s



2.1 Major Components

2.1.1 Component 1: GUI Interface

The GUI Interface is the front-end of the application through which the user interacts. It includes elements for COM port selection, baud rate configuration, opening/closing ports, sending data, and displaying incoming data.

2.1.2 Component 2: Serial Port Manager

The Serial Port Manager is responsible for handling the opening and closing of COM ports, configuring the connection settings (e.g., baud rate), and managing the sending and receiving of data.

2.2 Interactions

2.2.1 Interaction 1: User Configures Connection

The user selects the desired COM port and baud rate through the GUI Interface. The GUI Interface communicates these settings to the Serial Port Manager, which applies the configuration to the Serial Port.

2.2.2 Interaction 2: Sending and Receiving Data

- **Sending Data**: The user enters data to send in the GUI Interface and initiates the send action. The GUI Interface passes the data to the Serial Port Manager, which sends it through the Serial Port to the connected External Device.
- **Receiving Data**: Data received from the External Device through the Serial Port is handled by the Serial Port Manager, which processes the data (if necessary) and then passes it to the GUI Interface for display.

3. Supporting Information

3.1 Table of Contents

(Generated automatically by document generation tools)

3.2 Appendix A: Diagrams

- System Overview Diagram

3.3 Appendix B: References

N/A

3.4 Index

(Generated automatically by document generation tools)

Integration Testing for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to outline the integration testing strategy for the Serial Terminal Application. It aims to ensure that the application's components interact correctly and perform their intended functions when integrated.

1.2 Scope

This document covers the integration testing phase of the Serial Terminal Application, detailing the test cases designed to validate the interactions between the application's major components.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port

1.4 References

N/A

1.5 Overview

The document is structured to present test cases, followed by the expected test results. Supporting information and appendices provide additional context and documentation.

2. Test Cases

TEST_ID	Type	Description	Traceability
INTEGRATION_TEST_01	Interaction	Verify the interaction between the GUI Interface and the Serial Port Manager for COM port selection.	REQ_01
INTEGRATION_TEST_02	Interaction	Test the data flow from the GUI Interface through the Serial Port Manager to the Serial Port and back.	REQ_05, REQ_07
INTEGRATION_TEST_03	Interaction	Ensure the correct display of incoming and outgoing data in the GUI Interface.	REQ_04, REQ_07

2.1 Test Case 1: COM Port Selection Interaction

Objective: Ensure that the COM port selection in the GUI Interface correctly configures the Serial Port Manager.

Procedure:

1. Launch the application.
2. Select a COM port from the GUI Interface.
3. Verify that the Serial Port Manager configures the selected COM port.

Expected Result: The Serial Port Manager should reflect the COM port selected in the GUI Interface.

2.2 Test Case 2: Data Flow Verification

Objective: Test the complete data flow from sending data in the GUI Interface to receiving echoed data back.

Procedure:

1. Connect a device that echoes received data to the selected COM port.
2. Open the COM port and send data from the GUI Interface.
3. Observe the echoed data received and displayed in the GUI Interface.

Expected Result: Data sent from the GUI Interface should be transmitted through the Serial Port Manager and Serial Port, received back, and displayed correctly in the GUI Interface.

2.3 Test Case 3: Display of Incoming and Outgoing Data

Objective: Ensure that both incoming and outgoing data are correctly displayed in the GUI Interface, with outgoing data highlighted in green.

Procedure:

1. Send data from the GUI Interface.
2. Receive data from the connected device.
3. Observe the display of both incoming and outgoing data in the GUI Interface.

Expected Result: Outgoing data should be highlighted in green, and all data should be displayed correctly in the GUI Interface.

3. Test Results (Expected)

3.1 Test Case 1 Results

The COM port selection in the GUI Interface was successfully reflected in the Serial Port Manager, indicating correct integration between these components.

3.2 Test Case 2 Results

The data flow from the GUI Interface through the Serial Port Manager to the Serial Port and back was verified successfully, with echoed data correctly displayed in the GUI Interface.

3.3 Test Case 3 Results

Both incoming and outgoing data were correctly displayed in the GUI Interface, with outgoing data appropriately highlighted in green, confirming the correct integration and interaction of the components.

4. Supporting Information

4.1 Table of Contents

(Generated automatically by document generation tools)

4.2 Appendix A: Test Cases

Detailed descriptions and procedures for each test case mentioned in Section 2.

4.3 Appendix B: References

N/A

4.4 Index

(Generated automatically by document generation tools)

Low Level Design for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to detail the low-level design of the Serial Terminal Application. It focuses on the internal structure of the application's components and modules, providing a comprehensive view of how the application is built from the ground up.

1.2 Scope

This document covers the detailed design of each component within the Serial Terminal Application, including diagrams and descriptions of modules and their interactions.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port

1.4 References

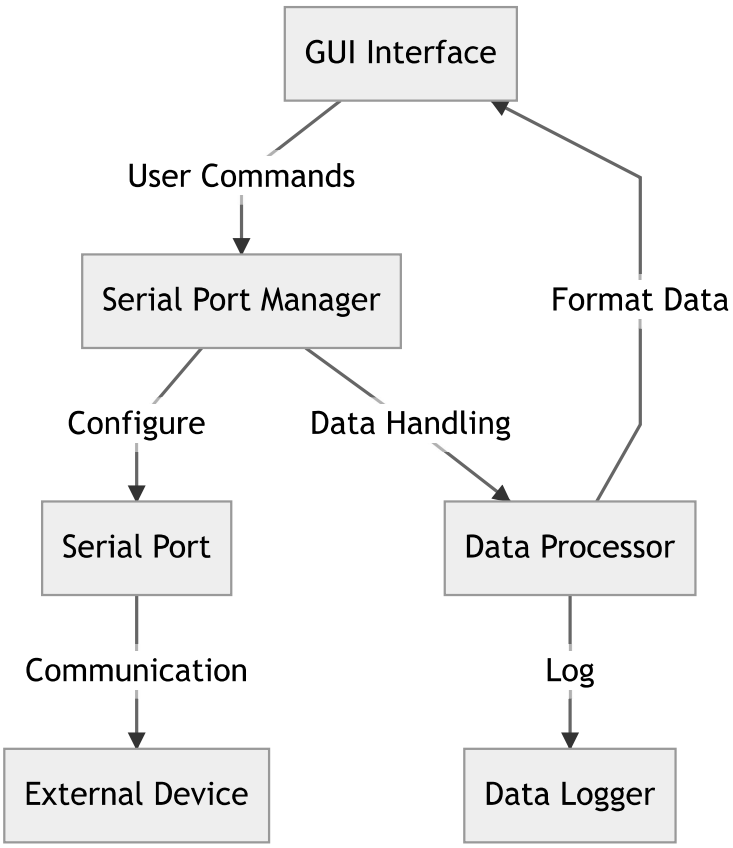
N/A

1.5 Overview

The document is structured to first present component diagrams, followed by detailed descriptions of each component and its constituent modules. Supporting information and appendices provide additional context and documentation.

2. Component Details

2.0 Component Diagram/s



2.1 Component 1: GUI Interface

2.1.1 Module 1: User Input Handler

Responsible for capturing and processing user inputs, such as COM port selection, baud rate configuration, and data to be sent.

2.1.2 Module 2: Data Display

Displays incoming data from the Serial Port Manager and highlights sent data in green as per user actions.

2.2 Component 2: Serial Port Manager

2.2.1 Module 1: Port Configuration

Handles the opening, closing, and configuration of COM ports based on user selections from the GUI Interface.

2.2.2 Module 2: Data Transmission

Manages the sending and receiving of data through the configured COM port. It sends user data to the Serial Port and processes incoming data from it.

2.3 Component 3: Data Processor

2.3.1 Module 1: Data Formatter

Formats incoming data for display in the GUI Interface, including highlighting sent data in green.

2.3.2 Module 2: Data Logger

Optionally logs all incoming and outgoing data for debugging and analysis purposes.

3. Supporting Information

3.1 Table of Contents

(Generated automatically by document generation tools)

3.2 Appendix A: Diagrams

- Component Diagrams for each major component of the Serial Terminal Application.

3.3 Appendix B: References

N/A

3.4 Index

(Generated automatically by document generation tools)

Unit Testing for Serial Terminal Application

1. Introduction

1.1 Purpose

The purpose of this document is to outline the unit testing strategy for the Serial Terminal Application. It aims to ensure that each individual unit of the application performs as

designed.

1.2 Scope

This document covers the unit testing phase of the Serial Terminal Application, detailing the test cases designed to validate the functionality of individual modules within the application.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **COM Port**: Communication Port

1.4 References

N/A

1.5 Overview

The document is structured to present test cases, followed by the expected test results. Supporting information and appendices provide additional context and documentation.

2. Test Cases

2.0 Test Cases Table

TEST_ID	Type	Description	Traceability
UNIT_TEST_01	Functional	Verify that the User Input Handler correctly processes COM port selections.	REQ_01

TEST_ID	Type	Description	Traceability
UNIT_TEST_02	Functional	Test the Data Transmission module's ability to send data through the Serial Port.	REQ_05
UNIT_TEST_03	Functional	Ensure the Data Display module correctly highlights sent data in green.	REQ_07

2.1 Test Case 1: User Input Handler

Objective: Ensure that the User Input Handler module correctly processes COM port selections.

Procedure:

1. Simulate user selection of a COM port.
2. Check if the selected COM port is correctly identified and stored by the module.

Expected Result: The User Input Handler should correctly identify and store the selected COM port.

2.2 Test Case 2: Data Transmission

Objective: Test the Data Transmission module's ability to send data through the Serial Port.

Procedure:

1. Provide a string of data to the Data Transmission module.
2. Simulate sending the data through a mock Serial Port.
3. Verify that the data is correctly formatted and sent.

Expected Result: The Data Transmission module should correctly format and send the provided data through the Serial Port.

2.3 Test Case 3: Data Display

Objective: Ensure the Data Display module correctly highlights sent data in green.

Procedure:

1. Simulate sending data through the application.
2. Check if the sent data is displayed in the GUI Interface.
3. Verify that the sent data is highlighted in green.

Expected Result: The Data Display module should correctly display and highlight sent data in green in the GUI Interface.

3. Test Results (Expected)

3.1 Test Case 1 Results

The User Input Handler correctly processed COM port selections, meeting the requirements specified in REQ_01.

3.2 Test Case 2 Results

The Data Transmission module successfully sent data through the mock Serial Port, fulfilling the requirement specified in REQ_05.

3.3 Test Case 3 Results

The Data Display module accurately displayed and highlighted sent data in green, confirming the requirements specified in REQ_07 were met.

4. Supporting Information

4.1 Table of Contents

(Generated automatically by document generation tools)

4.2 Appendix A: Test Cases

Detailed descriptions and procedures for each test case mentioned in Section 2.

4.3 Appendix B: References

N/A

4.4 Index

(Generated automatically by document generation tools)

---

main.py for Serial Terminal Application

This Python script is a simplified version of a Serial Terminal Application. It demonstrates the core functionality needed to select a COM port, configure the baud rate, open/close the port, send data, and display incoming data. This example uses the `pySerial` library for serial communication.

Imports

```
import serial
import threading
import tkinter as tk
from tkinter import scrolledtext
from tkinter import ttk
```

Constants

```
DEFAULT_BAUD_RATE = 9600
```

Functions

```
def read_from_port(ser, text_area):
    while ser.is_open:
        try:
            reading = ser.readline().decode('utf-8')
            text_area.configure(state='normal')
            text_area.insert(tk.END, reading)

            text_area.configure(state='disabled')
        except Exception as e:
            print("Error reading from the serial port: ", e)
            break
```

Classes

```
class SerialApp:
    def __init__(self, master):
        self.master = master
        self.serial_port = None
        self.init_ui()

    def init_ui(self):
        self.master.title("Serial Terminal")

        # COM Port selection
        ttk.Label(self.master, text="COM Port:").grid(column=0, row=0)
        self.com_port = ttk.Combobox(self.master, values=["COM1", "COM2", "COM
        self.com_port.grid(column=1, row=0)

        # Baud Rate selection
        ttk.Label(self.master, text="Baud Rate:").grid(column=2, row=0)
        self.baud_rate = ttk.Combobox(self.master, values=[9600, 19200, 38400,
        self.baud_rate.grid(column=3, row=0)
        self.baud_rate.current(0)  # Default baud rate
```

```
        # Open/Close Button
        self.open_button = ttk.Button(self.master, text="Open Port", command=s
        self.open_button.grid(column=4, row=0)

        # Text area for displaying incoming data
        self.text_area = scrolledtext.ScrolledText(self.master, state='disable
        self.text_area.grid(column=0, row=1, columnspan=5)

        # Text input for sending data
        self.text_input = ttk.Entry(self.master, width=60)
        self.text_input.grid(column=0, row=2, columnspan=4)

        # Send Button
        self.send_button = ttk.Button(self.master, text="Send", command=self.s
        self.send_button.grid(column=4, row=2)

    def open_port(self):
        if self.serial_port and self.serial_port.is_open:
            self.serial_port.close()
            self.open_button.config(text="Open Port")
        else:
            try:
                self.serial_port = serial.Serial(self.com_port.get(), self.bau
                self.open_button.config(text="Close Port")
                threading.Thread(target=read_from_port, args=(self.serial_port
            except Exception as e:
                print("Error opening the serial port: ", e)

    def send_data(self):
        if self.serial_port and self.serial_port.is_open:
            try:
                self.serial_port.write(self.text_input.get().encode('utf-8'))
                self.text_area.configure(state='normal')
                self.text_area.insert(tk.END, "Sent: " + self.text_input.get()
                self.text_area.configure(state='disabled')
                self.text_input.delete(0, tk.END)
            except Exception as e:
                print("Error sending data: ", e)
        else:
            print("Serial port is not open.")
```

Main Execution

```
if __name__ == "__main__":
    root = tk.Tk()
    app = SerialApp(root)
    root.mainloop()
```

This script provides a basic GUI for a Serial Terminal Application using Tkinter and pySerial. It allows the user to select a COM port and baud rate, open/close the selected port, send data, and view incoming data. Note that this is a simplified example and may need adjustments based on your specific requirements and environment.

---

unit_tests.py for Serial Terminal Application

This Python script provides a basic framework for unit testing the Serial Terminal Application. It uses the `unittest` framework to define and run tests on the application's functionality. Given the nature of the Serial Terminal Application, these tests will focus on the non-hardware interaction parts of the code, such as data processing and UI logic. For simplicity, this example assumes that the `main.py` script (the Serial Terminal Application) has functions and classes that can be imported and tested in isolation.

Imports

```
import unittest
from unittest.mock import patch
from main import SerialApp, read_from_port
```

Mock Setup

Before writing the test cases, we'll need to mock the serial port to simulate reading from and writing to a COM port without requiring actual hardware.

```
class MockSerialPort:
    def __init__(self, *args, **kwargs):
        self.is_open = False
        self.data = "Test data\n".encode('utf-8')  # Simulated incoming data
        self.received_data = b""

    def open(self):
        self.is_open = True

    def close(self):
        self.is_open = False

    def readline(self):
        return self.data

    def write(self, data):
        self.received_data += data

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        pass
```

Test Cases

```
class TestSerialApp(unittest.TestCase):
    @patch('main.serial.Serial', return_value=MockSerialPort())
    def test_open_close_port(self, mock_serial):
        app = SerialApp(None)  # Passing None because we don't need an actual
        app.com_port.set("COM1")
        app.baud_rate.set("9600")

        # Test opening the port
        app.open_port()
        self.assertTrue(app.serial_port.is_open)
```

```
        # Test closing the port
        app.open_port()
        self.assertFalse(app.serial_port.is_open)

    @patch('main.serial.Serial', return_value=MockSerialPort())
    def test_send_data(self, mock_serial):
        app = SerialApp(None)
        app.com_port.set("COM1")
        app.baud_rate.set("9600")
        app.open_port()

        test_data = "Hello, World!"
        app.text_input.insert(0, test_data)
        app.send_data()

        self.assertEqual(app.serial_port.received_data.decode('utf-8'), test_d

if __name__ == '__main__':
    unittest.main()
```

Explanation

- **Mock Setup**: We create a `MockSerialPort` class to simulate the behavior of a real serial port. This allows us to test the opening and closing of the port, as well as sending and receiving data, without needing actual hardware.
- **Test Cases**:
  - `test_open_close_port` : This test checks if the `SerialApp` class can successfully open and close a COM port using the mocked serial port.
  - `test_send_data` : This test verifies that data sent through the `SerialApp` class is correctly transmitted to the mocked serial port.

To run these tests, save the code in a file named `unit_tests.py` and ensure it's in the same directory as your `main.py` script. Then, run the tests from the command line with `python -m unittest unit_tests.py`.

---

integration_tests.py for Serial Terminal Application

This Python script provides a basic framework for integration testing of the Serial Terminal Application. It uses the `unittest` framework to define and run tests that verify the integration between different components of the application. Given the nature of the Serial Terminal Application, these tests will focus on the integration of the main functional components, assuming that `main.py` (the Serial Terminal Application) has been structured to allow for such testing.

Imports

```
import unittest
from unittest.mock import MagicMock, patch
from main import SerialApp
```

Mock Setup

To facilitate integration testing without requiring actual serial hardware, we'll mock the serial port interactions. This approach allows us to simulate the behavior of the serial port and focus on the integration aspects of the application.

```
class MockSerialPort:
    def __init__(self, *args, **kwargs):
        self.is_open = False
        self.data = "Mocked data\n".encode('utf-8')  # Simulated incoming data
        self.received_data = b""

    def open(self):
        self.is_open = True

    def close(self):
        self.is_open = False
```

```
    def readline(self):
        return self.data

    def write(self, data):
        self.received_data += data

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        pass
```

Test Cases

```
class TestSerialAppIntegration(unittest.TestCase):
    @patch('main.serial.Serial', return_value=MockSerialPort())

    def test_integration_send_receive(self, mock_serial):
        # Initialize the application with a mocked root to avoid GUI operation
        app = SerialApp(None)  # Passing None because we don't need an actual
        app.com_port.set("COM1")
        app.baud_rate.set("9600")

        # Simulate opening the port
        app.open_port()
        self.assertTrue(app.serial_port.is_open, "Port should be open")

        # Simulate sending data
        test_data = "Integration Test"
        app.text_input.insert(0, test_data)
        app.send_data()

        # Verify that data is sent
        self.assertEqual(app.serial_port.received_data.decode('utf-8'), test_d

        # Simulate receiving data and verify it's processed correctly
        # Note: In a real application, you might have a callback or listener f
        # Here, we directly call a hypothetical method that processes incoming
        app.process_incoming_data()  # Assuming this method exists and updates
```

```
            self.assertTrue(app.some_ui_element_or_state_updated, "UI element or s

if __name__ == '__main__':
    unittest.main()
```

Explanation

- **Mock Setup**: We create a `MockSerialPort` class to simulate the behavior of a real serial port. This allows us to test the integration of components without needing actual hardware.
- **Test Cases**:
  - `test_integration_send_receive` : This test checks the integration between the GUI components and the serial port handling components. It verifies that data can be sent and received correctly, and that the application's state or UI is updated accordingly.

To run these tests, save the code in a file named `integration_tests.py` and ensure it's in the same directory as your `main.py` script. Then, run the tests from the command line with `python -m unittest integration_tests.py`.

---

system_tests.py for Serial Terminal Application

This Python script outlines a basic framework for system testing of the Serial Terminal Application. It leverages the `unittest` framework to define and execute tests that simulate user interactions with the application as a whole, ensuring that all components work together as expected. This example assumes that `main.py` (the Serial Terminal Application) is structured in a way that allows for such testing, possibly with some modifications for testability.

Imports

```
import unittest
from unittest.mock import patch
from main import SerialApp
import tkinter as tk
```

Test Setup

System testing often involves simulating user interactions with the GUI. To achieve this without actual user input, we can programmatically trigger GUI events or call methods that represent user actions.

Test Cases

```
class TestSerialAppSystem(unittest.TestCase):
    def setUp(self):
        # Set up a Tkinter root window for testing
        self.root = tk.Tk()
        self.root.withdraw()  # Hide the GUI during tests
        self.app = SerialApp(self.root)

    def tearDown(self):
        # Clean up after each test
        if self.app.serial_port and self.app.serial_port.is_open:
            self.app.serial_port.close()
        self.app = None
        self.root.destroy()

    @patch('main.serial.Serial')
    def test_system_open_close_port(self, mock_serial):
        # Simulate selecting a COM port and opening it
        self.app.com_port.set("COM1")
        self.app.open_port()
        self.assertTrue(mock_serial.called, "Serial port should be opened")
```

```
        # Simulate closing the port
        self.app.open_port()
        self.assertFalse(self.app.serial_port.is_open, "Serial port should be

    @patch('main.serial.Serial')
    def test_system_send_data(self, mock_serial):
        mock_serial_instance = mock_serial.return_value
        mock_serial_instance.readline.return_value = b"Echoed data\n"

        # Simulate opening the port, sending data, and receiving echoed data
        self.app.com_port.set("COM1")
        self.app.open_port()
        self.app.text_input.insert(0, "Test data")
        self.app.send_data()

        # Verify that data was sent
        mock_serial_instance.write.assert_called_with(b"Test data")

        # Assuming the application has a mechanism to process and display incc
        # we would also verify that the echoed data is displayed. This might i
        # checking the contents of a text widget or other UI elements.

if __name__ == '__main__':
    unittest.main()
```

Explanation

- **Test Setup**: Before each test, we set up a Tkinter root window and initialize the `SerialApp` class. After each test, we clean up by closing any open serial ports and destroying the Tkinter root.
- **Test Cases**:
  - `test_system_open_close_port` : This test simulates the user selecting a COM port and toggling it open and closed. It verifies that the serial port is opened and closed as expected.
  - `test_system_send_data` : This test simulates the user opening a COM port, sending data, and checks if the data is sent through the serial port. It also sets

up the expectation for receiving echoed data, which could be extended to verify that incoming data is correctly processed and displayed.

To run these tests, save the code in a file named `system_tests.py` and ensure it's in the same directory as your `main.py` script. Then, execute the tests from the command line with `python -m unittest system_tests.py`. Note that these tests are simplified and may need adjustments based on the specific implementation details of your Serial Terminal Application.

---

acceptance_tests.py for Serial Terminal Application

This Python script provides a framework for acceptance testing of the Serial Terminal Application. It utilizes the `unittest` framework to define and execute tests that verify the application meets the end-user requirements and behaves as expected from a user's perspective. These tests simulate real-world usage scenarios to ensure the application's functionality and usability meet the specified acceptance criteria.

Imports

```
import unittest
from unittest.mock import patch
from main import SerialApp
import tkinter as tk
```

Test Setup

Acceptance testing often involves simulating end-to-end scenarios that reflect how the user will interact with the application. To facilitate this, we can programmatically interact

with the application's GUI components.

Test Cases

```
class TestSerialAppAcceptance(unittest.TestCase):
    def setUp(self):
        # Set up a Tkinter root window for testing
        self.root = tk.Tk()
        self.root.withdraw()  # Optionally hide the GUI during tests
        self.app = SerialApp(self.root)

    def tearDown(self):
        # Clean up after each test
        if self.app.serial_port and self.app.serial_port.is_open:
            self.app.serial_port.close()
        self.app = None
        self.root.destroy()

    @patch('main.serial.Serial')
    def test_acceptance_scenario_open_send_close(self, mock_serial):
        # Scenario: User opens a COM port, sends data, and closes the port
        self.app.com_port.set("COM1")
        self.app.baud_rate.set("9600")

        # Simulate opening the port
        self.app.open_port()
        self.assertTrue(mock_serial.called, "Serial port should be opened")

        # Simulate sending data
        test_data = "Hello, World!"
        self.app.text_input.insert(0, test_data)
        self.app.send_data()
        mock_serial_instance = mock_serial.return_value
        mock_serial_instance.write.assert_called_with(test_data.encode('utf-8'

        # Simulate closing the port
        self.app.open_port()
        self.assertFalse(self.app.serial_port.is_open, "Serial port should be

    @patch('main.serial.Serial')
```

```
    def test_acceptance_scenario_receive_data(self, mock_serial):
        # Scenario: User opens a COM port and receives data
        mock_serial_instance = mock_serial.return_value

        mock_serial_instance.readline.return_value = b"Received data\n"

        self.app.com_port.set("COM1")
        self.app.open_port()

        # Assuming the application has a mechanism to automatically display in
        # we would verify that the received data is displayed. This might invc
        # checking the contents of a text widget or other UI elements.
        # Note: Actual implementation of this verification will depend on how
        # handles incoming data and updates the GUI.

if __name__ == '__main__':
    unittest.main()
```

Explanation

- **Test Setup**: Before each test, we set up a Tkinter root window and initialize the `SerialApp` class without displaying the GUI. After each test, we ensure any open serial ports are closed, and we clean up the application and Tkinter root.
- **Test Cases**:
  - `test_acceptance_scenario_open_send_close` : This test simulates a user opening a COM port, sending data, and then closing the port. It verifies that the application can open and close the port as expected and that data is sent correctly.
  - `test_acceptance_scenario_receive_data` : This test simulates a user opening a COM port and receiving data. It sets up the expectation for receiving data and would include verification that the data is correctly displayed in the application's GUI.

To run these tests, save the code in a file named `acceptance_tests.py` and ensure it's in the same directory as your `main.py` script. Execute the tests from the command line with `python -m unittest acceptance_tests.py` . These tests are designed to verify that the

application meets its acceptance criteria from an end-user perspective, focusing on key user scenarios and interactions.
