SCADA IED Simulator

User Manual

Version 2.0

Modbus TCP Server Simulator for Raspberry Pi

Table of Contents

1. Introduction

The SCADA IED Simulator is a professional-grade Modbus TCP server designed for testing, development, and training purposes. It provides a complete simulation environment for Intelligent Electronic Devices (IEDs) commonly used in industrial control systems and SCADA networks.

1.1 Key Features

- Full Modbus TCP server implementation supporting all standard function codes
- 100 registers per type (Coils, Discrete Inputs, Holding Registers, Input Registers)
- Web-based Human Machine Interface (HMI) for monitoring and control
- Live data simulation with configurable waveform generators
- Custom variable management with engineering unit scaling
- Configuration export and import functionality
- Real-time activity logging and system statistics

1.2 Use Cases

This simulator is designed for:

- Testing SCADA software and RTU/PLC integrations
- Developing and debugging Modbus client applications
- Training personnel on industrial communication protocols
- Simulating field devices during system commissioning
- Creating test scenarios for alarm and trending systems

2. System Requirements

2.1 Hardware Requirements

Component	Requirement
Platform	Raspberry Pi 4 (recommended) or any Linux system
Memory	2 GB RAM minimum (4 GB recommended)
Storage	2 GB free disk space
Network	Ethernet or WiFi connectivity

2.2 Software Requirements

- Raspberry Pi OS (64-bit recommended) or Ubuntu/Debian Linux
- Python 3.7 or higher
- Flask 3.0.0 or higher
- pymodbus 3.6.0 or higher

3. Installation

3.1 Quick Installation

Follow these steps to install the SCADA IED Simulator:

Step 1: Create Project Directory

```
mkdir -p ~/scada_simulator/templates
cd ~/scada_simulator
```

Step 2: Create Virtual Environment

```
python3 -m venv venv
source venv/bin/activate
```

Step 3: Install Dependencies

```
pip install flask pymodbus
```

Step 4: Copy Application Files

Copy the following files to your project directory:

- enhanced_app.py - Main application file
- templates/index.html - Dashboard interface
- templates/simulation.html - Live simulation interface
- templates/variables.html - Custom variables interface

Step 5: Run the Simulator

```
python3 enhanced_app.py
```

3.2 Configuring as a System Service

To run the simulator automatically at system startup, create a systemd service file:

```
sudo nano /etc/systemd/system/scada-simulator.service
```

Add the following content:

```
[Unit]
Description=SCADA IED Simulator
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/scada_simulator
ExecStart=/home/pi/scada_simulator/venv/bin/python3 enhanced_app.py
Restart=always
```

```
[Install]
WantedBy=multi-user.target
```

Enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable scada-simulator
sudo systemctl start scada-simulator
```

4. Configuration

4.1 Network Settings

The default network configuration is defined in enhanced_app.py:

Parameter	Default	Description
MODBUS_ADDRESS	254	Modbus Unit ID
SERVER_IP	0.0.0.0	Listen on all interfaces
SERVER_PORT	5002	Modbus TCP port
FLASK_PORT	5000	Web interface port

4.2 Firewall Configuration

If using UFW firewall, allow the required ports:

```
sudo ufw allow 5000/tcp    # Web interface
sudo ufw allow 5002/tcp    # Modbus TCP
```

5. Web Interface

Access the web interface by navigating to http://<device-ip>:5000 in a web browser.

5.1 Dashboard

The Dashboard provides real-time monitoring and control:

- System Statistics: Uptime, total requests, and server status
- Connection Information: Protocol, IP address, port, and Unit ID
- Register Controls: Toggle digital values and set analog values
- Activity Log: Real-time display of all register changes

5.2 Live Simulation

The Live Simulation page provides four waveform generators:

- Sine Wave: Configurable amplitude, frequency, and offset
- Ramp/Sawtooth: Linear increase from minimum to maximum value
- Random Walk: Simulates noisy sensor readings
- Square Wave: Alternates between high and low values

The Digital Pulse section allows toggling of discrete inputs to simulate events such as breaker trips or alarms.

5.3 Custom Variables

Create custom variables with engineering unit scaling:

- Define meaningful names (e.g., VOLTAGE_L1, PRESSURE_TANK1)
- Map to specific register types and addresses
- Configure scaling to convert raw values to engineering units
- Export and import variable configurations

6. Modbus Register Reference

6.1 Register Types

Register Type	Function Codes	Address Range	Access
Coils	01, 05, 15	0-99	Read/Write
Discrete Inputs	02	0-99	Read Only
Holding Registers	03, 06, 16	0-99	Read/Write
Input Registers	04	0-99	Read Only

6.2 Supported Function Codes

Code	Function	Description
01	Read Coils	Read digital output status
02	Read Discrete Inputs	Read digital input status
03	Read Holding Registers	Read analog output values
04	Read Input Registers	Read analog input values
05	Write Single Coil	Write single digital output
06	Write Single Register	Write single analog value
15	Write Multiple Coils	Write multiple digital outputs
16	Write Multiple Registers	Write multiple analog values

7. API Reference

The simulator provides a REST API for programmatic access.

7.1 Status Endpoints

GET /api/status

Returns current values of all registers (first 10 of each type).

GET /api/system_status

Returns system statistics including uptime, request count, and recent changes.

7.2 Register Write Endpoints

POST /api/set_coil

Request body: {"address": 0, "value": 1}

POST /api/set_discrete_input

Request body: {"address": 0, "value": 1}

POST /api/set_holding_register

Request body: {"address": 0, "value": 1000}

POST /api/set_input_register

Request body: {"address": 0, "value": 1000}

7.3 Configuration Endpoints

GET /api/export_config

Exports complete system configuration as JSON.

POST /api/import_config

Imports configuration from JSON payload.

GET /api/custom_variables

Returns all custom variable definitions.

POST /api/custom_variables

Creates or updates a custom variable.

8. Troubleshooting

8.1 Web Interface Not Accessible

1. Verify the application is running: ps aux | grep enhanced_app
2. Check the port is listening: sudo netstat -tlnp | grep 5000
3. Verify firewall rules: sudo ufw status
4. Test locally: curl http://localhost:5000

8.2 Modbus Connection Failed

5. Check port 5002 is listening: sudo netstat -tlnp | grep 5002
6. Verify firewall allows port 5002: sudo ufw allow 5002/tcp
7. Confirm correct Unit ID (254) is configured in your client
8. Test connectivity: ping <device-ip>

8.3 Service Won't Start

9. Check service status: sudo systemctl status scada-simulator
10. View logs: sudo journalctl -u scada-simulator -f
11. Verify file paths in service file are correct
12. Check Python virtual environment is properly configured

8.4 Finding Device IP Address

```
hostname -I
```

Or for more detail:

```
ip addr show
```

9. Integration Examples

9.1 Python Client Example

```
from pymodbus.client import ModbusTcpClient

client = ModbusTcpClient('192.168.1.100', port=5002)
client.connect()

# Read 10 input registers starting at address 0
result = client.read_input_registers(0, 10, slave=254)
print(result.registers)

# Write value 1000 to holding register 0
client.write_register(0, 1000, slave=254)

client.close()
```

9.2 SEL RTAC Configuration

Configure the following settings in SEL RTAC:

- Protocol: Modbus TCP
- IP Address: <Raspberry Pi IP>
- Port: 5002
- Unit ID: 254
- Scan Rate: 1000ms (recommended)

9.3 Generic SCADA Configuration

For most SCADA systems, configure:

- Device Type: Modbus TCP Slave
- Connection: TCP/IP
- IP Address: <Device IP>
- Port: 5002
- Slave/Unit ID: 254

End of Document