# **Crypto Trade Simulator - Documentation**

## **Overview**

The Crypto Trade Simulator is a desktop application that allows users to simulate cryptocurrency trading operations with real-time market data. The application connects to cryptocurrency exchanges via WebSockets to receive live orderbook data, calculates various trading metrics, and provides performance benchmarking capabilities.

## **Features**

* **Real-time Market Data**: Connects to OKX exchange via WebSocket to receive live L2 orderbook data
* **Trade Simulation**: Calculates expected slippage, fees, and market impact for potential trades
* **Performance Monitoring**: Tracks and visualizes data processing, UI update, and end-to-end latency metrics
* **Benchmarking Tools**: Built-in tools to measure and report application performance
* **Dynamic Visualization**: Real-time orderbook display and latency metric graphs

## **Interface**

The application features a tabbed interface with two main sections:

### **Simulation Tab**

#### **Input Parameters (Left Panel)**

* **Exchange**: Currently fixed to OKX
* **Asset**: Trading pair (e.g., BTC-USDT-SWAP)
* **Order Type**: Market or Limit
* **Order Side**: Buy or Sell
* **Quantity**: Trade size in USD
* **Fee Tier**: Fee rate based on exchange tier level
* **Volatility Override**: Optional manual volatility setting
* **Update Speed**: Refresh rate in milliseconds
* **Connection Status**: WebSocket connection indicator
* **Market Statistics**: Current volatility and average volume

#### **Output Parameters (Right Panel)**

* **Top Bid/Ask**: Current best bid and ask prices
* **Spread**: Price difference between top bid and ask
* **Expected Slippage**: Estimated price slippage for the trade
* **Estimated Fees**: Trading fees based on selected tier
* **Market Impact**: Estimated price impact of the trade
* **Net Cost/Proceeds**: Total cost or proceeds after fees and slippage
* **Maker/Taker Ratio**: Predicted order fill distribution
* **Fee Rate**: Effective fee rate based on maker/taker ratio
* **Latency**: Current data processing latency
* **Order Book Preview**: Visual display of current market depth

### **Latency Metrics Tab**

* **Latency Metrics Display**: Current data processing, UI update, and end-to-end latency
* **Benchmark Controls**: Tools to run benchmarks and generate reports
* **Latency Visualization**: Time series and distribution plots of latency metrics

## **Technical Implementation**

### **Key Components**

1. **TradeSimulatorUI**: Main application class handling the UI and data integration
2. **MarketModels**: Provides trading-related calculations (imported from models.py)
3. **LatencyTracker**: Records and analyzes performance metrics
4. **UILatencyMonitor**: Monitors UI update performance
5. **EndToEndLatencyMonitor**: Measures full processing cycles

### **Data Flow**

1. The application connects to the exchange WebSocket API
2. Real-time orderbook data is received and processed
3. Market models are updated with the latest data
4. Trading metrics are calculated based on user inputs
5. Performance metrics are captured throughout the process
6. The UI is updated at the specified refresh rate

### **Performance Monitoring**

The application uses decorator-based latency tracking (@measure\_latency) to monitor:

* Data processing time
* UI update time
* End-to-end cycle time

These metrics are recorded, visualized, and can be exported as reports.

## **Using the Application**

### **Basic Operation**

1. **Start the application**: Run gui\_app.py
2. **Connect to market data**: Enter a valid asset pair and click "Connect"
3. **Configure trade parameters**: Select order type, side, quantity, and fee tier
4. **View simulation results**: Check the right panel for trade metrics
5. **Adjust update speed**: Set your preferred UI refresh rate

### **Performance Benchmarking**

1. Go to the "Latency Metrics" tab
2. Click "Run Benchmark" to open the benchmark configuration
3. Set the desired number of cycles and load parameters
4. Click "Start Benchmark" to run the test
5. View results in the latency visualization charts
6. Generate a detailed report with "Generate Report"

### **Advanced Features**

* **Volatility Override**: Enter a manual volatility value to override automatic calculations
* **Maker/Taker Ratio**: The application predicts how your order will be filled based on market conditions
* **Dynamic Refresh Rate**: Adjust the UI update speed to balance between responsiveness and performance

## **System Requirements**

* Python 3.6 or higher
* Required packages:
  + tkinter
  + websockets
  + matplotlib
  + numpy
  + json
  + asyncio
  + Threading