Thank you for the excellent feedback! I'm glad the System Kernel PRD perfectly captured your vision. Let's proceed with Level 2 components, starting with the DataHandler.

# **Product Requirements Document (PRD): DataHandler Component**

**Document Version:** 1.0  
 **Date:** June 20, 2025  
 **Component Level:** 2 - Data Pipeline  
 **Status:** Master Specification

## **1. Component Identity**

### **1.1 Component Name**

**DataHandler** (Market Data Abstraction Layer)

### **1.2 Primary Role**

The DataHandler serves as the exclusive entry point for all market data into the system. It provides a complete abstraction between data sources (live feed or historical file) and the trading logic, ensuring the core system remains identical whether backtesting or live trading.

### **1.3 Single Responsibility**

To ingest raw market data from either a live feed (Rithmic) or historical file (CSV), normalize it into a standard format, and emit uniform tick events to the system.

### **1.4 Critical Design Principle**

**DIR-DATA-03 Compliance:** The DataHandler MUST implement a clear abstraction layer ensuring that ALL downstream components receive identical data structures and events regardless of whether the system is in backtest or live mode. This is non-negotiable and fundamental to the system's integrity.

## **2. Inputs & Dependencies**

### **2.1 Configuration Input**

From settings.yaml:

data:

mode: "backtest" # or "live"

symbol: "ES" # Single asset only (DIR-SYS-02)

backtest:

file\_path: "data/ES\_2023\_2024.csv"

replay\_speed: 1.0 # 1.0 = real-time, 0 = as fast as possible

live:

# Credentials from environment variables

# RITHMIC\_USER, RITHMIC\_PASSWORD, RITHMIC\_SYSTEM

### **2.2 Data Sources**

**Backtest Mode:**

* **Input:** CSV file with tick data
* **Format:** timestamp,price,volume
* **Location:** Path specified in configuration

**Live Mode:**

* **Input:** Rithmic API tick stream
* **Connection:** Using credentials from environment
* **Contract:** Single futures contract specified in config

### **2.3 External Dependencies**

* **Backtest:** CSV file must exist and be readable
* **Live:** Network connection to Rithmic servers
* **Both:** Event bus from System Kernel

## **3. Processing Logic**

### **3.1 Initialization**

The DataHandler uses an abstract base class pattern:

AbstractDataHandler (base class)

├── LiveDataHandler (for production)

└── BacktestDataHandler (for testing)

**Initialization Steps:**

1. **Mode Detection**
   * Read mode from configuration
   * Instantiate appropriate handler class
2. **Handler-Specific Setup** **BacktestDataHandler:**
   * Open CSV file
   * Read header to verify format
   * Create file reader positioned at first data row
   * Log: "Backtest data loaded: [filename], [row\_count] ticks"
3. **LiveDataHandler:**
   * Load credentials from environment
   * Create Rithmic client instance
   * Log: "Connecting to Rithmic..."
4. **Event Bus Connection**
   * Get reference to system event bus
   * Ready to emit events

### **3.2 Data Processing Flow**

#### **3.2.1 Backtest Mode Operation**

**Simple Sequential Processing:**

1. **Read Next Line**
   * Parse CSV line: timestamp,price,volume
   * Convert timestamp to datetime object
   * Convert price and volume to appropriate types

**Create TickData Object** TickData:

symbol: str (from config)

timestamp: datetime

price: float

volume: int

1. **Handle Replay Speed**
   * If replay\_speed = 0: Emit immediately
   * If replay\_speed = 1.0: Calculate appropriate delay
   * Sleep for calculated duration
2. **Emit Event**
   * Create NEW\_TICK event with TickData payload
   * Publish to event bus
3. **End of File**
   * Log: "Backtest complete, [total\_ticks] processed"
   * Emit BACKTEST\_COMPLETE event
   * Stop processing

#### **3.2.2 Live Mode Operation**

**Real-Time Stream Processing:**

1. **Connect to Rithmic**
   * Establish connection with retry logic
   * Subscribe to tick data for configured symbol
   * Log: "Connected to Rithmic, subscribed to [symbol]"
2. **On Tick Received**
   * Extract: timestamp, price, volume from Rithmic message
   * Create same TickData object as backtest mode
3. **Emit Event**
   * Immediate emission (no delay)
   * Same NEW\_TICK event structure
4. **Connection Management**
   * Monitor connection health
   * Automatic reconnection on disconnect
   * Log all connection state changes

### **3.3 Critical Abstraction Guarantee**

**Both modes MUST produce identical output:**

NEW\_TICK Event:

type: "NEW\_TICK"

payload: TickData {

symbol: "ES"

timestamp: 2025-06-20 10:30:45.123

price: 5150.25

volume: 10

}

Downstream components cannot and should not know or care about the data source.

## **4. Outputs & Events**

### **4.1 Primary Output**

**Event Name:** NEW\_TICK **Frequency:** Every tick (hundreds to thousands per minute) **Payload Structure:**

TickData:

symbol: str # Always from config

timestamp: datetime # Microsecond precision

price: float # Tick price

volume: int # Tick volume

### **4.2 Status Events**

* **BACKTEST\_COMPLETE:** Emitted when CSV file fully processed
* **CONNECTION\_LOST:** Emitted on Rithmic disconnection
* **CONNECTION\_RESTORED:** Emitted on successful reconnection

## **5. Critical Requirements**

### **5.1 Data Integrity Requirements**

* **No Data Loss:** Every tick from source must generate an event
* **No Duplicates:** Each tick processed exactly once
* **Ordering Preserved:** Ticks emitted in chronological order
* **No Modification:** Price and volume passed through unchanged

### **5.2 Performance Requirements**

* **Backtest Mode:** Process historical ticks as fast as system can handle (when replay\_speed = 0)
* **Live Mode:** Sub-millisecond latency from receipt to event emission
* **Memory Usage:** Constant memory footprint (no accumulation)

### **5.3 Reliability Requirements**

* **Backtest:** Graceful handling of malformed CSV lines (log and skip)
* **Live:** Automatic reconnection with exponential backoff
* **Both:** Clear error messages for configuration issues

### **5.4 Abstraction Requirements**

* **Identical Interface:** Same event structure regardless of mode
* **No Mode Leakage:** Downstream components remain mode-agnostic
* **Single Asset:** Only one symbol processed per instance (DIR-SYS-02)

## **6. Integration Points**

### **6.1 Upstream Integration**

**Backtest Mode:**

* Reads from: Filesystem (CSV file)
* File format: Standard tick data CSV

**Live Mode:**

* Connects to: Rithmic API
* Protocol: Rithmic's proprietary protocol
* Authentication: Via environment variables

### **6.2 Downstream Integration**

**Primary Consumer:** BarGenerator

* Subscribes to: NEW\_TICK events
* Expects: Consistent TickData structure
* Frequency: Every tick

### **6.3 System Integration**

* Initialized by: System Kernel
* Lifecycle: Started after all components initialized
* Shutdown: Stops on system shutdown signal

## **7. Error Handling**

### **7.1 Backtest Mode Errors**

* **File Not Found:** Log error and exit
* **Malformed Line:** Log warning and skip line
* **Empty File:** Log error and exit

### **7.2 Live Mode Errors**

* **Connection Failed:** Retry with exponential backoff (1s, 2s, 4s, 8s, 16s)
* **Authentication Failed:** Log error and exit
* **Subscription Failed:** Log error and retry
* **Unexpected Disconnect:** Attempt automatic reconnection

### **7.3 Common Errors**

* **Invalid Configuration:** Log specific issue and exit
* **Event Bus Failure:** Log error and exit

## **8. Logging Specification**

### **8.1 Startup Logs**

* "DataHandler initializing in [MODE] mode"
* "Loading data from: [filepath]" (backtest)
* "Connecting to Rithmic for symbol: [symbol]" (live)

### **8.2 Operational Logs**

* Every 1000 ticks: "[count] ticks processed"
* Connection state changes (live mode)
* Any errors or warnings

### **8.3 Shutdown Logs**

* "DataHandler shutdown initiated"
* "Total ticks processed: [count]"

## **9. Testing Considerations**

### **9.1 Unit Testing**

* Mock CSV file for backtest mode
* Mock Rithmic connection for live mode
* Verify identical event output from both modes

### **9.2 Integration Testing**

* Connect to BarGenerator
* Verify end-to-end data flow
* Test mode switching

### **9.3 Performance Testing**

* Measure tick processing rate
* Verify no memory leaks
* Test under high-frequency conditions

## **10. Future Considerations (Out of Scope v1.0)**

* Multiple symbol support
* Additional data sources (IB, CQG)
* Tick filtering or preprocessing
* Data persistence or replay
* WebSocket support

These features are explicitly NOT part of Version 1.0 to maintain focus on core functionality.

This DataHandler PRD maintains the same clarity and alignment as the System Kernel. It focuses on the critical abstraction principle while avoiding unnecessary complexity.

**Next Component:** BarGenerator

Shall we proceed?