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

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

## **1. Component Identity**

### **1.1 Component Name**

**BarGenerator** (Time-Series Aggregation Engine)

### **1.2 Primary Role**

The BarGenerator transforms the continuous stream of tick data into discrete, time-based OHLCV bars (candlesticks). It simultaneously maintains two timeframes (5-minute and 30-minute) that form the foundation of the trading strategy.

### **1.3 Single Responsibility**

To aggregate tick data into accurate OHLCV bars for multiple timeframes and emit standardized bar events when each time period completes.

### **1.4 Critical Design Principle**

The BarGenerator must maintain temporal accuracy and handle data gaps gracefully. It serves as the bridge between raw tick data and the structured time series that indicators require.

## **2. Inputs & Dependencies**

### **2.1 Configuration Input**

From settings.yaml:

bars:

timeframes: [5, 30] # Minutes - fixed for the strategy

gap\_fill: true # Forward-fill gaps

### **2.2 Event Input**

**Single Input Event:** NEW\_TICK

* **Source:** DataHandler
* **Frequency:** Continuous (hundreds to thousands per minute)
* **Payload:** TickData object

### **2.3 Dependencies**

* Event bus for receiving ticks and publishing bars
* No external dependencies
* No database or file system access

## **3. Processing Logic**

### **3.1 Core Data Structures**

The BarGenerator maintains two concurrent "work-in-progress" bars:

Active Bars:

├── 5-minute bar (current)

└── 30-minute bar (current)

Bar Structure (OHLCV):

- timestamp: datetime (bar start time)

- open: float (first tick price)

- high: float (highest tick price)

- low: float (lowest tick price)

- close: float (last tick price)

- volume: int (sum of tick volumes)

### **3.2 Bar Construction Logic**

#### **3.2.1 Timestamp Calculation**

**Critical Concept: "Flooring"** Every tick must be assigned to the correct bar based on its timestamp:

For 5-minute bars:

10:32:45 → 10:30:00 (belongs to 10:30-10:35 bar)

10:34:59 → 10:30:00 (same bar)

10:35:00 → 10:35:00 (new bar)

For 30-minute bars:

10:32:45 → 10:30:00 (belongs to 10:30-11:00 bar)

10:59:59 → 10:30:00 (same bar)

11:00:00 → 11:00:00 (new bar)

#### **3.2.2 Processing Each Tick**

**On NEW\_TICK Event:**

1. **Extract Tick Data**
   * Get timestamp, price, volume from event payload
2. **Calculate Bar Timestamps**
   * 5-min bar timestamp = floor(tick.timestamp to 5 minutes)
   * 30-min bar timestamp = floor(tick.timestamp to 30 minutes)
3. **Check for Bar Completion** For each timeframe (5 and 30):  
   * If calculated timestamp > current bar timestamp:
     + Current bar is complete → Finalize and emit
     + Start new bar with current tick
4. **Update Active Bars** For each active bar:  
   * If first tick of bar: open = price
   * Update: high = max(high, price)
   * Update: low = min(low, price)
   * Always: close = price
   * Add: volume += tick.volume

### **3.3 Bar Finalization**

When a bar period ends:

**Create BarData Object** BarData:

symbol: str (from tick)

timestamp: datetime (bar start time)

open: float

high: float

low: float

close: float

volume: int

timeframe: int (5 or 30)

1. **Emit Bar Event**
   * For 5-minute bars: Emit NEW\_5MIN\_BAR
   * For 30-minute bars: Emit NEW\_30MIN\_BAR
2. **Reset for New Bar**
   * Create new empty bar structure
   * Set open to first tick of new period

### **3.4 Gap Handling**

**Critical Requirement: No Missing Bars**

When a tick arrives after a gap (no ticks for one or more complete bars):

1. **Detect Gap**
   * If new bar timestamp > expected next bar
2. **Forward-Fill Missing Bars**
   * For each missing bar period:
     + Create synthetic bar
     + OHLC = previous bar's close price
     + Volume = 0
     + Emit as normal bar
3. **Continue Normal Processing**
   * Process current tick for new bar

**Example:**

Last bar: 10:30:00-10:35:00, close=5150.25

Next tick: 10:42:17, price=5151.00

Must emit:

- 10:35:00 bar (OHLC=5150.25, volume=0)

- 10:40:00 bar (OHLC=5150.25, volume=0)

Then start 10:40:00 bar with new tick

## **4. Outputs & Events**

### **4.1 Primary Outputs**

**Event: NEW\_5MIN\_BAR**

* **Frequency:** Every 5 minutes (when market active)
* **Payload:** BarData with timeframe=5

**Event: NEW\_30MIN\_BAR**

* **Frequency:** Every 30 minutes (when market active)
* **Payload:** BarData with timeframe=30

### **4.2 BarData Structure**

BarData:

symbol: "ES"

timestamp: 2025-06-20 10:30:00 # Bar START time

open: 5150.25

high: 5151.50

low: 5149.75

close: 5151.00

volume: 1250

timeframe: 5 # or 30

### **4.3 Event Timing**

* Events emitted immediately when bar period completes
* No delay or buffering
* Maintains chronological order

## **5. Critical Requirements**

### **5.1 Accuracy Requirements**

* **Temporal Precision:** Bars must align exactly with clock boundaries
* **No Data Loss:** Every tick must be included in exactly one bar
* **No Overlaps:** Bars must not overlap in time
* **Correct Aggregation:** OHLCV values must be mathematically correct

### **5.2 Reliability Requirements**

* **Gap Handling:** Must forward-fill to maintain continuous series
* **First Tick Handling:** Correctly initialize first bar of session
* **Memory Safety:** No memory leaks from accumulated data

### **5.3 Performance Requirements**

* **Processing Latency:** <100 microseconds per tick
* **Bar Emission Latency:** <1 millisecond per bar
* **Memory Usage:** Constant (only 2 active bars)

### **5.4 Consistency Requirements**

* **Deterministic Output:** Same ticks must produce same bars
* **Mode Agnostic:** Same logic for backtest and live
* **Single Asset:** Process one symbol only (DIR-SYS-02)

## **6. Integration Points**

### **6.1 Upstream Integration**

**Single Source:** DataHandler

* Event: NEW\_TICK
* Contains: TickData with timestamp, price, volume
* Frequency: Continuous stream

### **6.2 Downstream Integration**

**Primary Consumer:** IndicatorEngine

* Events: NEW\_5MIN\_BAR, NEW\_30MIN\_BAR
* Expects: Complete, accurate OHLCV data
* Uses: Both timeframes for different indicators

### **6.3 System Integration**

* Initialized by: System Kernel
* Lifecycle: Passive component (event-driven)
* State: Minimal (just two active bars)

## **7. State Management**

### **7.1 Internal State**

State Structure:

├── active\_5min\_bar

│ ├── timestamp

│ ├── open

│ ├── high

│ ├── low

│ ├── close

│ └── volume

└── active\_30min\_bar

└── (same structure)

### **7.2 State Persistence**

* No persistence required
* State rebuilt from tick stream
* Lost bars not recoverable

## **8. Error Handling**

### **8.1 Invalid Data**

* **Negative Price:** Log error, skip tick
* **Zero/Negative Volume:** Log warning, include with 0 volume
* **Timestamp in Past:** Log error, skip tick

### **8.2 System Errors**

* **Event Bus Failure:** Log critical, no recovery
* **Memory Allocation Failure:** Log critical, system exit

## **9. Logging Specification**

### **9.1 Startup**

* "BarGenerator initialized for timeframes: [5, 30]"

### **9.2 Operational**

* Every new bar: "[TIMEFRAME]-min bar completed: [timestamp]"
* Gap detected: "Gap detected, forward-filling [n] bars"
* Errors: Specific error messages with tick data

### **9.3 Debug (if enabled)**

* Every tick: "Processing tick: [timestamp] [price] [volume]"
* Bar updates: "Updated [timeframe]-min bar: OHLCV"

## **10. Testing Considerations**

### **10.1 Unit Tests**

* Normal tick sequences
* Gap scenarios
* Boundary conditions (market open/close)
* High-frequency tick bursts

### **10.2 Validation Tests**

* Compare output with external charting software
* Verify OHLCV calculations
* Confirm timestamp alignment

### **10.3 Edge Cases**

* First tick of day
* Ticks at exact bar boundaries
* Long gaps (hours)
* Rapid tick sequences

## **11. Implementation Notes**

### **11.1 Timestamp Precision**

* Use microsecond precision throughout
* Be careful with timezone handling
* Consider market hours in production

### **11.2 Numerical Precision**

* Use float64 for prices
* Handle rounding consistently
* Volume as integer only

### **11.3 Performance Optimization**

* Pre-calculate next bar timestamps
* Minimize object creation
* Use efficient data structures

## **12. What This Component Does NOT Do**

* Does NOT convert to Heiken Ashi (that's IndicatorEngine)
* Does NOT store historical bars
* Does NOT calculate any indicators
* Does NOT validate tick prices against limits
* Does NOT handle multiple symbols
* Does NOT support custom timeframes
* Does NOT persist state between runs

This BarGenerator PRD completes the tick-to-bar transformation layer, maintaining the same clarity and focus as previous components. It handles the critical task of time series construction while remaining simple and reliable.

**Next Component:** IndicatorEngine

Shall we proceed?