IoT Data Processing Mini Language

Use Case Documentation

**Version:** 1.0

**Domain:** IoT Sensor Data Processing, Filtering, and Automated Response

# 1. Introduction

IoTScript is a domain-specific programming language designed for processing, filtering, and responding to IoT sensor data in real-time. The language provides intuitive constructs for common IoT scenarios including data acquisition, threshold monitoring, data transformation, aggregation, and automated alerting.

**Primary Use Cases:**

* Environmental monitoring (temperature, humidity, air quality)
* Industrial equipment monitoring and predictive maintenance
* Smart home automation and energy management
* Agricultural sensor networks for irrigation and climate control
* Healthcare device monitoring and patient alert systems

# 2. Statement Types and Use Cases

## 2.1 Variable Declaration (var)

### Purpose

Variable declarations allow storing sensor readings, computed values, and configuration parameters for later use in processing pipelines.

### Why It's Needed

IoT applications require storing intermediate values such as threshold limits, calibration constants, aggregated metrics, and historical data points for comparison and trend analysis.

### Example: Smart Greenhouse Temperature Management

// Define operational thresholds

var min\_temp = 18.0

var max\_temp = 28.0

var optimal\_temp = 23.0

// Store current reading

read current\_temp from "greenhouse.temp.sensor"

// Calculate deviation from optimal

var temp\_deviation = current\_temp - optimal\_temp

**Justification:** Variables store configuration values that can be adjusted without rewriting logic, and hold computed metrics like deviations for decision-making.

## 2.2 Read Statement (read...from)

### Purpose

The read statement retrieves data from sensors, data streams, or external sources into program variables for processing.

### Why It's Needed

Every IoT application begins with data acquisition. Sensors must be polled or streamed from, and the language needs a clear, declarative way to specify data sources.

### Example: Multi-Sensor Industrial Monitoring

// Read from multiple sensors

read motor\_temp from "factory.motor1.temperature"

read motor\_vibration from "factory.motor1.vibration"

read motor\_current from "factory.motor1.current"

// Read from time-series database

read historical\_temps from "timeseries.motor1.temp.24h"

**Justification:** Industrial IoT requires monitoring multiple parameters simultaneously. The read statement provides a uniform interface for accessing heterogeneous data sources including real-time sensors and historical databases.

## 2.3 Filter Statement (filter...where)

### Purpose

The filter statement selectively processes data based on specified conditions, essential for focusing on relevant sensor readings.

### Why It's Needed

IoT systems generate massive data streams, but only a subset is typically relevant. Filtering reduces processing overhead, focuses on critical events, and eliminates noise from sensors.

### Example: Air Quality Monitoring Station

// Read air quality index stream

read aqi\_readings from "city.airquality.stream"

// Filter only unhealthy readings (AQI > 100)

filter aqi\_readings where aqi\_readings.value > 100

// Alert on filtered data

alert "Air quality unhealthy" when aqi\_readings.value > 150

**Justification:** Air quality stations produce continuous readings, but alerts are only needed when thresholds are exceeded. Filtering eliminates normal readings and focuses resources on problematic data.

## 2.4 Transform Statement (transform...with)

### Purpose

The transform statement applies mathematical operations or aggregations to modify data, including unit conversions, normalization, and statistical computations.

### Why It's Needed

Sensor data often requires transformation for standardization, unit conversion, calibration correction, or aggregation into meaningful metrics before analysis or storage.

### Example: Weather Station Data Normalization

// Read raw sensor values

read temp\_celsius from "weather.temperature"

read pressure\_hpa from "weather.pressure"

read wind\_speed\_mps from "weather.windspeed"

// Transform to standard units

transform temp\_celsius with temp\_celsius \* 9/5 + 32 // to Fahrenheit

transform pressure\_hpa with pressure\_hpa \* 0.02953 // to inHg

transform wind\_speed\_mps with wind\_speed\_mps \* 2.237 // to mph

// Calculate heat index

read humidity from "weather.humidity"

var heat\_index = -42.379 + 2.049\*temp\_celsius + 10.143\*humidity

**Justification:** Weather data must be presented in user-preferred units and derived metrics like heat index require mathematical transformations of multiple sensor inputs.

## 2.5 Aggregation Functions (avg, sum, max, min, count)

### Purpose

Aggregation functions compute statistical summaries over collections of sensor readings, essential for trend analysis and anomaly detection.

### Why It's Needed

IoT applications often need to analyze patterns over time windows rather than individual readings. Aggregations reduce data volume and reveal trends that single measurements cannot show.

### Example: Smart Building Energy Management

// Read hourly power consumption data

read power\_readings from "building.power.hourly"

// Calculate statistics

var avg\_consumption = avg(power\_readings)

var peak\_consumption = max(power\_readings)

var total\_consumption = sum(power\_readings)

// Detect anomalous usage

var current\_reading = 0

read current\_reading from "building.power.current"

if current\_reading > avg\_consumption \* 1.5 {

alert "Abnormal power consumption detected"

}

**Justification:** Energy management requires understanding consumption patterns. Aggregations enable baseline establishment and anomaly detection by comparing current usage against historical averages.

## 2.6 Conditional Statement (if...else)

### Purpose

Conditional statements enable branching logic based on sensor values or computed conditions, fundamental for decision-making in IoT systems.

### Why It's Needed

IoT applications must respond differently to various conditions. Thermostats adjust heating or cooling, security systems distinguish between alert levels, and industrial controls implement safety protocols based on sensor states.

### Example: Smart Irrigation System

// Read soil moisture and weather forecast

read soil\_moisture from "garden.moisture.sensor"

read rain\_forecast from "weather.forecast.rain"

// Threshold-based irrigation decision

if soil\_moisture < 30 {

if rain\_forecast == true {

emit "skip" to "irrigation.control"

alert "Irrigation postponed - rain expected"

} else {

emit "activate" to "irrigation.control"

alert "Irrigation activated - low moisture"

}

} else {

emit "off" to "irrigation.control"

}

**Justification:** Irrigation decisions depend on multiple factors. Nested conditionals allow complex logic considering both current soil conditions and forecasted weather to optimize water usage.

## 2.7 Loop Statements (while, for)

### Purpose

Loop statements enable repeated operations on data collections or continuous monitoring until conditions change.

### Why It's Needed

IoT systems must process arrays of sensor data, iterate over device collections, or maintain continuous monitoring states. Loops provide the mechanism for repetitive operations without code duplication.

### Example: Multi-Zone HVAC System

// Read temperature sensors from all zones

read zone\_temps from "building.hvac.zones"

var target\_temp = 22.0

var all\_comfortable = true

// Check each zone using for loop

for zone in zone\_temps {

var temp\_diff = zone - target\_temp

if temp\_diff < -2 {

emit "heat\_on" to "hvac.zone." + zone

all\_comfortable = false

}

if temp\_diff > 2 {

emit "cool\_on" to "hvac.zone." + zone

all\_comfortable = false

}

}

if all\_comfortable {

emit "system\_idle" to "hvac.central"

}

**Justification:** Multi-zone HVAC systems must individually control each zone. For loops iterate through all zones applying the same logic, enabling scalable control regardless of building size.

## 2.8 Emit Statement (emit...to)

### Purpose

The emit statement sends data or commands to output destinations, actuators, or downstream systems.

### Why It's Needed

IoT systems must produce actionable outputs. After processing sensor data, results need to be stored in databases, sent to visualization dashboards, or used to control actuators and devices.

### Example: Smart Manufacturing Quality Control

// Read production line sensors

read part\_dimension from "assembly.measurement.sensor"

read part\_weight from "assembly.scale.sensor"

// Quality specifications

var spec\_min\_dim = 49.8

var spec\_max\_dim = 50.2

var spec\_weight = 100.0

// Check quality and emit results

if part\_dimension >= spec\_min\_dim and part\_dimension <= spec\_max\_dim {

emit "pass" to "quality.control.gate"

emit part\_dimension to "database.quality.metrics"

} else {

emit "reject" to "quality.control.gate"

emit part\_dimension to "database.quality.defects"

alert "Part rejected - dimension out of spec"

}

**Justification:** Manufacturing quality control requires both physical actions (actuating gates) and data logging. Emit statements provide a unified interface for controlling hardware and persisting measurement data.

## 2.9 Alert Statement (alert...when)

### Purpose

The alert statement triggers notifications to operators, administrators, or monitoring systems when specific conditions occur.

### Why It's Needed

Human intervention is often required when sensors detect anomalies, thresholds are breached, or systems enter critical states. Alerts ensure timely notification through appropriate channels.

### Example: Hospital Patient Monitoring System

// Read vital signs

read heart\_rate from "patient.hr.monitor"

read blood\_oxygen from "patient.spo2.monitor"

read blood\_pressure\_sys from "patient.bp.systolic"

// Critical thresholds

alert "CRITICAL: Bradycardia detected" when heart\_rate < 50

alert "CRITICAL: Tachycardia detected" when heart\_rate > 120

alert "CRITICAL: Low oxygen saturation" when blood\_oxygen < 90

alert "WARNING: Hypertension detected" when blood\_pressure\_sys > 140

// Combined condition alert

if heart\_rate > 120 and blood\_oxygen < 92 {

alert "EMERGENCY: Multiple critical vitals - immediate response required"

emit "code\_blue" to "hospital.emergency.system"

}

**Justification:** Healthcare monitoring demands immediate alerts when patient conditions deteriorate. Alert statements with conditional triggers ensure medical staff are notified of life-threatening situations without constant manual monitoring.

# 3. Complete Application Example

**Scenario:** Data Center Environmental Monitoring and Control System

This comprehensive example demonstrates how multiple statement types work together in a real-world IoT application.

// ===== DATA CENTER ENVIRONMENTAL CONTROL =====

// Configuration

var temp\_min = 18.0

var temp\_max = 27.0

var humidity\_min = 40.0

var humidity\_max = 60.0

// Read all environmental sensors

read rack\_temps from "datacenter.racks.temperature"

read humidity from "datacenter.humidity.sensor"

read power\_consumption from "datacenter.power.meters"

// Calculate statistics

var avg\_temp = avg(rack\_temps)

var max\_temp = max(rack\_temps)

var total\_power = sum(power\_consumption)

// Filter hot spots

filter rack\_temps where rack\_temps > temp\_max

// Temperature management

if max\_temp > temp\_max + 2 {

alert "CRITICAL: Temperature exceeds safe threshold"

emit "emergency\_cooling" to "hvac.datacenter.control"

emit max\_temp to "monitoring.critical.alerts"

} else {

if avg\_temp > temp\_max {

emit "increase\_cooling" to "hvac.datacenter.control"

}

}

// Humidity control

if humidity < humidity\_min {

emit "humidifier\_on" to "hvac.datacenter.humidity"

alert "Low humidity - humidifier activated"

}

if humidity > humidity\_max {

emit "dehumidifier\_on" to "hvac.datacenter.humidity"

alert "High humidity - dehumidifier activated"

}

// Process each hot rack individually

for temp in rack\_temps {

if temp > temp\_max {

alert "Hot rack detected: " + temp + "C"

emit temp to "database.thermal.hotspots"

}

}

// Power efficiency tracking

emit total\_power to "monitoring.power.consumption"

emit avg\_temp to "monitoring.thermal.average"

// Calculate PUE (Power Usage Effectiveness)

read it\_power from "datacenter.it.power"

var pue = total\_power / it\_power

emit pue to "monitoring.efficiency.pue"

**This example demonstrates:**

* **Variable declarations** for configuration and computed metrics
* **Read statements** acquiring data from multiple sensor types
* **Aggregations** calculating statistics over sensor arrays
* **Filtering** isolating problematic readings
* **Nested conditionals** implementing multi-level response logic
* **Loops** processing individual rack temperatures
* **Emit statements** controlling HVAC systems and logging data
* **Alerts** notifying operators of critical conditions
* **Transformations** deriving efficiency metrics

# 4. Summary

IoTScript provides a complete set of constructs specifically designed for IoT data processing workflows. Each statement type addresses a fundamental requirement in sensor-driven applications:

| **Statement** | **Primary Purpose** |
| --- | --- |
| **var** | Store configuration, thresholds, and computed values |
| **read** | Acquire data from sensors and external sources |
| **filter** | Focus on relevant data by eliminating normal readings |
| **transform** | Convert units, normalize values, apply calibrations |
| **Aggregations** | Calculate statistics for trend analysis and baselines |
| **if/else** | Implement decision logic based on conditions |
| **while/for** | Process collections and maintain monitoring loops |
| **emit** | Output data and control actuators/devices |
| **alert** | Notify operators of critical conditions |

The language's design prioritizes readability and domain-specific expressiveness, enabling rapid development of IoT data processing applications without sacrificing power or flexibility. By providing specialized constructs for common IoT patterns, IoTScript reduces development time while maintaining code clarity and maintainability.

*End of Document*

*IoT Data Processing Mini Language v1.0*