



APEX YAML Syntax Reference Guide

Version: 2.1 **Date:** 2025-08-30 **Author:** Mark Andrew Ray-Smith Cityline Ltd





 **CRITICAL SYNTAX UPDATE:** This document is currently being updated to reflect the correct APEX SpEL syntax. APEX processes HashMap data where fields are accessed using `#fieldName` syntax, NOT `#data.fieldName`. Many examples in this document still show the incorrect `#data.` prefix and are being corrected. Always use `#fieldName` for field access in APEX YAML configurations.

Table of Contents

1. [Introduction & Overview](#)
2. [Document Structure & Metadata](#)
3. [Core Syntax Elements](#)
4. [Rules Section](#)
5. [Rule Groups Section](#)  **NEW**
6. [Enrichments Section](#)
7. [Dataset Definitions](#)
8. [External Data-Source References](#)  **NEW**
9. [Pipeline Orchestration](#)  **NEW**
10. [Advanced Features](#)
11. [Best Practices](#)
12. [Common Patterns](#)
13. [Examples & Use Cases](#)
14. [Troubleshooting](#)
15. [Reference](#)
16. [Migration & Compatibility](#)

1. Introduction & Overview

What is APEX YAML

APEX YAML is a declarative configuration language for the APEX Rules Engine that enables business users and developers to define data validation rules, enrichment logic, and business processes without writing code. It combines the simplicity of YAML with the power of Spring Expression Language (SpEL) to create maintainable, testable business logic.

Key Principles

- **Declarative:** Describe what you want, not how to achieve it
- **Readable:** Business-friendly syntax that non-developers can understand
- **Powerful:** Full access to SpEL expressions and Java functionality
- **Maintainable:** Clear structure with separation of concerns
- **Testable:** Configuration can be validated and tested independently
- **Modular:** External data-source references enable clean architecture and reusable components

Design Philosophy

APEX YAML follows these core principles:

1. **Data-Driven:** All logic operates on HashMap data context using direct field references (`#fieldName`)
2. **Expression-Based:** Conditions and calculations use SpEL expressions
3. **Type-Safe:** Strong typing with automatic type conversion
4. **Null-Safe:** Built-in null safety with optional navigation operators
5. **Performance-Oriented:** Optimized for high-throughput processing

Relationship to Spring Expression Language (SpEL)

APEX YAML leverages SpEL for all expressions, providing:

- Mathematical operations and functions
- String manipulation and regex support
- Date/time operations
- Java class and method access
- Collection operations
- Conditional logic (ternary operators)

Document Structure Overview

Every APEX YAML document follows this structure:

```
metadata:
  # Document identification and configuration

data-source-refs: # Optional: External data-source references
  # References to external infrastructure configurations

pipeline: # Optional: Pipeline orchestration
  # Complete ETL/data processing workflows

rules:
  # Validation and business rules

enrichments:
  # Data enrichment logic

data-sources: # Optional: Inline data-source configurations
  # Direct data-source configurations (legacy approach)
```

Clean Architecture with External References

APEX 2.0 introduces **external data-source references** that enable clean separation of concerns:

- **Infrastructure Configuration:** External, reusable data-source configurations
- **Business Logic Configuration:** Lean, focused enrichment and validation rules
- **Configuration Caching:** External configurations cached for performance
- **Enterprise Scalability:** Shared infrastructure across multiple rule configurations

2. Document Structure & Metadata

Required Metadata Section

Every APEX YAML document must begin with a metadata section:

```
metadata:
  name: "Document Name"
  version: "1.0.0"
  description: "Document description"
  type: "rule-config"
  author: "author@company.com"
```

```
created-by: "author@company.com"
created-date: "2024-12-24"
domain: "Business Domain"
tags: ["tag1", "tag2", "tag3"]
```

Metadata Properties

| Property | Required | Description | Example |
|--------------|----------|------------------------------|------------------------------------|
| name | Yes | Human-readable document name | "Financial Settlement Rules" |
| version | Yes | Semantic version number | "1.2.3" |
| description | Yes | Brief description of purpose | "Post-trade settlement enrichment" |
| type | Yes | Document type identifier | "rule-config" |
| author | No | Document author | "john.doe@bank.com" |
| created-by | No | Creator identifier | "settlement-team@bank.com" |
| created-date | No | Creation date (ISO format) | "2024-12-24" |
| domain | No | Business domain | "Financial Services" |
| tags | No | Categorization tags | ["finance", "settlement"] |

Document-Level Configuration

Additional configuration options:

```
metadata:
  name: "Example Configuration"
  version: "1.0.0"
  type: "rule-config"

# Processing configuration
processing:
  parallel: true
  timeout: 30000 # milliseconds
  retry-count: 3

# Logging configuration
logging:
  level: "INFO"
  include-context: true

# Performance configuration
performance:
  cache-enabled: true
  cache-ttl: 3600 # seconds
```

Document Types

APEX supports several document types, each with specific purposes and validation requirements:

| Type | Purpose | Required Fields | Top-level Sections |
|----------------------|--|------------------------------|---|
| rule-config | Business rules and validation logic | author | rules , enrichments |
| enrichment | Data enrichment configurations | author | enrichments |
| dataset | Reference data and lookup tables | source | data |
| scenario | End-to-end processing scenarios | business-domain , owner | scenario , data-types , rule-configurations |
| scenario-registry | Scenario collection management | created-by | scenarios |
| bootstrap | Demo and initialization configurations | business-domain , created-by | bootstrap , data-sources |
| rule-chain | Sequential rule execution definitions | author | rule-chains |
| external-data-config | External data source configurations | author | dataSources , configuration |
| pipeline | ETL and data processing pipeline orchestration | author | pipeline , data-sources , data-sinks |

External Data Configuration

External data configuration files define how APEX connects to and interacts with external data sources such as databases, REST APIs, file systems, and message queues.

Example: Database Configuration

```

metadata:
  name: "Production Database Sources"
  version: "1.0.0"
  description: "Database connections for production environment"
  type: "external-data-config"
  author: "data.team@company.com"
  tags: ["database", "production", "postgresql"]

dataSources:
- name: "user-database"
  type: "database"
  sourceType: "postgresql"
  enabled: true
  description: "Primary user database"

connection:
  host: "prod-db.company.com"
  port: 5432
  database: "userdb"
  username: "app_user"
  password: "${DB_PASSWORD}"

queries:

```

```

    getUserById: "SELECT * FROM users WHERE id = :id"
    getActiveUsers: "SELECT * FROM users WHERE status = 'ACTIVE'"

cache:
  enabled: true
  ttlSeconds: 300
  maxSize: 1000

configuration:
  defaultConnectionTimeout: 30000
  monitoring:
    enabled: true
    healthCheckLogging: true

```

Example: REST API Configuration

```

metadata:
  name: "External API Sources"
  version: "1.0.0"
  description: "REST API connections for data enrichment"
  type: "external-data-config"
  author: "integration.team@company.com"

dataSources:
- name: "currency-rates-api"
  type: "rest-api"
  enabled: true
  description: "Real-time currency exchange rates"

connection:
  baseUrl: "https://api.exchangerates.com/v1"
  timeout: 5000

endpoints:
  getCurrentRate: "/rates/{currency}"
  getHistoricalRate: "/rates/{currency}/{date}"

authentication:
  type: "api-key"
  keyHeader: "X-API-Key"
  keyValue: "${EXCHANGE_API_KEY}"

```

Pipeline Configuration

Pipeline configuration files define complete ETL (Extract, Transform, Load) workflows that orchestrate data processing from multiple sources to multiple destinations.

Example: CSV to Database Pipeline

```

metadata:
  id: "csv-to-h2-pipeline-demo"
  name: "CSV to H2 ETL Pipeline Demo"
  version: "1.0.0"
  description: "Demonstration of CSV data processing with H2 database output"
  type: "pipeline"
  author: "APEX Demo Team"
  tags: ["demo", "etl", "csv", "h2", "pipeline"]

# Pipeline orchestration - defines the complete ETL workflow
pipeline:

```

```
name: "customer-etl-pipeline"
description: "Extract customer data from CSV, transform, and load into H2 database"
```

```
# Pipeline steps executed in sequence
```

```
steps:
```

- name: "extract-customers"
type: "extract"
source: "customer-csv-input"
operation: "getAllCustomers"
description: "Read all customer records from CSV file"
- name: "load-to-database"
type: "load"
sink: "customer-h2-database"
operation: "insertCustomer"
description: "Insert customer records into H2 database"
depends-on: ["extract-customers"]

```
# Pipeline execution configuration
```

```
execution:
```

```
mode: "sequential" # or "parallel" for independent steps
error-handling: "stop-on-error" # or "continue-on-error"
max-retries: 3
retry-delay-ms: 1000
```

```
# Input data source configuration
```

```
data-sources:
```

- name: "customer-csv-input"
type: "file-system"
source-type: "csv"
enabled: true
description: "Customer CSV file input for ETL processing"

```
connection:
```

```
base-path: "./target/demo/etl/data"
file-pattern: "customers.csv"
encoding: "UTF-8"
```

```
file-format:
```

```
type: "csv"
has-header-row: true
delimiter: ","
column-mappings:
  "customer_id": "id"
  "customer_name": "customerName"
  "email_address": "email"
```

```
# Output data sink configuration
```

```
data-sinks:
```

- name: "customer-h2-database"
type: "database"
source-type: "h2"
enabled: true
description: "H2 database for storing processed customer data"

```
connection:
```

```
database: "./target/demo/etl/output/customer_database"
username: "sa"
password: ""
mode: "PostgreSQL"
```

```
operations:
```

```
insertCustomer: |
  INSERT INTO customers (
    customer_id, customer_name, email, processed_at
  ) VALUES (
```

```

        :id, :customerName, :email, CURRENT_TIMESTAMP
    )

schema:
  auto-create: true
  table-name: "customers"
  init-script: |
    CREATE TABLE IF NOT EXISTS customers (
      customer_id INTEGER PRIMARY KEY,
      customer_name VARCHAR(255) NOT NULL,
      email VARCHAR(255),
      processed_at TIMESTAMP
    );

```

3. Core Syntax Elements

3.1 Data Access Patterns

Direct Field Access

All data access in APEX YAML uses direct field references to access HashMap data:

```

# Accessing top-level fields
condition: "#fieldName != null"

# Accessing nested fields (when data contains nested objects)
condition: "#trade.security.instrumentId != null"

# Using in calculations
expression: "#quantity * #price"

```

⚠ CRITICAL SYNTAX NOTE: APEX processes HashMap data structures where field names are accessed directly using `#fieldName` syntax. Do **NOT** use `#data.fieldName` syntax as this will cause SpEL evaluation errors. The correct pattern is always `#fieldName` for HashMap keys.

Nested Field Access with Dot Notation

Access nested objects using dot notation:

```

# Simple nesting
condition: "#customer.address.country == 'US'"

# Deep nesting
condition: "#trade.tradeHeader.partyTradeIdentifier.tradeId != null"

# Array/list access
condition: "#positions[0].instrumentId != null"

```

Null-Safe Navigation

Use the `?.` operator for null-safe navigation:


```
# Safe navigation - won't throw NullPointerException
condition: "#trade?.security?.instrumentId != null"

# Equivalent to checking each level for null
condition: "#trade != null && #trade.security != null && #trade.security.instrumentId != null"
```

Array and Collection Access

Access arrays and collections:

```
# Array index access
condition: "#positions[0].quantity > 0"

# Collection size
condition: "#positions.size() > 0"

# Collection operations
condition: "#positions.[quantity > 1000].size() > 0" # Filter collection
```

3.2 Condition Syntax

Boolean Expressions

Basic boolean logic:

```
# Simple boolean check
condition: "#isActive"

# Negation
condition: "!#isDeleted"

# Complex boolean logic
condition: "#isActive && !#isDeleted"
```

Comparison Operators

All standard comparison operators are supported:

```
# Equality
condition: "#status == 'ACTIVE'"

# Inequality
condition: "#quantity != 0"

# Numeric comparisons
condition: "#price > 100.0"
condition: "#quantity >= 1000"
condition: "#discount < 0.1"
condition: "#rating <= 5"
```

Logical Operators

Combine conditions with logical operators:

```

# AND operator
condition: "#isActive && #quantity > 0"

# OR operator
condition: "#status == 'PENDING' || #status == 'PROCESSING'"

# NOT operator
condition: "!#isDeleted && #isVisible"

# Complex combinations with parentheses
condition: "(#type == 'EQUITY' || #type == 'BOND') && #quantity > 0"

```

String Operations

String manipulation and comparison:

```

# String equality (case-sensitive)
condition: "#currency == 'USD'"

# String contains
condition: "#description.contains('SWAP')"

# String starts with / ends with
condition: "#instrumentId.startsWith('US')"
condition: "#instrumentId.endsWith('005')"

# String length
condition: "#instrumentId.length() == 12"

# Case-insensitive comparison
condition: "#currency.toUpperCase() == 'USD'"

```

Regular Expression Support

Use regex for pattern matching:

```

# ISIN format validation
condition: "#instrumentId.matches('^[A-Z]{2}[A-Z0-9]{9}[0-9]$')"

# Email validation
condition: "#email.matches('^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')"

# Phone number validation
condition: "#phone.matches('^\\+?[1-9]\\d{1,14}$')"

```

Null Checks and Validation

Proper null handling:

```

# Null check
condition: "#fieldName != null"

# Not null and not empty for strings
condition: "#fieldName != null && #fieldName.trim().length() > 0"

# Null-safe string operations

```

```
condition: "#{fieldName?.trim()?.length() > 0}"

# Default values for null fields
expression: "#{fieldName != null ? #fieldName : 'DEFAULT_VALUE'"}
```

3.3 Expression Language

SpEL Integration

APEX YAML provides full access to Spring Expression Language features:

```
# Variable assignment and reuse
expression: "#root.setVariable('tradeValue', #quantity * #price); #tradeValue"

# Method chaining
expression: "#instrumentId.substring(0, 2).toUpperCase()"

# Collection operations
expression: "#positions.[quantity * price].sum()"
```

Mathematical Operations

Standard mathematical operations:

```
# Basic arithmetic
expression: "#quantity * #price"
expression: "#total - #discount"
expression: "#principal + #interest"
expression: "#amount / #exchangeRate"
expression: "#base % #divisor"

# Mathematical functions via Java Math class
expression: "T(java.lang.Math).max(#value1, #value2)"
expression: "T(java.lang.Math).min(#value1, #value2)"
expression: "T(java.lang.Math).abs(#value)"
expression: "T(java.lang.Math).sqrt(#value)"
expression: "T(java.lang.Math).pow(#base, #exponent)"
expression: "T(java.lang.Math).round(#value * 100) / 100.0" # Round to 2 decimals
```

String Manipulation

String operations and formatting:

```
# String concatenation
expression: "#{firstName + ' ' + #lastName}"

# String formatting
expression: "T(java.lang.String).format('Trade %s: %,.2f %s', #tradeId, #amount, #currency)"

# String manipulation
expression: "#text.toUpperCase()"
expression: "#text.toLowerCase()"
expression: "#text.trim()"
expression: "#text.substring(0, 10)"
expression: "#text.replace('OLD', 'NEW')"
```

Date and Time Functions

Date/time operations using Java time classes:

```
# Current date/time
expression: "T(java.time.LocalDate).now()"
expression: "T(java.time.Instant).now().toString()"

# Date formatting
expression: "T(java.time.LocalDate).now().format(T(java.time.format.DateTimeFormatter).ofPattern('yyyyMMdd'))"

# Date arithmetic
expression: "#tradeDate.plusDays(2)" # Add 2 days
expression: "#startDate.plusMonths(1)" # Add 1 month
expression: "#endDate.minusYears(1)" # Subtract 1 year

# Date comparisons
condition: "#settlementDate.isAfter(T(java.time.LocalDate).now())"
condition: "#maturityDate.isBefore(#tradeDate.plusYears(10))"
```

Java Class Access

Access Java classes and static methods using `T()` syntax:

```
# UUID generation
expression: "T(java.util.UUID).randomUUID().toString()"

# BigDecimal operations
expression: "T(java.math.BigDecimal).valueOf(#amount).multiply(T(java.math.BigDecimal).valueOf(#rate))"

# Collections utilities
expression: "T(java.util.Collections).max(#values)"
expression: "T(java.util.Collections).min(#values)"

# Custom utility classes
expression: "T(com.company.utils.FinancialUtils).calculateInterest(#principal, #rate, #days)"
```

4. Rules Section

4.1 Validation Rules

Validation rules check data integrity and business constraints:

```
rules:
- id: "trade-id-required"
  name: "Trade ID Required"
  condition: "#trade != null && #trade.tradeId != null && #trade.tradeId.trim().length() > 0"
  message: "Trade ID is required and cannot be empty"
  severity: "ERROR"
  priority: 1

- id: "isin-format-validation"
  name: "ISIN Format Validation"
  condition: "#security != null && #security.isin != null && #security.isin.matches('^[A-Z]{2}[A-Z0-9]{9}[0-9]$')"
```

```

message: "ISIN must follow format: 2 country letters + 9 alphanumeric + 1 check digit"
severity: "ERROR"
priority: 1

- id: "trade-value-positive"
  name: "Trade Value Must Be Positive"
  condition: "#quantity != null && #price != null && (#quantity * #price) > 0"
  message: "Trade value must be positive"
  severity: "ERROR"
  priority: 1

```

Rule Properties

| Property | Required | Description | Example |
|-----------|----------|-----------------------------------|---------------------|
| id | Yes | Unique rule identifier | "trade-id-required" |
| name | Yes | Human-readable rule name | "Trade ID Required" |
| condition | Yes | SpEL expression that must be true | "#field != null" |
| message | Yes | Error/warning message | "Field is required" |
| severity | Yes | ERROR, WARNING, INFO | "ERROR" |
| priority | No | Execution priority (1 = highest) | 1 |

Severity Levels

- **ERROR:** Critical validation failure, stops processing
- **WARNING:** Non-critical issue, processing continues
- **INFO:** Informational message, no impact on processing

Complex Validation Examples

```

rules:
  # Multi-field validation
  - id: "settlement-date-validation"
    name: "Settlement Date Must Be After Trade Date"
    condition: "#tradeDate != null && #settlementDate != null && #settlementDate.isAfter(#tradeDate)"
    message: "Settlement date must be after trade date"
    severity: "ERROR"
    priority: 1

  # Conditional validation
  - id: "margin-required-for-derivatives"
    name: "Margin Required for Derivative Trades"
    condition: "#instrumentType != 'DERIVATIVE' || (#instrumentType == 'DERIVATIVE' && #marginAmount != null && #marginAm"
    message: "Margin amount is required for derivative trades"
    severity: "ERROR"
    priority: 2

  # Range validation
  - id: "credit-rating-range"
    name: "Credit Rating Must Be Valid"
    condition: "#creditRating == null || (#creditRating >= 1 && #creditRating <= 10)"
    message: "Credit rating must be between 1 and 10"
    severity: "WARNING"

```

```
priority: 3
```

4.2 Business Rules

Business rules implement domain-specific logic:

```
rules:
  # Business logic rule
  - id: "high-value-trade-approval"
    name: "High Value Trade Requires Approval"
    condition: "#tradeValue > 10000000" # $10M threshold
    message: "Trade exceeds $10M threshold and requires additional approval"
    severity: "WARNING"
    priority: 1

  # Regulatory compliance rule
  - id: "emir-reporting-required"
    name: "EMIR Reporting Required"
    condition: "#counterparty.jurisdiction == 'EU' && #notionalAmount > 1000000"
    message: "Trade requires EMIR reporting"
    severity: "INFO"
    priority: 2

  # Risk management rule
  - id: "concentration-limit-check"
    name: "Concentration Limit Check"
    condition: "#portfolioConcentration <= 0.25" # 25% limit
    message: "Position exceeds 25% concentration limit"
    severity: "ERROR"
    priority: 1
```

5. Rule Groups Section

5.1 Overview

Rule Groups allow you to organize related rules and apply logical operators (AND/OR) to combine their results. Rule Groups support advanced execution features including parallel processing, configurable short-circuiting, and debug mode for comprehensive testing and troubleshooting.

5.2 Basic Rule Group Configuration

```
rule-groups:
  - id: "validation-group"
    name: "Input Validation"
    description: "Validates all input parameters"
    category: "validation"
    priority: 10
    enabled: true
    operator: "AND" # "AND" or "OR" - how to combine rule results
    stop-on-first-failure: true # Enable/disable short-circuit evaluation
    parallel-execution: false # Execute rules in parallel when possible
    debug-mode: false # Enable debug logging and disable short-circuiting
    rule-ids:
      - "trade-id-required"
```

- "isin-format-validation"
- "trade-value-positive"

Rule Group Properties

| Property | Required | Default | Description | Example |
|-----------------------|----------|-----------|--|----------------------------------|
| id | Yes | - | Unique rule group identifier | "validation-group" |
| name | Yes | - | Human-readable group name | "Input Validation" |
| description | No | "" | Group description | "Validates all input parameters" |
| category | No | "default" | Group category | "validation" |
| priority | No | 100 | Execution priority (lower = higher priority) | 10 |
| enabled | No | true | Whether group is active | true |
| operator | No | "AND" | Logic operator: "AND" or "OR" | "AND" |
| stop-on-first-failure | No | false | Enable short-circuit evaluation | true |
| parallel-execution | No | false | Execute rules in parallel | false |
| debug-mode | No | false | Enable debug logging | false |
| rule-ids | Yes | - | List of rule IDs to include | ["rule1", "rule2"] |

5.3 Execution Behavior

AND Groups (All Rules Must Pass)

```
rule-groups:
- id: "strict-validation"
  name: "Strict Validation Group"
  description: "All validation rules must pass"
  operator: "AND"
  stop-on-first-failure: true # Stop on first failure for efficiency
  rule-ids:
    - "trade-id-required" # Must pass
    - "isin-format-validation" # Must pass
    - "trade-value-positive" # Must pass
```

Execution Flow (Short-Circuit Enabled):

```
Rule 1: PASS → Continue to Rule 2
Rule 2: PASS → Continue to Rule 3
Rule 3: FAIL → STOP (return false) - Remaining rules NOT evaluated
```

OR Groups (Any Rule Can Pass)

```
rule-groups:
- id: "eligibility-check"
  name: "Customer Eligibility Check"
  description: "Customer meets at least one eligibility criteria"
  operator: "OR"
  stop-on-first-failure: true # Stop on first success for OR groups
  rule-ids:
    - "high-value-customer" # Any can pass
    - "premium-member" # Any can pass
    - "long-term-client" # Any can pass
```

Execution Flow (Short-Circuit Enabled):

```
Rule 1: FAIL → Continue to Rule 2
Rule 2: FAIL → Continue to Rule 3
Rule 3: PASS → STOP (return true) - Remaining rules NOT evaluated
```

5.4 Advanced Execution Features

Short-Circuit Control

```
rule-groups:
# Production-optimized (short-circuit enabled)
- id: "production-validation"
  operator: "AND"
  stop-on-first-failure: true # Stop on first failure for performance
  debug-mode: false # Disable debug for performance
  rule-ids: ["rule1", "rule2", "rule3"]

# Complete evaluation (short-circuit disabled)
- id: "comprehensive-validation"
  operator: "AND"
  stop-on-first-failure: false # Evaluate all rules regardless of failures
  debug-mode: false # No debug logging
  rule-ids: ["rule1", "rule2", "rule3"]
```

Parallel Execution

```
rule-groups:
- id: "parallel-validation"
  name: "Parallel Rule Execution"
  description: "Execute CPU-intensive rules in parallel"
  operator: "AND"
  parallel-execution: true # Enable parallel processing
  stop-on-first-failure: false # Parallel execution disables short-circuiting
  rule-ids:
    - "complex-calculation-rule"
    - "external-api-validation"
    - "database-lookup-rule"
```

Parallel Execution Characteristics:

- **Thread Pool:** `min(rule_count, available_processors)`
- **Short-Circuiting:** Automatically disabled to ensure all rules complete

- **Error Handling:** Individual rule failures don't crash the group
- **Use Cases:** CPU-intensive rules, independent validations

Debug Mode

```
rule-groups:
- id: "debug-validation"
  name: "Debug Mode Validation"
  description: "Complete evaluation with debug logging"
  operator: "AND"
  debug-mode: true           # Enable debug logging
  stop-on-first-failure: false # Debug mode disables short-circuiting
  rule-ids:
    - "rule1"
    - "rule2"
    - "rule3"
```

Debug Output Example:

```
DEBUG: Rule 'trade-id-required' in group 'debug-validation' evaluated to: true
DEBUG: Rule 'isin-format-validation' in group 'debug-validation' evaluated to: false
DEBUG: Rule 'trade-value-positive' in group 'debug-validation' evaluated to: true
DEBUG: Group 'debug-validation' evaluation complete. Evaluated: 3, Passed: 2, Failed: 1, Final result: false
```

Debug Mode Configuration Options:

```
# Option 1: YAML configuration
debug-mode: true

# Option 2: System property (overrides YAML if not specified)
# -Dapex.rulegroup.debug=true
```

5.5 Configuration Scenarios

Production-Optimized Configuration

```
rule-groups:
- id: "production-group"
  name: "Production Validation"
  operator: "AND"
  stop-on-first-failure: true # Enable short-circuiting for performance
  parallel-execution: false  # Disable parallel for simplicity
  debug-mode: false          # Disable debug for performance
  rule-ids: ["critical-rule1", "critical-rule2"]
```

Debug-Optimized Configuration

```
rule-groups:
- id: "debug-group"
  name: "Debug Validation"
  operator: "AND"
  stop-on-first-failure: false # Disable short-circuiting for complete evaluation
  parallel-execution: false    # Disable parallel for deterministic debugging
```

```
debug-mode: true           # Enable debug logging
rule-ids: ["test-rule1", "test-rule2", "test-rule3"]
```

Performance-Optimized Configuration

```
rule-groups:
- id: "performance-group"
  name: "High-Performance Validation"
  operator: "OR"
  stop-on-first-failure: true # Stop on first success
  parallel-execution: true   # Use parallel processing
  debug-mode: false          # Disable debug for performance
  rule-ids: ["fast-rule1", "fast-rule2", "fast-rule3"]
```

5.6 Performance Comparison

| Configuration | Speed | Memory | CPU | Use Case |
|----------------------------|-----------|---------|--------|------------------------------|
| Short-Circuit + Sequential | Fastest | Lowest | Lowest | Production systems |
| Complete + Sequential | Slower | Medium | Medium | Debugging, reporting |
| Complete + Parallel | Variable* | Higher | Higher | CPU-intensive rules |
| Debug Mode | Slowest | Highest | Medium | Development, troubleshooting |

*Parallel execution speed depends on rule complexity and available CPU cores.

5.7 Best Practices

Performance Best Practices

```
rule-groups:
# Order rules by likelihood of failure (most likely to fail first)
- id: "optimized-validation"
  operator: "AND"
  stop-on-first-failure: true
  rule-ids:
    - "quick-null-check"      # Fast, likely to fail
    - "format-validation"     # Medium speed
    - "complex-business-rule" # Slow, unlikely to fail
```

Error Handling Best Practices

```
rule-groups:
# Separate critical and non-critical validations
- id: "critical-validation"
  name: "Critical Business Rules"
  operator: "AND"
  stop-on-first-failure: true
  rule-ids: ["mandatory-field-check", "regulatory-compliance"]

- id: "warning-validation"
  name: "Warning-Level Checks"
  operator: "OR"
```

```
stop-on-first-failure: false # Check all warnings
rule-ids: ["data-quality-warning", "business-recommendation"]
```

Testing Best Practices

```
rule-groups:
  # Use debug mode for comprehensive testing
  - id: "test-validation"
    name: "Test Environment Validation"
    operator: "AND"
    debug-mode: true          # Enable for testing
    stop-on-first-failure: false # See all test results
    rule-ids: ["test-rule1", "test-rule2", "test-rule3"]
```

6. Enrichments Section

6.1 Lookup Enrichments

Lookup enrichments add data by matching keys against datasets:

```
enrichments:
  - id: "lei-enrichment"
    type: "lookup-enrichment"
    condition: "#counterparty != null && #counterparty.name != null"
    lookup-config:
      lookup-key: "counterparty.name" # Field path (no # prefix in lookup-key)
      lookup-dataset:
        type: "inline"
        key-field: "name"
        data:
          - name: "Deutsche Bank AG"
            lei: "7LTFWZYICNSX8D621K86"
            jurisdiction: "DE"
            entityType: "BANK"
          - name: "JPMorgan Chase"
            lei: "8EE8DF3643E15DBFDA05"
            jurisdiction: "US"
            entityType: "BANK"
    field-mappings:
      - source-field: "lei"
        target-field: "counterparty.lei"
      - source-field: "jurisdiction"
        target-field: "counterparty.jurisdiction"
      - source-field: "entityType"
        target-field: "counterparty.entityType"
```

Lookup Enrichment Properties

| Property | Required | Description |
|----------|----------|------------------------------|
| id | Yes | Unique enrichment identifier |
| type | Yes | Must be "lookup-enrichment" |

| Property | Required | Description |
|----------------|----------|-------------------------------|
| condition | Yes | When to apply this enrichment |
| lookup-config | Yes | Lookup configuration |
| field-mappings | Yes | How to map lookup results |

Lookup Configuration

| Property | Required | Description |
|----------------|----------|---|
| lookup-key | Yes | Field path or expression for lookup key |
| lookup-dataset | Yes | Dataset definition |

Dynamic Lookup Keys

Use expressions for complex lookup keys:

```
lookup-config:
  lookup-key: "#counterparty.lei + '_' + #venue.country" # Composite key
  # or
  lookup-key: "#instrumentId.substring(0, 2)" # Derived key
```

5.2 Calculation Enrichments

Calculation enrichments derive new fields using expressions:

```
enrichments:
- id: "trade-value-calculation"
  type: "calculation-enrichment"
  condition: "#quantity != null && #price != null"
  calculations:
    - field: "tradeValue"
      expression: "#quantity * #price"
    - field: "tradeValueUSD"
      expression: "#currency == 'USD' ? #tradeValue : #tradeValue * #exchangeRate"
    - field: "commission"
      expression: "#tradeValue * 0.001" # 0.1% commission
    - field: "netAmount"
      expression: "#tradeValue + #commission"

- id: "risk-calculations"
  type: "calculation-enrichment"
  condition: "#tradeValue != null"
  calculations:
    - field: "var1Day"
      expression: "#tradeValue * 0.025" # 2.5% VaR
    - field: "var10Day"
      expression: "#var1Day * T(java.lang.Math).sqrt(10)"
    - field: "riskLevel"
      expression: "#var1Day > 1000000 ? 'HIGH' : (#var1Day > 100000 ? 'MEDIUM' : 'LOW')"
```

Calculation Properties

| Property | Required | Description |
|------------|----------|--|
| field | Yes | Target field name for the calculated value |
| expression | Yes | SpEL expression to calculate the value |

Complex Calculations

```

calculations:
  # Conditional calculations with ternary operators
  - field: "settlementPriority"
    expression: "#tradeValue > 100000000 ? 'HIGH' : (#tradeValue > 10000000 ? 'MEDIUM' : 'NORMAL')"

  # Multi-step calculations referencing previous calculations
  - field: "baseCommission"
    expression: "#tradeValue * #commissionRate"
  - field: "minimumCommission"
    expression: "25.0"
  - field: "finalCommission"
    expression: "T(java.lang.Math).max(#baseCommission, #minimumCommission)"

  # Date calculations
  - field: "settlementDate"
    expression: "#tradeDate.plusDays(#settlementCycle)"

  # String manipulations
  - field: "tradeReference"
    expression: "#counterpartyCode + '-' + #tradeId + '-' + T(java.time.LocalDate).now().format(T(java.time.format.DateTimeFormatter.ofPattern('dd-MM-yyyy')))"

```

7. Dataset Definitions

7.1 Inline Datasets

Inline datasets embed data directly in the configuration:

```

lookup-dataset:
  type: "inline"
  key-field: "instrumentId" # Field used for lookup matching
  data:
    - instrumentId: "GB00B03MLX29"
      name: "Royal Dutch Shell PLC"
      currency: "GBP"
      assetClass: "EQUITY"
      country: "GB"
    - instrumentId: "US0378331005"
      name: "Apple Inc"
      currency: "USD"
      assetClass: "EQUITY"
      country: "US"

```

Dataset Properties

| Property | Required | Description |
|-----------|----------|-------------------------------------|
| type | Yes | "inline" for embedded data |
| key-field | Yes | Field name used for lookup matching |
| data | Yes | Array of data objects |

Multi-Key Datasets

For composite keys, use expressions in the lookup-key:

```
lookup-config:
  lookup-key: "#lei + '_' + #country"
  lookup-dataset:
    type: "inline"
    key-field: "compositeKey"
  data:
    - compositeKey: "7LTWFZYICNSX8D621K86_GB"
      settlementMethod: "CREST"
      account: "CREST001234"
    - compositeKey: "7LTWFZYICNSX8D621K86_US"
      settlementMethod: "DTC"
      account: "DTC567890"
```

6.2 External Datasets

Reference external data sources:

```
lookup-dataset:
  type: "external"
  source: "reference-data-service"
  endpoint: "/api/securities"
  key-field: "isin"
  cache-ttl: 3600 # Cache for 1 hour
  timeout: 5000 # 5 second timeout
```

External Dataset Properties

| Property | Required | Description |
|-----------|----------|---------------------------------|
| type | Yes | "external" for external sources |
| source | Yes | Data source identifier |
| endpoint | No | API endpoint or query |
| key-field | Yes | Field used for lookup matching |
| cache-ttl | No | Cache time-to-live in seconds |
| timeout | No | Request timeout in milliseconds |

8. External Data-Source References

8.1 Overview

External Data-Source References are APEX 2.0's enterprise-grade solution for clean architecture and configuration management. This system enables **separation of concerns** by splitting configurations into:

- **Infrastructure Configuration:** External, reusable data-source configurations
- **Business Logic Configuration:** Lean, focused enrichment and validation rules

7.2 Benefits of External References

Clean Architecture

- **Separation of Concerns:** Infrastructure and business logic cleanly separated
- **Reusable Components:** External data-source configurations shared across multiple rule configurations
- **Maintainable Code:** Lean business logic configurations easy to understand and modify

Enterprise Scalability

- **Configuration Caching:** External configurations cached for performance
- **Connection Pooling:** Shared database connections across multiple enrichments
- **Environment Management:** Different infrastructure configurations for dev/test/prod

Production Readiness

- **Named Parameter Binding:** Enhanced database integration with parameter validation
- **Field Mapping Case Sensitivity:** Production-ready field handling
- **Error Handling:** Comprehensive error handling and fallback mechanisms

7.3 External Data-Source Reference Syntax

Basic Structure

```
metadata:
  name: "Business Logic Configuration"
  version: "2.0.0"
  description: "Lean configuration using external data-source references"

# External data-source references (infrastructure configuration - reusable)
data-source-refs:
  - name: "database-name"
    source: "data-sources/database-config.yaml"
    enabled: true
    description: "Reference to external database configuration"

# Business logic enrichments (lean and focused)
enrichments:
  - id: "enrichment-id"
    type: "lookup-enrichment"
    condition: "#field != null"
    lookup-config:
      lookup-key: "#field"
      lookup-dataset:
        type: "database"
        data-source-ref: "database-name" # References external data-source
```

query-ref: "namedQuery"

Named query from external config

External Data-Source Reference Properties

| Property | Required | Description | Example |
|-------------|----------|--|--|
| name | Yes | Unique identifier for the data-source reference | "postgresql-customer-database" |
| source | Yes | Path to external data-source configuration file | "data-sources/customer-db.yaml" |
| enabled | No | Whether this reference is active (default: true) | true |
| description | No | Human-readable description | "Customer database for profile enrichment" |

7.4 External Data-Source Configuration Files

External data-source configuration files contain infrastructure-specific settings:

Database Data-Source Configuration

```
# File: data-sources/postgresql-customer-database.yaml
metadata:
  name: "PostgreSQL Customer Database"
  version: "1.0.0"
  type: "external-data-config"
  description: "PostgreSQL customer database configuration"

# Database connection configuration
connection:
  type: "database"
  driver: "postgresql"
  # For PostgreSQL
  host: "localhost"
  port: 5432
  database: "customer_data"
  username: "postgres"
  password: "password"
  pool:
    initial-size: 5
    max-size: 20
    timeout: 30000

# H2 Database Configuration Examples
# =====

# File-based H2 (RECOMMENDED for demos)
h2-file-connection:
  type: "database"
  driver: "h2"
  # File-based H2 enables true database sharing between processes
  database: "./target/h2-demo/apex_demo_shared"
  username: "sa"
  password: ""

# In-memory H2 (NOT RECOMMENDED - creates isolated instances)
```



```

h2-memory-connection:
  type: "database"
  driver: "h2"
  # WARNING: Each connection creates a separate in-memory instance
  database: "shared_demo" # Becomes jdbc:h2:mem:shared_demo
  username: "sa"
  password: ""

# H2 TCP Server (for multi-process access)
h2-tcp-connection:
  type: "database"
  driver: "h2"
  host: "localhost"
  port: 9092
  database: "shared_demo"
  username: "sa"
  password: ""

# Enhanced H2 with Custom Parameters (NEW!)
h2-custom-connection:
  type: "database"
  driver: "h2"
  # Custom parameters can be specified after the database path
  # Format: "path/to/database;PARAM1=value1;PARAM2=value2"
  database: "./target/h2-demo/custom;MODE=MySQL;CACHE_SIZE=32768;TRACE_LEVEL_FILE=2"
  username: "sa"
  password: ""
  # This generates: jdbc:h2:./target/h2-demo/custom;MODE=MySQL;CACHE_SIZE=32768;TRACE_LEVEL_FILE=2;DB_CLOSE_DELAY=-1

# H2 In-memory with Custom Parameters
h2-memory-custom-connection:
  type: "database"
  driver: "h2"
  # In-memory database with custom parameters
  database: "mem:testdb;CACHE_SIZE=16384;MODE=Oracle;TRACE_LEVEL_SYSTEM_OUT=1"
  username: "sa"
  password: ""
  # This generates: jdbc:h2:mem:testdb;CACHE_SIZE=16384;MODE=Oracle;TRACE_LEVEL_SYSTEM_OUT=1;DB_CLOSE_DELAY=-1

# H2 Parameter Reference
# =====

# Common H2 Parameters for Performance Tuning:
# - MODE: Database compatibility mode (PostgreSQL, MySQL, Oracle, DB2, HSQLDB)
# - CACHE_SIZE: Database cache size in KB (default: 16384 = 16MB)
# - MAX_MEMORY_ROWS: Maximum rows kept in memory (default: 40000)
# - MAX_MEMORY_UNDO: Maximum undo log entries in memory (default: 50000)

# Common H2 Parameters for Debugging:
# - TRACE_LEVEL_FILE: SQL logging level to file (0=off, 1=error, 2=info, 4=debug)
# - TRACE_LEVEL_SYSTEM_OUT: SQL logging to console (0=off, 1=error, 2=info)
# - TRACE_MAX_FILE_SIZE: Maximum trace file size in MB (default: 16)

# Common H2 Parameters for Connection Management:
# - DB_CLOSE_DELAY: Keep database open after last connection (-1=forever, 0=immediate, >0=seconds)
# - DB_CLOSE_ON_EXIT: Close database when JVM exits (TRUE/FALSE)
# - AUTO_SERVER: Enable automatic mixed mode (TRUE/FALSE)

# Common H2 Parameters for Initialization:
# - INIT: SQL script to run on database startup
# - IFEXISTS: Only connect if database exists (TRUE/FALSE)
# - ACCESS_MODE_DATA: Database access mode (r=read-only, rw=read-write)

# Example Configurations:
performance-tuned-h2:
  database: "./target/h2-demo/performance;MODE=PostgreSQL;CACHE_SIZE=65536;MAX_MEMORY_ROWS=100000"

```

```

debug-enabled-h2:
  database: "./target/h2-demo/debug;TRACE_LEVEL_FILE=2;TRACE_LEVEL_SYSTEM_OUT=1;TRACE_MAX_FILE_SIZE=32"

mysql-compatible-h2:
  database: "./target/h2-demo/mysql;MODE=MySQL;CACHE_SIZE=32768"

read-only-h2:
  database: "./target/h2-demo/readonly;ACCESS_MODE_DATA=r;IFEXISTS=TRUE"

auto-init-h2:
  database: "./target/h2-demo/autoinit;INIT=RUNSCRIPT FROM 'classpath:schema.sql'"

# Named queries for reuse
queries:
  getActiveCustomerById:
    sql: |
      SELECT
        customer_id,
        customer_name,
        customer_type,
        tier,
        region,
        status,
        created_date
      FROM customers
      WHERE customer_id = :customerId
        AND status = 'ACTIVE'
    parameters:
      - name: "customerId"
        type: "string"
        required: true
        description: "Customer identifier"

# Connection health check
health-check:
  query: "SELECT 1"
  timeout: 5000
  interval: 30000

```

7.5 Using External References in Enrichments

Simple Database Lookup with External Reference

```

metadata:
  name: "Customer Profile Enrichment - External Reference"
  version: "2.1.0"
  description: "Customer profile enrichment using external data-source reference"

# External data-source references
data-source-refs:
  - name: "postgresql-customer-database"
    source: "data-sources/postgresql-customer-database.yaml"
    enabled: true

# Business logic enrichments
enrichments:
  - id: "customer-profile-lookup"
    type: "lookup-enrichment"
    description: "Customer profile enrichment using external data-source reference"
    condition: "#customerId != null && #customerId != ''"

    lookup-config:

```

```

lookup-key: "#customerId"
lookup-dataset:
  type: "database"
  data-source-ref: "postgresql-customer-database" # External reference
  query-ref: "getActiveCustomerById" # Named query
  parameters:
    - field: "customerId"
      type: "string"

# Field mappings from database columns to enriched object fields
field-mappings:
  - source-field: "CUSTOMER_NAME"
    target-field: "customerName"
    required: true
  - source-field: "CUSTOMER_TYPE"
    target-field: "customerType"
    required: true
  - source-field: "TIER"
    target-field: "customerTier"
    required: true

```

7.6 Advanced External Reference Patterns

Multiple External Data-Sources

```

metadata:
  name: "Multi-Source Transaction Processing"
  version: "2.0.0"
  description: "Transaction processing with multiple external data-sources"

# Multiple external data-source references
data-source-refs:
  - name: "customer-database"
    source: "data-sources/customer-database.yaml"
    enabled: true
  - name: "settlement-database"
    source: "data-sources/settlement-database.yaml"
    enabled: true
  - name: "market-data-api"
    source: "data-sources/market-data-api.yaml"
    enabled: true

# Business logic using multiple external sources
enrichments:
  - id: "customer-enrichment"
    type: "lookup-enrichment"
    lookup-config:
      lookup-dataset:
        data-source-ref: "customer-database"
        query-ref: "getCustomerProfile"

  - id: "settlement-enrichment"
    type: "lookup-enrichment"
    lookup-config:
      lookup-dataset:
        data-source-ref: "settlement-database"
        query-ref: "getSettlementInstructions"

  - id: "market-data-enrichment"
    type: "lookup-enrichment"
    lookup-config:
      lookup-dataset:
        data-source-ref: "market-data-api"

```

```
query-ref: "getCurrentPrice"
```

7.7 Configuration Caching and Performance

Automatic Configuration Caching

External data-source configurations are automatically cached for performance:

```
# External configurations are loaded once and cached
data-source-refs:
- name: "shared-database"
  source: "data-sources/shared-database.yaml" # Loaded once, cached
  enabled: true

# Multiple enrichments can reference the same external configuration
enrichments:
- id: "enrichment-1"
  lookup-config:
    lookup-dataset:
      data-source-ref: "shared-database" # Uses cached configuration

- id: "enrichment-2"
  lookup-config:
    lookup-dataset:
      data-source-ref: "shared-database" # Uses cached configuration
```

Performance Benefits

- **Configuration Loading:** External configurations loaded once and cached
- **Connection Pooling:** Database connections shared across enrichments
- **Query Preparation:** Named queries prepared once and reused
- **Memory Efficiency:** Reduced memory footprint through shared configurations

7.8 Field Mapping and Case Sensitivity

Production-Ready Field Mapping

External data-source references support case-sensitive field mapping for production environments:

```
enrichments:
- id: "database-lookup"
  type: "lookup-enrichment"
  lookup-config:
    lookup-dataset:
      data-source-ref: "postgresql-database"
      query-ref: "getRecord"

# Field mappings handle case sensitivity
field-mappings:
- source-field: "CUSTOMER_NAME" # Uppercase database column
  target-field: "customerName" # camelCase target field
  required: true
- source-field: "CUSTOMER_TYPE" # Uppercase database column
  target-field: "customerType" # camelCase target field
  required: true
```

7.9 Error Handling and Validation

External Reference Validation

APEX validates external data-source references at configuration load time:

```
data-source-refs:
- name: "invalid-reference"
  source: "non-existent-file.yaml" # ❌ Will cause validation error
  enabled: true

- name: "valid-reference"
  source: "data-sources/valid-config.yaml" # ✅ Will validate successfully
  enabled: true
```

Error Handling Patterns

```
enrichments:
- id: "resilient-lookup"
  type: "lookup-enrichment"
  condition: "#customerId != null"

lookup-config:
  lookup-dataset:
    data-source-ref: "customer-database"
    query-ref: "getCustomer"

# Error handling configuration
error-handling:
  on-error: "continue" # Continue processing on error
  fallback-value: null # Default value on lookup failure
  log-errors: true # Log errors for monitoring
```

8.5 Data Sinks (Output Destinations)

Data Sinks provide output capabilities for APEX, enabling processed data to be written to various destinations including databases, files, message queues, and REST APIs. This complements the existing data-sources functionality by providing a complete data pipeline solution.

Overview

Data sinks follow the same architectural patterns as data sources, supporting:

- **Multiple Output Types:** Database, file system, message queue, REST API, cache
- **Batch Processing:** Efficient bulk operations with configurable batch sizes
- **Error Handling:** Comprehensive retry mechanisms and dead letter queues
- **Schema Management:** Auto-creation and validation of database schemas
- **Format Support:** JSON, CSV, XML, SQL, and custom formats

Basic Data Sink Configuration

```
metadata:
  name: "Data Pipeline with Output"
  version: "1.0.0"
  description: "Complete data pipeline with input and output"
```

```

data-sinks:
- name: "customer-database-sink"
  type: "database"
  source-type: "h2"
  enabled: true
  description: "H2 database for processed customer data"

connection:
  database: "./target/output/customer_data"
  username: "sa"
  password: ""
  mode: "PostgreSQL"

operations:
  insertCustomer: "INSERT INTO customers (id, name, email, processed_at) VALUES (:id, :name, :email, :processedAt)"
  updateCustomer: "UPDATE customers SET name = :name, email = :email WHERE id = :id"
  upsertCustomer: "MERGE INTO customers (id, name, email, processed_at) KEY (id) VALUES (:id, :name, :email, :process

schema:
  auto-create: true
  table-name: "customers"
  init-script: |
    CREATE TABLE IF NOT EXISTS customers (
      id INTEGER PRIMARY KEY,
      name VARCHAR(255) NOT NULL,
      email VARCHAR(255),
      processed_at TIMESTAMP
    );

error-handling:
  strategy: "log-and-continue"
  max-retries: 3
  retry-delay: 1000
  dead-letter-table: "failed_records"

batch:
  enabled: true
  batch-size: 50
  timeout-ms: 10000
  transaction-mode: "per-batch"

```

File System Data Sink

```

data-sinks:
- name: "audit-file-sink"
  type: "file-system"
  source-type: "json"
  enabled: true
  description: "Audit trail file output"

connection:
  base-path: "./target/output/audit"
  file-pattern: "audit_{timestamp}.json"
  encoding: "UTF-8"

operations:
  writeAuditRecord: "WRITE_JSON"
  appendAuditRecord: "APPEND_JSON"

output-format:
  format: "json"
  pretty-print: true

```

```
encoding: "UTF-8"
include-timestamp: true

batch:
  enabled: true
  batch-size: 100
  flush-interval-ms: 5000
```

Data Sink Properties

| Property | Required | Description | Example |
|----------------|----------|---|--|
| name | Yes | Unique identifier for the data sink | "customer-database-sink" |
| type | Yes | Type of data sink | "database", "file-system", "message-queue" |
| source-type | No | Specific implementation type | "h2", "postgresql", "csv", "json" |
| enabled | No | Whether this sink is active (default: true) | true |
| description | No | Human-readable description | "Customer data output sink" |
| connection | Yes | Connection configuration | See connection examples |
| operations | Yes | Named operations (SQL, templates, etc.) | See operations examples |
| schema | No | Schema management configuration | See schema examples |
| error-handling | No | Error handling strategy | See error handling examples |
| batch | No | Batch processing configuration | See batch examples |
| output-format | No | Output format settings | See format examples |

Error Handling Strategies

| Strategy | Description | Use Case |
|--------------------|--|--------------------------------------|
| fail-fast | Stop processing on first error | Critical data integrity requirements |
| log-and-continue | Log error and continue processing | Best effort processing |
| dead-letter | Send failed records to dead letter queue | Error recovery and analysis |
| retry-and-fail | Retry failed operations, then fail | Transient error handling |
| retry-and-continue | Retry failed operations, then continue | Resilient processing |

Complete Pipeline Example

```
metadata:
  name: "CSV to Database Pipeline"
```

```

version: "1.0.0"
description: "Complete pipeline from CSV input to database output"

# Input data source
data-source-refs:
- name: "customer-csv-input"
  source: "data-sources/customer-csv.yaml"
  enabled: true

# Data transformation
enrichments:
- id: "customer-data-enrichment"
  type: "field-transformation"
  description: "Enrich and validate customer data"
  condition: "true"

  calculations:
  - field: "processedAt"
    expression: "new java.util.Date()"
  - field: "status"
    expression: "'PROCESSED'"

# Output data sink
data-sinks:
- name: "customer-h2-output"
  type: "database"
  source-type: "h2"
  enabled: true

  connection:
    database: "./target/output/processed_customers"
    username: "sa"
    password: ""

  operations:
    insertCustomer: "INSERT INTO customers (id, name, email, processed_at, status) VALUES (:id, :name, :email, :process

  schema:
    auto-create: true
    table-name: "customers"

  batch:
    enabled: true
    batch-size: 100

```

Pipeline Orchestration

9.1 Overview

Pipeline Orchestration is APEX's revolutionary approach to YAML-driven data processing workflows. This system embodies the core APEX principle that **all processing logic should be contained in the YAML configuration file**, eliminating hardcoded orchestration in Java code.

Key Benefits

- **YAML-Driven Processing:** Complete pipeline workflows defined in YAML
- **Dependency Management:** Automatic step dependency resolution and validation
- **Error Handling:** Configurable error handling strategies with optional steps

- **Data Flow:** Automatic data passing between pipeline steps
- **Monitoring:** Built-in step timing and execution tracking
- **Validation:** Pipeline configuration validation with circular dependency detection

Core Principle

Before (Hardcoded Java):

```
pipelineEngine.execute("getAllCustomers", "customer-csv-input",
    "customer-h2-database", "insertCustomer");
```

After (YAML-Driven):

```
pipelineEngine.executePipeline("customer-etl-pipeline");
```

9.2 Pipeline Configuration Structure

Basic Pipeline Syntax

```
pipeline:
  name: "pipeline-name"
  description: "Pipeline description"

  steps:
    - name: "step-name"
      type: "step-type"
      # Step-specific configuration

  execution:
    mode: "sequential" # or "parallel"
    error-handling: "stop-on-error" # or "continue-on-error"

  monitoring:
    enabled: true
    log-progress: true
```

Complete Pipeline Example

```
metadata:
  name: "CSV to H2 ETL Pipeline Demo"
  version: "1.0.0"
  description: "Complete ETL pipeline using APEX orchestration"

# Pipeline orchestration - defines the complete ETL workflow
pipeline:
  name: "customer-etl-pipeline"
  description: "Extract customer data from CSV, transform, and load into H2 database"

# Pipeline steps executed in sequence
steps:
  - name: "extract-customers"
    type: "extract"
    source: "customer-csv-input"
    operation: "getAllCustomers"
    description: "Read all customer records from CSV file"
```

```

- name: "load-to-database"
  type: "load"
  sink: "customer-h2-database"
  operation: "insertCustomer"
  description: "Insert customer records into H2 database"
  depends-on: ["extract-customers"]

- name: "audit-logging"
  type: "audit"
  sink: "audit-log-file"
  operation: "writeAuditRecord"
  description: "Write audit records to JSON file"
  depends-on: ["load-to-database"]
  optional: true

# Pipeline execution configuration
execution:
  mode: "sequential"
  error-handling: "stop-on-error"
  max-retries: 3
  retry-delay-ms: 1000

# Pipeline monitoring and metrics
monitoring:
  enabled: true
  log-progress: true
  collect-metrics: true
  alert-on-failure: true

# Data sources and sinks referenced by pipeline steps
data-sources:
- name: "customer-csv-input"
  type: "file-system"
  # ... data source configuration

data-sinks:
- name: "customer-h2-database"
  type: "database"
  # ... database sink configuration

- name: "audit-log-file"
  type: "file-system"
  # ... file sink configuration

```

9.3 Pipeline Steps

Step Types

| Type | Purpose | Required Fields | Description |
|-----------|---------------------|--------------------|---------------------------------|
| extract | Data extraction | source , operation | Read data from external sources |
| load | Data loading | sink , operation | Write data to external sinks |
| transform | Data transformation | transformation | Transform data between steps |
| audit | Audit logging | sink , operation | Write audit records |

Extract Steps

Extract steps read data from external data sources:

```
steps:
- name: "extract-customers"
  type: "extract"
  source: "customer-csv-input" # Data source name
  operation: "getAllCustomers" # Named query/operation
  description: "Read customer data from CSV"
  parameters:
    limit: 1000
    offset: 0
```

Load Steps

Load steps write data to external data sinks:

```
steps:
- name: "load-to-database"
  type: "load"
  sink: "customer-h2-database" # Data sink name
  operation: "insertCustomer" # Named operation
  description: "Insert customers into database"
  depends-on: ["extract-customers"]
  parameters:
    batch-size: 100
    upsert: true
```

Transform Steps

Transform steps modify data between extraction and loading:

```
steps:
- name: "transform-data"
  type: "transform"
  description: "Apply business transformations"
  depends-on: ["extract-customers"]
  transformations:
    - name: "add-processing-timestamp"
      type: "field-addition"
      field: "processed_at"
      value: "CURRENT_TIMESTAMP"

    - name: "validate-email"
      type: "validation"
      field: "email"
      rule: "email-format"
```

Audit Steps

Audit steps create audit trails and logging:

```
steps:
- name: "audit-logging"
  type: "audit"
  sink: "audit-log-file"
  operation: "writeAuditRecord"
```

```
description: "Create audit trail"
depends-on: ["load-to-database"]
optional: true # Won't fail pipeline if it fails
```

9.4 Step Dependencies

Dependency Declaration

Steps can declare dependencies on other steps:

```
steps:
- name: "step-a"
  type: "extract"
  # ... configuration

- name: "step-b"
  type: "transform"
  depends-on: ["step-a"] # Wait for step-a to complete

- name: "step-c"
  type: "load"
  depends-on: ["step-a", "step-b"] # Wait for both steps
```

Dependency Validation

APEX automatically validates dependencies:

- **Circular Dependency Detection:** Prevents infinite loops
- **Missing Dependency Validation:** Ensures all referenced steps exist
- **Topological Sorting:** Executes steps in correct dependency order

9.5 Error Handling

Pipeline-Level Error Handling

```
pipeline:
  execution:
    error-handling: "stop-on-error" # Stop pipeline on any error
    # OR
    error-handling: "continue-on-error" # Continue with remaining steps
    max-retries: 3
    retry-delay-ms: 1000
```

Step-Level Error Handling

```
steps:
- name: "optional-step"
  type: "audit"
  optional: true # Pipeline continues if this step fails
  retry:
    max-attempts: 3
    delay-ms: 1000
    backoff-multiplier: 2.0
```

9.6 Data Flow

Automatic Data Passing

Data flows automatically between pipeline steps:

1. **Extract Step** → Stores data in pipeline context
2. **Transform Step** → Reads from context, transforms, stores result
3. **Load Step** → Reads transformed data, writes to sink
4. **Audit Step** → Reads original/transformed data for auditing

Data Context

```
# Data automatically available in pipeline context:
# - extractedData: Raw data from extract steps
# - transformedData: Processed data from transform steps
# - stepResults: Results from each completed step
```

9.7 Monitoring and Metrics

Built-in Monitoring

```
pipeline:
  monitoring:
    enabled: true
    log-progress: true      # Log step start/completion
    collect-metrics: true   # Collect timing metrics
    alert-on-failure: true  # Alert on pipeline failures
```

Execution Results

Pipeline execution provides detailed results:

```
YamlPipelineExecutionResult result = pipelineEngine.executePipeline("pipeline-name");

// Overall pipeline status
boolean success = result.isSuccess();
long duration = result.getDurationMs();
int totalSteps = result.getTotalSteps();

// Individual step results
for (PipelineStepResult stepResult : result.getStepResults()) {
    String stepName = stepResult.getStepName();
    boolean stepSuccess = stepResult.isSuccess();
    long stepDuration = stepResult.getDurationMs();
}
```

10. Advanced Features

9.1 Conditional Logic

Ternary Operators

Use ternary operators for conditional expressions:

```
# Basic ternary
expression: "#condition ? 'value1' : 'value2'"

# Nested ternary for multiple conditions
expression: "#score >= 90 ? 'A' : (#score >= 80 ? 'B' : (#score >= 70 ? 'C' : 'F'))"

# Complex conditions
expression: "#type == 'EQUITY' && #quantity > 1000 ? 'LARGE_EQUITY' : 'OTHER'"

# Null-safe ternary
expression: "#field != null ? #field : 'DEFAULT'"
```

Complex Branching

Handle multiple conditions efficiently:

```
calculations:
- field: "riskCategory"
  expression: |
    #assetClass == 'EQUITY' ?
      (#marketCap > 10000000000 ? 'LARGE_CAP_EQUITY' : 'SMALL_CAP_EQUITY') :
    #assetClass == 'BOND' ?
      (#creditRating.startsWith('AA') ? 'HIGH_GRADE_BOND' : 'INVESTMENT_GRADE_BOND') :
    #assetClass == 'DERIVATIVE' ?
      'DERIVATIVE' :
    'OTHER'
```

Performance Optimization

Optimize conditions for better performance:

```
# Good: Check simple conditions first
condition: "#isActive && #complexCalculation() > threshold"

# Better: Use short-circuit evaluation
condition: "#isActive && (#value != null && #value > 0) && #complexCalculation() > threshold"

# Best: Cache expensive calculations
calculations:
- field: "expensiveResult"
  expression: "#complexCalculation()"
- field: "finalResult"
  expression: "#isActive && #expensiveResult > threshold"
```

8.2 Function Usage

Built-in Functions

APEX provides access to standard Java functions:

```

# String functions
expression: "#text.toUpperCase()"
expression: "#text.substring(0, 10)"
expression: "#text.matches('[A-Z]{2}[0-9]{10}')"

# Math functions
expression: "T(java.lang.Math).max(#value1, #value2)"
expression: "T(java.lang.Math).round(#value * 100) / 100.0"

# Date functions
expression: "T(java.time.LocalDate).now().plusDays(2)"
expression: "#date.format(T(java.time.format.DateTimeFormatter).ofPattern('yyyy-MM-dd'))"

# Collection functions
expression: "#list.size()"
expression: "#list.contains('value')"
expression: "#list.[field > 100].size()" # Filter and count

```

Custom Function Integration

Access custom utility classes:

```

# Custom financial calculations
expression: "T(com.company.utils.FinancialUtils).calculateYield(#price, #coupon, #maturity)"

# Custom validation functions
condition: "T(com.company.validators.ISINValidator).isValid(#isin)"

# Custom formatting functions
expression: "T(com.company.formatters.CurrencyFormatter).format(#amount, #currency)"

```

Error Handling in Functions

Handle potential errors gracefully:

```

# Safe division
expression: "#denominator != 0 ? #numerator / #denominator : 0"

# Safe string operations
expression: "#text != null && #text.length() > 10 ? #text.substring(0, 10) : #text"

# Try-catch equivalent using ternary
expression: "#value != null && #value.matches('[0-9]+') ? T(java.lang.Integer).parseInt(#value) : 0"

```

10. Best Practices

10.1 Performance Guidelines

Condition Optimization

Write efficient conditions:

```

# Good: Simple conditions first
condition: "#isActive && #expensiveCheck()"

# Better: Use null checks to avoid expensive operations
condition: "#field != null && #field.expensiveOperation() > 0"

# Best: Cache results of expensive operations
calculations:
  - field: "cachedResult"
    expression: "#expensiveOperation()"
  - field: "finalCheck"
    expression: "#isActive && #cachedResult > threshold"

```

Dataset Sizing

Optimize dataset performance:

```

# Good: Small inline datasets (< 100 records)
lookup-dataset:
  type: "inline"
  key-field: "code"
  data:
    - code: "USD"
      name: "US Dollar"
    # ... < 100 records

# Better: Use external datasets for large data
lookup-dataset:
  type: "external"
  source: "reference-data-service"
  cache-ttl: 3600 # Cache for performance

```

Expression Efficiency

Write efficient expressions:

```

# Avoid: Repeated expensive calculations
expression: "#complexCalc() + #complexCalc() * 0.1"

# Better: Calculate once and reuse
calculations:
  - field: "baseValue"
    expression: "#complexCalc()"
  - field: "finalValue"
    expression: "#baseValue + #baseValue * 0.1"

```

8.2 Maintainability

Naming Conventions

Use consistent, descriptive names:

```

# Good naming conventions
rules:
  - id: "trade-id-required" # kebab-case for IDs
    name: "Trade ID Required" # Title Case for names

```



```

enrichments:
- id: "lei-enrichment"           # descriptive, specific
  field: "counterparty.lei"      # clear field paths

calculations:
- field: "tradeValueUSD"         # camelCase for calculated fields
  expression: "#quantity * #price"

```

Documentation Standards

Document complex logic:

```

enrichments:
- id: "complex-risk-calculation"
  type: "calculation-enrichment"
  # Purpose: Calculate portfolio risk metrics according to Basel III requirements
  # Input: position data with market values and volatilities
  # Output: VaR, expected shortfall, and risk-weighted assets
  condition: "#positions != null && #positions.size() > 0"
  calculations:
    # Calculate 1-day VaR at 99% confidence level
    - field: "var1Day99"
      expression: "#portfolioValue * 0.025" # 2.5% VaR multiplier

    # Scale to 10-day VaR using square root of time rule
    - field: "var10Day99"
      expression: "#var1Day99 * T(java.lang.Math).sqrt(10)"

```

8.3 Error Handling

Graceful Degradation

Handle missing or invalid data gracefully:

```

# Provide defaults for missing data
calculations:
- field: "effectiveRate"
  expression: "#customRate != null ? #customRate : #standardRate"

- field: "safeCalculation"
  expression: "#denominator != null && #denominator != 0 ? #numerator / #denominator : 0"

```

Null Safety

Always check for null values:

```

# Safe navigation
condition: "#trade?.security?.instrumentId != null"

# Explicit null checks
condition: "#trade != null && #trade.security != null && #trade.security.instrumentId != null"

# Safe string operations
expression: "#text != null && #text.trim().length() > 0 ? #text.toUpperCase() : 'UNKNOWN'"

```

11. Common Patterns

11.1 Financial Services Patterns

Reference Data Enrichment Pattern

Standard pattern for enriching with reference data:

```
enrichments:
- id: "security-master-enrichment"
  type: "lookup-enrichment"
  condition: "#instrumentId != null"
  lookup-config:
    lookup-key: "instrumentId"
    lookup-dataset:
      type: "external"
      source: "security-master"
      key-field: "isin"
  field-mappings:
    - source-field: "name"
      target-field: "security.name"
    - source-field: "assetClass"
      target-field: "security.assetClass"
    - source-field: "currency"
      target-field: "security.currency"
```

Risk Calculation Pattern

Standard risk metrics calculation:

```
enrichments:
- id: "risk-metrics"
  type: "calculation-enrichment"
  condition: "#marketValue != null"
  calculations:
    # Value at Risk calculations
    - field: "var1Day95"
      expression: "#marketValue * 0.0164" # 1.64 * volatility
    - field: "var1Day99"
      expression: "#marketValue * 0.0233" # 2.33 * volatility
    - field: "var10Day99"
      expression: "#var1Day99 * T(java.lang.Math).sqrt(10)"

    # Risk classification
    - field: "riskLevel"
      expression: "#var1Day99 > 1000000 ? 'HIGH' : (#var1Day99 > 100000 ? 'MEDIUM' : 'LOW')"
```

Regulatory Compliance Pattern

Standard regulatory field generation:

```
enrichments:
- id: "regulatory-fields"
  type: "calculation-enrichment"
  calculations:
```

```

# UTI generation
- field: "regulatory.uti"
  expression: "#reportingEntity.lei + '-' + #tradeId + '-' + T(java.time.LocalDate).now().format(T(java.time.format

# Jurisdiction flags
- field: "regulatory.emirApplicable"
  expression: "#counterparty.jurisdiction == 'EU'"
- field: "regulatory.mifidApplicable"
  expression: "#venue.country == 'GB' || #venue.country == 'DE' || #venue.country == 'FR'"

```

9.2 Data Validation Patterns

Format Validation Pattern

Standard format validation approach:

```

rules:
- id: "isin-format"
  name: "ISIN Format Validation"
  condition: "#isin == null || #isin.matches('[A-Z]{2}[A-Z0-9]{9}[0-9]$')"
  message: "ISIN must be 12 characters: 2 letters + 9 alphanumeric + 1 digit"
  severity: "ERROR"

- id: "lei-format"
  name: "LEI Format Validation"
  condition: "#lei == null || #lei.matches('[A-Z0-9]{18}[0-9]{2}$')"
  message: "LEI must be 20 characters: 18 alphanumeric + 2 check digits"
  severity: "ERROR"

```

Business Rule Validation Pattern

Standard business rule validation:

```

rules:
- id: "settlement-date-business-rule"
  name: "Settlement Date Must Be Business Day"
  condition: "#settlementDate == null || T(com.company.utils.BusinessDayUtils).isBusinessDay(#settlementDate, #market.c
  message: "Settlement date must be a business day in the market country"
  severity: "ERROR"

- id: "trade-limit-check"
  name: "Trade Limit Validation"
  condition: "#tradeValue <= #counterparty.creditLimit"
  message: "Trade value exceeds counterparty credit limit"
  severity: "ERROR"

```

Cross-Field Validation Pattern

Validate relationships between fields:

```

rules:
- id: "settlement-after-trade-date"
  name: "Settlement Date After Trade Date"
  condition: "#tradeDate == null || #settlementDate == null || #settlementDate.isAfter(#tradeDate)"
  message: "Settlement date must be after trade date"

```

```
severity: "ERROR"

- id: "currency-consistency"
  name: "Currency Consistency Check"
  condition: "#security.currency == null || #trade.currency == null || #security.currency == #trade.currency"
  message: "Security currency must match trade currency"
  severity: "WARNING"
```

12. Examples & Use Cases

12.1 Simple Examples

Basic Lookup Example

Simple counterparty LEI lookup:

```
metadata:
  name: "Simple LEI Lookup"
  version: "1.0.0"
  type: "rule-config"

enrichments:
- id: "lei-lookup"
  type: "lookup-enrichment"
  condition: "#counterpartyName != null"
  lookup-config:
    lookup-key: "counterpartyName"
    lookup-dataset:
      type: "inline"
      key-field: "name"
      data:
        - name: "Deutsche Bank AG"
          lei: "7LTWFZYICNSX8D621K86"
        - name: "JPMorgan Chase"
          lei: "8EE8DF3643E15DBFDA05"
  field-mappings:
    - source-field: "lei"
      target-field: "counterpartyLEI"
```

Basic Calculation Example

Simple trade value calculation:

```
metadata:
  name: "Trade Value Calculation"
  version: "1.0.0"
  type: "rule-config"

enrichments:
- id: "trade-value"
  type: "calculation-enrichment"
  condition: "#quantity != null && #price != null"
  calculations:
    - field: "tradeValue"
      expression: "#quantity * #price"
    - field: "commission"
```

```
    expression: "#tradeValue * 0.001" # 0.1% commission
- field: "netAmount"
  expression: "#tradeValue + #commission"
```

Basic Validation Example

Simple field validation:

```
metadata:
  name: "Basic Validation"
  version: "1.0.0"
  type: "rule-config"

rules:
- id: "required-fields"
  name: "Required Fields Validation"
  condition: "#tradeId != null && #counterpartyName != null && #instrumentId != null"
  message: "Trade ID, counterparty name, and instrument ID are required"
  severity: "ERROR"
  priority: 1
```

10.2 Complex Examples

Multi-Step Enrichment Example

Complex enrichment with multiple dependencies:

```
metadata:
  name: "Complex Settlement Enrichment"
  version: "1.0.0"
  type: "rule-config"

enrichments:
# Step 1: Enrich counterparty data
- id: "counterparty-enrichment"
  type: "lookup-enrichment"
  condition: "#counterpartyName != null"
  lookup-config:
    lookup-key: "counterpartyName"
    lookup-dataset:
      type: "inline"
      key-field: "name"
      data:
        - name: "Deutsche Bank AG"
          lei: "7LTWFZYICNSX8D621K86"
          jurisdiction: "DE"
          creditRating: "A1"
  field-mappings:
    - source-field: "lei"
      target-field: "counterparty.lei"
    - source-field: "jurisdiction"
      target-field: "counterparty.jurisdiction"
    - source-field: "creditRating"
      target-field: "counterparty.creditRating"

# Step 2: Calculate trade metrics
- id: "trade-calculations"
  type: "calculation-enrichment"
  condition: "#quantity != null && #price != null"
```

```

calculations:
- field: "tradeValue"
  expression: "#quantity * #price"
- field: "tradeValueUSD"
  expression: "#currency == 'USD' ? #tradeValue : #tradeValue * #fxRate"

# Step 3: Determine settlement instructions based on enriched data
- id: "settlement-instructions"
  type: "lookup-enrichment"
  condition: "#counterparty.lei != null && #venue.country != null"
  lookup-config:
    lookup-key: "#counterparty.lei + '_' + #venue.country"
    lookup-dataset:
      type: "inline"
      key-field: "key"
      data:
        - key: "7LTFZYICNSX8D621K86_GB"
          method: "CREST"
          account: "CREST001234"
        - key: "7LTFZYICNSX8D621K86_US"
          method: "DTC"
          account: "DTC567890"
  field-mappings:
    - source-field: "method"
      target-field: "settlement.method"
    - source-field: "account"
      target-field: "settlement.account"

# Step 4: Calculate fees based on trade value and counterparty rating
- id: "fee-calculations"
  type: "calculation-enrichment"
  condition: "#tradeValueUSD != null && #counterparty.creditRating != null"
  calculations:
    - field: "commissionRate"
      expression: "#counterparty.creditRating.startsWith('A') ? 0.0005 : 0.001" # Premium rate for A-rated
    - field: "commission"
      expression: "#tradeValueUSD * #commissionRate"
    - field: "clearingFee"
      expression: "#tradeValueUSD * 0.0001" # 1 bp clearing fee
    - field: "totalFees"
      expression: "#commission + #clearingFee"
    - field: "netSettlementAmount"
      expression: "#tradeValueUSD + #totalFees"

```

13. Troubleshooting

13.1 Common Errors

Syntax Errors

Missing field reference prefix:

```

# Wrong - no # prefix
condition: "quantity > 0"

# Correct - direct field reference
condition: "#quantity > 0"

```

Incorrect field access:

```
# Wrong - using # prefix in lookup-key
lookup-key: "#counterparty.name"

# Correct - no # prefix in lookup-key
lookup-key: "counterparty.name"
```

Invalid SpEL syntax:

```
# Wrong - invalid operator
condition: "#value = 100"

# Correct - use == for comparison
condition: "#value == 100"
```

Runtime Errors

NullPointerException:

```
# Problematic - can throw NPE
expression: "#trade.security.instrumentId.substring(0, 2)"

# Safe - use null checks
expression: "#trade?.security?.instrumentId != null ? #trade.security.instrumentId.substring(0, 2) : null"
```

Type conversion errors:

```
# Problematic - string to number conversion
expression: "#stringValue + 100"

# Safe - explicit conversion with validation
expression: "#stringValue != null && #stringValue.matches('[0-9]+') ? T(java.lang.Integer).parseInt(#stringValue) + 100 :
```

Performance Issues

Expensive operations in conditions:

```
# Problematic - expensive operation repeated
condition: "#expensiveCalculation() > 0 && #expensiveCalculation() < 1000"

# Better - calculate once
calculations:
- field: "calculationResult"
  expression: "#expensiveCalculation()"
- field: "isValid"
  expression: "#calculationResult > 0 && #calculationResult < 1000"
```

11.2 Debugging Techniques

Expression Testing

Test expressions in isolation:

```
# Add debug calculations to test expressions
calculations:
- field: "debug.inputQuantity"
  expression: "#quantity"
- field: "debug.inputPrice"
  expression: "#price"
- field: "debug.multiplication"
  expression: "#quantity * #price"
- field: "debug.finalResult"
  expression: "#debug.multiplication"
```

Logging Strategies

Add logging fields for troubleshooting:

```
calculations:
- field: "log.processingTimestamp"
  expression: "T(java.time.Instant).now().toString()"
- field: "log.inputSummary"
  expression: "'Processing trade: ' + #tradeId + ' for ' + #counterpartyName"
- field: "log.calculationDetails"
  expression: "'Quantity: ' + #quantity + ', Price: ' + #price + ', Result: ' + (#quantity * #price)"
```

14. Reference

14.1 Syntax Quick Reference

Operators Table

| Operator | Description | Example |
|----------|------------------------|---|
| == | Equality | #status == 'ACTIVE' |
| != | Inequality | #quantity != 0 |
| > , >= | Greater than | #price > 100 |
| < , <= | Less than | #discount < 0.1 |
| && | Logical AND | #isActive && #quantity > 0 |
| | Logical OR | #status == 'PENDING' #status == 'PROCESSING' |
| ! | Logical NOT | !#isDeleted |
| ?: | Ternary | #value > 0 ? 'POSITIVE' : 'NEGATIVE' |
| ?. | Safe navigation | #trade?.security?.instrumentId |
| + | Addition/Concatenation | #quantity + #bonus |

| Operator | Description | Example |
|----------|----------------|---------------------------------|
| - | Subtraction | <code>#total - #discount</code> |
| * | Multiplication | <code>#quantity * #price</code> |
| / | Division | <code>#amount / #rate</code> |
| % | Modulo | <code>#value % 10</code> |

Function Reference

String Functions:

```
#text.toUpperCase()      # Convert to uppercase
#text.toLowerCase()     # Convert to lowercase
#text.trim()            # Remove whitespace
#text.substring(0, 10)  # Extract substring
#text.length()          # Get string length
#text.contains('substring') # Check if contains
#text.startsWith('prefix') # Check if starts with
#text.endsWith('suffix') # Check if ends with
#text.matches('regex')  # Regex match
#text.replace('old', 'new') # Replace text
```

Math Functions:

```
T(java.lang.Math).max(a, b)      # Maximum of two values
T(java.lang.Math).min(a, b)      # Minimum of two values
T(java.lang.Math).abs(value)     # Absolute value
T(java.lang.Math).sqrt(value)    # Square root
T(java.lang.Math).pow(base, exp) # Power
T(java.lang.Math).round(value)   # Round to nearest integer
T(java.lang.Math).ceil(value)    # Round up
T(java.lang.Math).floor(value)   # Round down
```

Date Functions:

```
T(java.time.LocalDate).now()      # Current date
T(java.time.Instant).now().toString() # Current timestamp
#date.plusDays(2)                 # Add days
#date.minusMonths(1)              # Subtract months
#date.isAfter(otherDate)          # Date comparison
#date.format(T(java.time.format.DateTimeFormatter).ofPattern('yyyy-MM-dd')) # Format date
```

12.2 SpEL Integration

Supported SpEL Features

APEX YAML supports these SpEL features:

- **Literal expressions:** `'Hello World'` , `123` , `true`
- **Property access:** `#property` , `#nested.property`
- **Method invocation:** `#text.toUpperCase()`

- **Operators:** Arithmetic, comparison, logical, ternary
- **Variables:** `#root` , `#this` , custom variables
- **Collection operations:** `#list[0]` , `#list.size()`
- **Type references:** `T(java.lang.Math).max(a, b)`
- **Safe navigation:** `#optional?.property`

APEX-Specific Extensions

APEX adds these extensions to standard SpEL:

- **Data context:** Automatic `#data` variable for input data
- **Field references:** Direct field access in lookup keys
- **Enrichment chaining:** Reference fields created by previous enrichments
- **Null-safe operations:** Enhanced null safety beyond standard SpEL

Limitations and Constraints

Not supported:

- **Variable assignment:** Cannot create new variables (except in calculations)
- **Loops:** No for/while loop constructs
- **Complex object creation:** Limited to simple expressions
- **File I/O:** No direct file system access
- **Network operations:** No direct HTTP/network calls

Performance constraints:

- **Expression complexity:** Keep expressions reasonably simple
- **Recursion:** Avoid recursive expressions
- **Memory usage:** Large datasets should use external sources

15. Migration & Compatibility

Version Compatibility

APEX YAML maintains backward compatibility within major versions:

- **Major versions** (1.x → 2.x): May introduce breaking changes
- **Minor versions** (1.1 → 1.2): Backward compatible, new features
- **Patch versions** (1.1.1 → 1.1.2): Bug fixes, fully compatible

Migration Strategies

From Version 1.0 to 1.1

No breaking changes, but new features available:

```
# New in 1.1: Enhanced error handling
rules:
  - id: "example-rule"
    name: "Example Rule"
```

```
condition: "#field != null"
message: "Field is required"
severity: "ERROR"
# New in 1.1: Custom error codes
error-code: "FIELD_REQUIRED"
# New in 1.1: Retry configuration
retry-on-failure: true
```

Deprecated Features

Version 1.0 deprecated syntax:

```
# Deprecated: Old action syntax
actions:
- type: "lookup"
  source: "dataset"

# Current: New enrichment syntax
enrichments:
- type: "lookup-enrichment"
  lookup-config:
    lookup-dataset:
      type: "inline"
```

From Version 1.x to 2.0 - External Data-Source References

APEX 2.0 introduces external data-source references for clean architecture:

Legacy Approach (1.x):

```
# Old: Inline data-source configuration
metadata:
  name: "Legacy Configuration"
  version: "1.0.0"

data-sources:
- name: "customer-database"
  type: "database"
  connection:
    url: "jdbc:postgresql://localhost:5432/customers"
    username: "user"
    password: "pass"
  queries:
    getCustomer:
      sql: "SELECT * FROM customers WHERE id = :id"

enrichments:
- id: "customer-lookup"
  type: "lookup-enrichment"
  lookup-config:
    lookup-dataset:
      type: "database"
      data-source: "customer-database"
      query: "getCustomer"
```

Modern Approach (2.0):

```
# New: External data-source references
metadata:
  name: "Modern Configuration"
  version: "2.0.0"

# Clean separation: Infrastructure references
data-source-refs:
  - name: "customer-database"
    source: "data-sources/customer-database.yaml" # External file
    enabled: true

# Clean separation: Business logic only
enrichments:
  - id: "customer-lookup"
    type: "lookup-enrichment"
    lookup-config:
      lookup-dataset:
        type: "database"
        data-source-ref: "customer-database" # Reference to external config
        query-ref: "getCustomer"           # Named query from external config
```

Migration Benefits:

- **Clean Architecture:** Infrastructure and business logic separated
- **Reusable Components:** External configurations shared across multiple rules
- **Configuration Caching:** External configurations cached for performance
- **Enterprise Scalability:** Environment-specific infrastructure configurations

Future Roadmap

Planned features:

- **Enhanced debugging:** Better error messages and debugging tools
- **Performance optimizations:** Improved expression evaluation
- **Extended functions:** More built-in functions and utilities
- **IDE integration:** Better tooling support
- **Schema validation:** Runtime schema validation
- **Advanced external references:** Support for more external data-source types

Conclusion

This APEX YAML Syntax Reference provides comprehensive guidance for creating maintainable, efficient, and robust business rules and enrichment logic. APEX 2.0's **external data-source reference system** enables enterprise-grade clean architecture with separation of concerns.

The key to success with APEX YAML is:

1. **Start simple:** Begin with basic patterns and gradually add complexity
2. **Use external references:** Leverage external data-source references for clean architecture
3. **Follow best practices:** Use proper naming, error handling, and performance optimization
4. **Test thoroughly:** Validate your configurations with comprehensive test data
5. **Document well:** Add comments and maintain clear, readable configurations

6. **Monitor performance:** Keep track of execution times and optimize as needed
7. **Separate concerns:** Keep infrastructure and business logic configurations separate

For additional support and examples, refer to the APEX documentation and community resources.