


APEX YAML Syntax Reference Guide

Version: 2.0 **Date:** 2025-08-28 **Author:** Mark Andrew Ray-Smith Cityline Ltd

Table of Contents

- [1. Introduction & Overview](#)
- [2. Document Structure & Metadata](#)
- [3. Core Syntax Elements](#)
- [4. Rules Section](#)
- [5. Enrichments Section](#)
- [6. Dataset Definitions](#)
- [7. External Data-Source References](#)  **NEW**
- [8. Advanced Features](#)
- [9. Best Practices](#)
- [10. Common Patterns](#)
- [11. Examples & Use Cases](#)
- [12. Troubleshooting](#)
- [13. Reference](#)
- [14. 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 a data context using `#data` prefix
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

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

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}"

```

3. Core Syntax Elements

3.1 Data Access Patterns

The `#data` Prefix

All data access in APEX YAML uses the `#data` prefix to reference the input data context:

```

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

# Accessing nested fields
condition: "#data.trade.security.instrumentId != null"

```

```
# Using in calculations
expression: "#data.trade.quantity * #data.trade.price"
```

Nested Field Access with Dot Notation

Access nested objects using dot notation:

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

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

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

Null-Safe Navigation

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

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

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

Array and Collection Access

Access arrays and collections:

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

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

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

3.2 Condition Syntax

Boolean Expressions

Basic boolean logic:

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

# Negation
condition: "!#data.isDeleted"

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

Comparison Operators

All standard comparison operators are supported:

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

# Inequality
condition: "#data.quantity != 0"

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

Logical Operators

Combine conditions with logical operators:

```
# AND operator
condition: "#data.isActive && #data.quantity > 0"

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

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

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

String Operations

String manipulation and comparison:

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

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

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

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

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

Regular Expression Support

Use regex for pattern matching:

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

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

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

Null Checks and Validation

Proper null handling:

```
# Null check
condition: "#data.fieldName != null"

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

# Null-safe string operations
condition: "#data.fieldName?.trim()?.length() > 0"

# Default values for null fields
expression: "#data.fieldName != null ? #data.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', #data.quantity * #data.price); #tradeValue"

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

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

Mathematical Operations

Standard mathematical operations:

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

# Mathematical functions via Java Math class
expression: "T(java.lang.Math).max(#data.value1, #data.value2)"
expression: "T(java.lang.Math).min(#data.value1, #data.value2)"
expression: "T(java.lang.Math).abs(#data.value)"
expression: "T(java.lang.Math).sqrt(#data.value)"
```



```
expression: "T(java.lang.Math).pow(#data.base, #data.exponent)"
expression: "T(java.lang.Math).round(#data.value * 100) / 100.0" # Round to 2 decimals
```

String Manipulation

String operations and formatting:

```
# String concatenation
expression: "#data.firstName + ' ' + #data.lastName"

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

# String manipulation
expression: "#data.text.toUpperCase()"
expression: "#data.text.toLowerCase()"
expression: "#data.text.trim()"
expression: "#data.text.substring(0, 10)"
expression: "#data.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: "#data.tradeDate.plusDays(2)" # Add 2 days
expression: "#data.startDate.plusMonths(1)" # Add 1 month
expression: "#data.endDate.minusYears(1)" # Subtract 1 year

# Date comparisons
condition: "#data.settlementDate.isAfter(T(java.time.LocalDate).now())"
condition: "#data.maturityDate.isBefore(#data.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(#data.amount).multiply(T(java.math.BigDecimal).valueOf(#data.rate))"

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

# Custom utility classes
expression: "T(com.company.utils.FinancialUtils).calculateInterest(#data.principal, #data.rate, #data.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: "#data.trade != null && #data.trade.tradeId != null && #data.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: "#data.security != null && #data.security.isin != null && #data.security.isin.matches('^[A-Z]{2}[A-Z0-9]{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: "#data.quantity != null && #data.price != null && (#data.quantity * #data.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	"#data.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: "#data.tradeDate != null && #data.settlementDate != null && #data.settlementDate.isAfter(#data.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: "#data.instrumentType != 'DERIVATIVE' || (#data.instrumentType == 'DERIVATIVE' && #data.marginAmount != nu
  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: "#data.creditRating == null || (#data.creditRating >= 1 && #data.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: "#data.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: "#data.counterparty.jurisdiction == 'EU' && #data.notionalAmount > 1000000"
    message: "Trade requires EMIR reporting"
    severity: "INFO"
    priority: 2

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

```

5. Enrichments Section

5.1 Lookup Enrichments

Lookup enrichments add data by matching keys against datasets:

```

enrichments:
- id: "lei-enrichment"
  type: "lookup-enrichment"
  condition: "#data.counterparty != null && #data.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"
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: "#data.quantity != null && #data.price != null"
  calculations:
    - field: "tradeValue"
      expression: "#data.quantity * #data.price"
    - field: "tradeValueUSD"
      expression: "#data.currency == 'USD' ? #tradeValue : #tradeValue * #data.exchangeRate"
    - field: "commission"
      expression: "#tradeValue * 0.001" # 0.1% commission
    - field: "netAmount"
      expression: "#tradeValue + #commission"

- id: "risk-calculations"
  type: "calculation-enrichment"
  condition: "#data.tradeValue != null"
  calculations:
    - field: "var1Day"
      expression: "#data.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.DateTi
```

6. Dataset Definitions

6.1 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

7. External Data-Source References

7.1 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"
  url: "jdbc:h2:mem:apex_demo_shared;DB_CLOSE_DELAY=-1;MODE=PostgreSQL"
  username: "sa"
  password: ""
  pool:
    initial-size: 5
    max-size: 20
    timeout: 30000

# 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. Advanced Features

8.1 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: "#data.type == 'EQUITY' && #data.quantity > 1000 ? 'LARGE_EQUITY' : 'OTHER'"
```

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

Complex Branching

Handle multiple conditions efficiently:

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

Performance Optimization

Optimize conditions for better performance:

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

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

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

8.2 Function Usage

Built-in Functions

APEX provides access to standard Java functions:

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

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

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

# Collection functions
expression: "#data.list.size()"
expression: "#data.list.contains('value')"
```

```
expression: "#data.list.[field > 100].size()" # Filter and count
```

Custom Function Integration

Access custom utility classes:

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

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

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

Error Handling in Functions

Handle potential errors gracefully:

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

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

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

9. Best Practices

9.1 Performance Guidelines

Condition Optimization

Write efficient conditions:

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

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

# Best: Cache results of expensive operations
calculations:
- field: "cachedResult"
  expression: "#data.expensiveOperation()"
- field: "finalCheck"
  expression: "#data.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: "#data.complexCalc() + #data.complexCalc() * 0.1"

# Better: Calculate once and reuse
calculations:
  - field: "baseValue"
    expression: "#data.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: "#data.quantity * #data.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: "#data.positions != null && #data.positions.size() > 0"
  calculations:
    # Calculate 1-day VaR at 99% confidence level
    - field: "var1Day99"
      expression: "#data.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: "#data.customRate != null ? #data.customRate : #data.standardRate"

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

```

Null Safety

Always check for null values:

```

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

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

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

```

10. Common Patterns

10.1 Financial Services Patterns

Reference Data Enrichment Pattern

Standard pattern for enriching with reference data:


```

enrichments:
- id: "security-master-enrichment"
  type: "lookup-enrichment"
  condition: "#data.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: "#data.marketValue != null"
  calculations:
    # Value at Risk calculations
    - field: "var1Day95"
      expression: "#data.marketValue * 0.0164" # 1.64 * volatility
    - field: "var1Day99"
      expression: "#data.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: "#data.reportingEntity.lei + '-' + #data.tradeId + '-' + T(java.time.LocalDate).now().format(T(java.t

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

```

9.2 Data Validation Patterns

Format Validation Pattern

Standard format validation approach:

```
rules:
- id: "isin-format"
  name: "ISIN Format Validation"
  condition: "#data.isin == null || #data.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: "#data.lei == null || #data.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: "#data.settlementDate == null || T(com.company.utils.BusinessDayUtils).isBusinessDay(#data.settlementDate,"
  message: "Settlement date must be a business day in the market country"
  severity: "ERROR"

- id: "trade-limit-check"
  name: "Trade Limit Validation"
  condition: "#data.tradeValue <= #data.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: "#data.tradeDate == null || #data.settlementDate == null || #data.settlementDate.isAfter(#data.tradeDate)"
  message: "Settlement date must be after trade date"
  severity: "ERROR"

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

11. Examples & Use Cases

11.1 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: "#data.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: "#data.quantity != null && #data.price != null"
    calculations:
      - field: "tradeValue"
        expression: "#data.quantity * #data.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: "#data.tradeId != null && #data.counterpartyName != null && #data.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: "#data.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: "#data.quantity != null && #data.price != null"
    calculations:
      - field: "tradeValue"
        expression: "#data.quantity * #data.price"
      - field: "tradeValueUSD"
        expression: "#data.currency == 'USD' ? #tradeValue : #tradeValue * #data.fxRate"

  # Step 3: Determine settlement instructions based on enriched data
  - id: "settlement-instructions"
    type: "lookup-enrichment"
    condition: "#data.counterparty.lei != null && #data.venue.country != null"

```

```

lookup-config:
  lookup-key: "#counterparty.lei + '_' + #venue.country"
lookup-dataset:
  type: "inline"
  key-field: "key"
  data:
    - key: "7LTFWZYICNSX8D621K86_GB"
      method: "CREST"
      account: "CREST001234"
    - key: "7LTFWZYICNSX8D621K86_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: "#data.tradeValueUSD != null && #data.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"

```

12. Troubleshooting

12.1 Common Errors

Syntax Errors

Missing #data prefix:

```

# Wrong
condition: "trade.quantity > 0"

# Correct
condition: "#data.trade.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: "#data.value = 100"

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

Runtime Errors

NullPointerException:

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

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

Type conversion errors:

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

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

Performance Issues

Expensive operations in conditions:

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

# Better - calculate once
calculations:
- field: "calculationResult"
  expression: "#data.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: "#data.quantity"
- field: "debug.inputPrice"
  expression: "#data.price"
```

- field: "debug.multiplication"
expression: "#data.quantity * #data.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: ' + #data.tradeId + ' for ' + #data.counterpartyName"
- field: "log.calculationDetails"
  expression: "'Quantity: ' + #data.quantity + ', Price: ' + #data.price + ', Result: ' + (#data.quantity * #data.price"
```

13. Reference

13.1 Syntax Quick Reference

Operators Table

Operator	Description	Example
==	Equality	#data.status == 'ACTIVE'
!=	Inequality	#data.quantity != 0
> , >=	Greater than	#data.price > 100
< , <=	Less than	#data.discount < 0.1
&&	Logical AND	#data.isActive && #data.quantity > 0
	Logical OR	#data.status == 'PENDING' #data.status == 'PROCESSING'
!	Logical NOT	!#data.isDeleted
?:	Ternary	#data.value > 0 ? 'POSITIVE' : 'NEGATIVE'
?.	Safe navigation	#data.trade?.security?.instrumentId
+	Addition/Concatenation	#data.quantity + #data.bonus
-	Subtraction	#data.total - #data.discount
*	Multiplication	#data.quantity * #data.price
/	Division	#data.amount / #data.rate
%	Modulo	#data.value % 10

Function Reference

String Functions:

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

Math Functions:

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

Date Functions:

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

12.2 SpEL Integration

Supported SpEL Features

APEX YAML supports these SpEL features:

- **Literal expressions:** 'Hello World' , 123 , true
- **Property access:** `#data.property` , `#data.nested.property`
- **Method invocation:** `#data.text.toUpperCase()`
- **Operators:** Arithmetic, comparison, logical, ternary
- **Variables:** `#root` , `#this` , custom variables
- **Collection operations:** `#data.list[0]` , `#data.list.size()`
- **Type references:** `T(java.lang.Math).max(a, b)`
- **Safe navigation:** `#data.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

14. 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: "#data.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.