CSV Import Support for Dynamic Attributes

Version 1.3.0-SNAPSHOT, 2026-02-13T23:56:09Z

Chapter 1. Executive Summary

This document proposes a design for importing `dynamicAttributeSets` and `dynamicAttributes` via CSV. The challenge is that dynamic attributes are schema-less, nested structures where the attribute names and types are not known at compile time, yet CSV is an inherently flat, column-based format.

Chapter 2. Background: Dynamic Attributes Data Model

2.1. Current Structure

```
// A model with dynamic attributes
public class Product extends BaseModel {
    protected String name;
    protected String sku;
    protected List<DynamicAttributeSet> dynamicAttributeSets = new ArrayList<>();
}

// Container for related attributes
public class DynamicAttributeSet {
    protected String name;                          // e.g., "logistics", "compliance"
    protected List<DynamicAttribute> attributes;
}

// Individual attribute with type and value
public class DynamicAttribute {
    protected String id;
    protected String name;                          // e.g., "weight", "hazmat_class"
    protected DynamicAttributeType type;            // String, Integer, Boolean, Date,
etc.
    protected Object value;                         // The actual value
    protected Object defaultValue;
    protected boolean required;
    protected boolean hidden;
    // ... other metadata
}
```

2.2. Example JSON Representation

```
{
  "name": "Widget Pro",
  "sku": "WGT-001",
  "dynamicAttributeSets": [
    {
      "name": "logistics",
      "attributes": [
        { "name": "weight", "type": "Double", "value": 2.5 },
        { "name": "dimensions", "type": "String", "value": "10x5x3" },
        { "name": "hazmat", "type": "Boolean", "value": false }
      ]
    },
    {
```

```
      "name": "compliance",
      "attributes": [
        { "name": "ce_certified", "type": "Boolean", "value": true },
        { "name": "certification_date", "type": "Date", "value": "2024-01-15" }
      ]
    }
  ]
}
```

Chapter 3. Design Challenges

1. **Flat vs Nested**: CSV is flat; dynamic attributes are deeply nested (model → sets → attributes)
2. **Schema Variability**: Different rows may have different dynamic attributes
3. **Type Coercion**: CSV values are strings; dynamic attributes have typed values
4. **Merge Semantics**: Should import replace all attributes or merge with existing?
5. **Validation**: How to validate against DynamicAttributeSetDefinition if one exists?
6. **Performance**: Potentially many columns for dynamic attributes

Chapter 4. Proposed Design: All Three Strategies Supported

The design supports **all three import strategies**, configurable via ImportProfile. Each strategy has its use cases, and users can choose based on their data shape and tooling.

4.1. Strategy Overview

Strategy	Best For	CSV Style	Configuration
DOT_NOTATION	Spreadsheet-friendly, sparse attributes	`dyn.setName.attrName` columns	`dynamicAttributeStrategy: DOT_NOTATION`
JSON_COLUMN	Complex structures, programmatic generation	Single `dynamicAttributes` JSON column	`dynamicAttributeStrategy: JSON_COLUMN`
VERTICAL	Attribute-centric imports, bulk attribute loading	Multiple rows per entity	`dynamicAttributeStrategy: VERTICAL`

4.2. Strategy 1: DOT_NOTATION (Header Convention)

Use a special header prefix and dot notation to map CSV columns to dynamic attributes.

4.2.1. Header Format

```
dyn.<setName>.<attributeName>
dyn.<setName>.<attributeName>:type
```

4.2.2. Example CSV

```
name,sku,dyn.logistics.weight,dyn.logistics.dimensions,dyn.logistics.hazmat,dyn.compli
ance.ce_certified,dyn.compliance.certification_date
"Widget Pro","WGT-001",2.5,"10x5x3",false,true,"2024-01-15"
"Gadget X","GDG-002",1.2,"5x5x2",false,true,"2024-03-20"
```

4.2.3. Type Inference

Types can be inferred or explicitly specified:

Method	Header	Notes
Inferred from value	`dyn.logistics.weight`	Parser attempts: Boolean → Integer → Long → Double → Date → String

Method	Header	Notes
Explicit type suffix	`dyn.logistics.weight:Double`	Forces specific type
From DynamicAttributeSetDefinition	`dyn.logistics.weight`	Looks up type from definition if exists
From ImportProfile	Configured in profile	Type specified in column mapping

4.2.4. Best For

- Spreadsheet editing (Excel, Google Sheets)
- Human-readable CSV files
- Sparse attributes (some rows have attributes, others don't)
- When attribute names are known upfront

4.3. Strategy 2: JSON_COLUMN (JSON-in-Cell)

Store the entire dynamic attributes structure as JSON in a dedicated CSV column.

4.3.1. Column Format

The column name is configurable (default: `dynamicAttributes`). The value is a JSON object:

```
{
  "setName": {
    "attrName": value,
    "attrName2": value2
  },
  "setName2": {
    "attrName": value
  }
}
```

4.3.2. Example CSV

```
name,sku,dynamicAttributes
"Widget Pro","WGT-
001","{""logistics"":{""weight"":2.5,""dimensions"":""10x5x3"",""hazmat"":false},""com
pliance"":{""ce_certified"":true,""certification_date"":""2024-01-15""}}"
"Gadget X","GDG-002","{""logistics"":{""weight"":1.2,""dimensions"":""5x5x2""}}"
```

4.3.3. Alternative: Relaxed JSON Format

For easier editing, support a relaxed format with single quotes or unquoted keys:

```
name,sku,dynamicAttributes
"Widget Pro","WGT-
001","{'logistics':{'weight':2.5,'hazmat':false},'compliance':{'ce_certified':true}}"
```

4.3.4. Full Structure JSON (Optional)

For complete control, support full DynamicAttributeSet structure:

```
name,sku,dynamicAttributeSets
"Widget Pro","WGT-
001","[{""name"":""logistics"",""attributes"":[{""name"":""weight"",""type"":""Double"
","""value"":2.5}]}]"
```

4.3.5. Best For

- Programmatically generated CSV files
- Complex attribute structures
- When attributes vary significantly between rows
- Integration with systems that export JSON

4.4. Strategy 3: VERTICAL (Multiple Rows per Entity)

One row per attribute, with a parent identifier to group rows into entities.

4.4.1. Column Format

Required columns:

Column	Description
`_entityKey`	Unique identifier to group rows (e.g., SKU, refName)
`_attrSet`	Dynamic attribute set name
`_attrName`	Attribute name within the set
`_attrType`	Attribute type (optional, can be inferred)
`_attrValue`	The attribute value

Column names are configurable in ImportProfile.

4.4.2. Example CSV

```
_entityKey,_attrSet,_attrName,_attrType,_attrValue
"WGT-001","logistics","weight","Double","2.5"
"WGT-001","logistics","dimensions","String","10x5x3"
"WGT-001","logistics","hazmat","Boolean","false"
"WGT-001","compliance","ce_certified","Boolean","true"
"WGT-001","compliance","certification_date","Date","2024-01-15"
"GDG-002","logistics","weight","Double","1.2"
"GDG-002","logistics","dimensions","String","5x5x2"
```

4.4.3. With Entity Fields

Include entity fields on each row (redundant but allows single-pass processing):

```
sku,name,_attrSet,_attrName,_attrType,_attrValue
"WGT-001","Widget Pro","logistics","weight","Double","2.5"
"WGT-001","Widget Pro","logistics","dimensions","String","10x5x3"
"WGT-001","Widget Pro","compliance","ce_certified","Boolean","true"
"GDG-002","Gadget X","logistics","weight","Double","1.2"
```

4.4.4. Best For

- Bulk attribute imports (adding many attributes to existing entities)
- Attribute-focused workflows
- When entity already exists and you're just updating attributes
- Exports from attribute management systems

4.5. Hybrid Mode: Combining Strategies

Enable multiple strategies in a single import by setting `dynamicAttributeStrategy: HYBRID`:

```
name,sku,dyn.logistics.weight,dynamicAttributes
"Widget Pro","WGT-001",2.5,"{""compliance"":{""ce_certified"":true}}"
"Gadget X","GDG-002",1.2,"{""compliance"":{""rohs"":true}}"
```

Processing order: 1. DOT_NOTATION columns processed first 2. JSON_COLUMN merged/overlaid 3. Explicit ColumnMappings applied last (highest priority)

Chapter 5. Detailed Design: Dot Notation Approach

5.1. Configuration in ImportProfile

5.1.1. New Enums

```
/**
 * Strategy for importing dynamic attributes from CSV.
 */
public enum DynamicAttributeImportStrategy {
    /**
     * Disable dynamic attribute import.
     */
    NONE,

    /**
     * Use dot-notation headers: dyn.setName.attrName[:type]
     * Best for spreadsheet-friendly imports.
     */
    DOT_NOTATION,

    /**
     * Use a single JSON column containing all dynamic attributes.
     * Best for programmatic imports.
     */
    JSON_COLUMN,

    /**
     * Use vertical format with multiple rows per entity.
     * Best for attribute-centric imports.
     */
    VERTICAL,

    /**
     * Support both DOT_NOTATION and JSON_COLUMN in same import.
     * DOT_NOTATION processed first, JSON_COLUMN merged on top.
     */
    HYBRID
}

/**
 * Strategy for merging imported dynamic attributes with existing.
 */
public enum DynamicAttributeMergeStrategy {
    /**
     * Remove all existing attributes, replace with imported.
```

```
     */
    REPLACE,

    /**
     * Keep existing attributes, update matching, add new.
     */
    MERGE,

    /**
     * Keep existing attributes, only add new (no updates to existing).
     */
    APPEND
}
```

5.1.2. DynamicAttributeMapping

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
public class DynamicAttributeMapping {
    /**
     * The CSV column name or pattern.
     * For DOT_NOTATION: can use "dyn.{setName}.{attrName}" for auto-discovery.
     * For JSON_COLUMN: the column containing JSON.
     * For VERTICAL: not used (uses verticalConfig).
     */
    private String sourceColumn;

    /**
     * Target attribute set name. If null, extracted from header.
     */
    private String targetSetName;

    /**
     * Target attribute name. If null, extracted from header.
     */
    private String targetAttributeName;

    /**
     * Force a specific type. If null, inferred.
     */
    private DynamicAttributeType type;

    /**
     * Date/time format for Date/DateTime types.
     */
    private String dateFormat;
```

```
    /**
     * Reference to DynamicAttributeSetDefinition for validation.
     */
    private String definitionRefName;
}
```

5.1.3. VerticalFormatConfig

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
public class VerticalFormatConfig {
    /**
     * Column containing the entity key for grouping rows.
     * Default: "_entityKey"
     */
    @Builder.Default
    private String entityKeyColumn = "_entityKey";

    /**
     * Field on the entity that matches entityKeyColumn values.
     * Used for lookups when updating existing entities.
     * Default: "refName"
     */
    @Builder.Default
    private String entityKeyField = "refName";

    /**
     * Column containing the attribute set name.
     * Default: "_attrSet"
     */
    @Builder.Default
    private String setNameColumn = "_attrSet";

    /**
     * Column containing the attribute name.
     * Default: "_attrName"
     */
    @Builder.Default
    private String attrNameColumn = "_attrName";

    /**
     * Column containing the attribute type (optional).
     * Default: "_attrType"
     */
    @Builder.Default
```

```
    private String attrTypeColumn = "_attrType";

    /**
     * Column containing the attribute value.
     * Default: "_attrValue"
     */
    @Builder.Default
    private String attrValueColumn = "_attrValue";

    /**
     * If true, entity fields can be included on each row.
     * First occurrence of each field value is used.
     * Default: false
     */
    @Builder.Default
    private boolean allowEntityFieldsOnRows = false;

    /**
     * Columns to exclude from entity field processing.
     * Typically the attribute columns themselves.
     */
    private List<String> excludeFromEntityFields;
}
```

5.1.4. JsonColumnConfig

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
public class JsonColumnConfig {
    /**
     * The CSV column name containing JSON dynamic attributes.
     * Default: "dynamicAttributes"
     */
    @Builder.Default
    private String columnName = "dynamicAttributes";

    /**
     * JSON format style.
     * COMPACT: {"setName":{"attrName":value}} - values only, types inferred
     * FULL:
[{"name":"setName","attributes":[{"name":"attrName","type":"String","value":"x"}]}]
     */
    @Builder.Default
    private JsonFormatStyle formatStyle = JsonFormatStyle.COMPACT;

    /**
```

```
     * Allow relaxed JSON parsing (single quotes, unquoted keys).
     * Default: true
     */
    @Builder.Default
    private boolean relaxedParsing = true;
}

public enum JsonFormatStyle {
    /**
     * Compact format: {"setName":{"attrName":value,...},...}
     * Types are inferred from JSON value types.
     */
    COMPACT,

    /**
     * Full structure matching DynamicAttributeSet model:
     *
[{"name":"setName","attributes":[{"name":"attrName","type":"Type","value":val}]}]
     */
    FULL
}
```

5.1.5. Updated ImportProfile

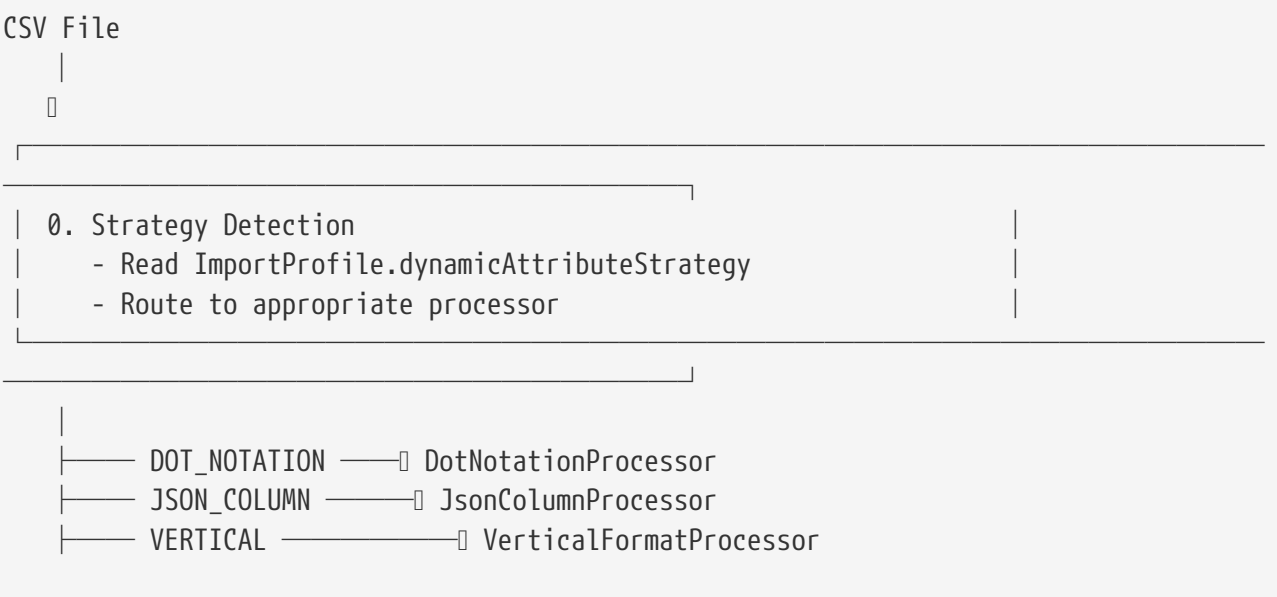
```
public class ImportProfile extends BaseModel {
    // ... existing fields ...

    //
═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════
    // DYNAMIC ATTRIBUTE CONFIGURATION
    //
═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════


    /**
     * Strategy for importing dynamic attributes.
     * Default: NONE (disabled)
     */
    @Builder.Default
    private DynamicAttributeImportStrategy dynamicAttributeStrategy =
            DynamicAttributeImportStrategy.NONE;

    /**
     * Strategy for merging imported dynamic attributes with existing.
     * Default: MERGE
     */
    @Builder.Default
    private DynamicAttributeMergeStrategy dynamicAttributeMergeStrategy =
```

```
            DynamicAttributeMergeStrategy.MERGE;

    // --- DOT_NOTATION Configuration ---

    /**
     * Prefix for auto-discovery of dynamic attribute columns.
     * Only used when strategy is DOT_NOTATION or HYBRID.
     * Default: "dyn."
     */
    @Builder.Default
    private String dynamicAttributePrefix = "dyn.";

    /**
     * Enable auto-discovery of dynamic attribute columns by prefix.
     * If false, only explicitly mapped columns are processed.
     */
    @Builder.Default
    private boolean enableDynamicAttributeDiscovery = true;

    // --- JSON_COLUMN Configuration ---

    /**
     * Configuration for JSON column strategy.
     * Only used when strategy is JSON_COLUMN or HYBRID.
     */
    private JsonColumnConfig jsonColumnConfig;

    // --- VERTICAL Configuration ---

    /**
     * Configuration for vertical format strategy.
     * Only used when strategy is VERTICAL.
     */
    private VerticalFormatConfig verticalFormatConfig;

    // --- Common Configuration ---

    /**
     * Explicit mappings for dynamic attribute columns.
     * Used for DOT_NOTATION when auto-discovery is disabled,
     * or to override inferred types/settings.
     */
    private List<DynamicAttributeMapping> dynamicAttributeMappings;

    /**
     * Reference to DynamicAttributeSetDefinition for type/validation lookup.
     * If specified, validates imported attributes against this definition.
     */
    private String dynamicAttributeDefinitionRefName;

    /**
```

```
     * If true, only allow attributes defined in the DynamicAttributeSetDefinition.
     * Unknown attributes will cause validation errors.
     * Default: false (allow any attributes)
     */
    @Builder.Default
    private boolean strictDynamicAttributeValidation = false;

    /**
     * Default type when type cannot be inferred.
     * Default: String
     */
    @Builder.Default
    private DynamicAttributeType defaultDynamicAttributeType =
            DynamicAttributeType.String;

    /**
     * Date format for parsing Date type attributes.
     * Default: "yyyy-MM-dd"
     */
    @Builder.Default
    private String dynamicAttributeDateFormat = "yyyy-MM-dd";

    /**
     * DateTime format for parsing DateTime type attributes.
     * Default: "yyyy-MM-dd'T'HH:mm:ss"
     */
    @Builder.Default
    private String dynamicAttributeDateTimeFormat = "yyyy-MM-dd'T'HH:mm:ss";
}
```

5.2. Processing Pipeline

5.2.1. Overall Flow



```
    └──── HYBRID ──────────────▯ Both Dot + Json processors
```

5.2.2. DOT_NOTATION Processing Pipeline

```
CSV Row
    |
    ▯
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 1. Header Parsing (once per import)                                                          │
│    - Identify columns with "dyn." prefix                                                     │
│    - Parse: dyn.<setName>.<attrName>[:type]                                                  │
│    - Build column → (setName, attrName, type) mapping                                        │
│    - Validate header format                                                                  │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
    |
    ▯
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 2. Regular Column Processing                                                                 │
│    - Process non-dynamic columns as before                                                   │
│    - Apply existing transformations                                                          │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
    |
    ▯
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 3. Dynamic Attribute Extraction                                                              │
│    - For each dyn.* column with non-empty value:                                             │
│      a. Get parsed header info (setName, attrName)                                           │
│      b. Determine type (explicit > definition > inferred)                                    │
│      c. Convert string value to typed value                                                  │
│      d. Create DynamicAttribute object                                                       │
│    - Group attributes by set name                                                            │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
    |
    ▯
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 4. Set Assembly                                                                              │
│    - Create DynamicAttributeSet for each set name                                            │
│    - Populate with collected attributes                                                      │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
    |
    ▯
```

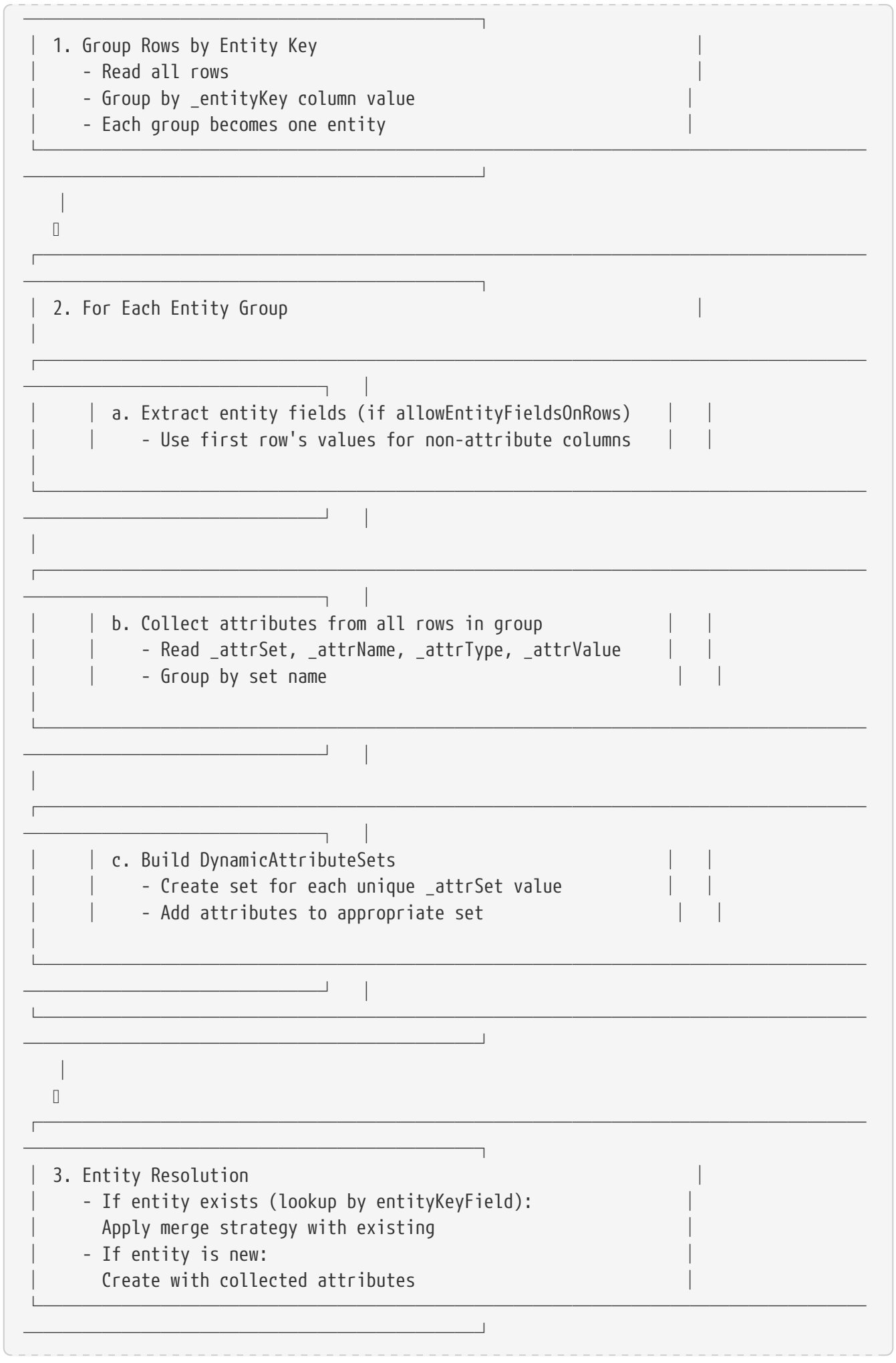
```
[Merge & Validate] ──────▶ Bean with dynamicAttributeSets
```

5.2.3. JSON_COLUMN Processing Pipeline

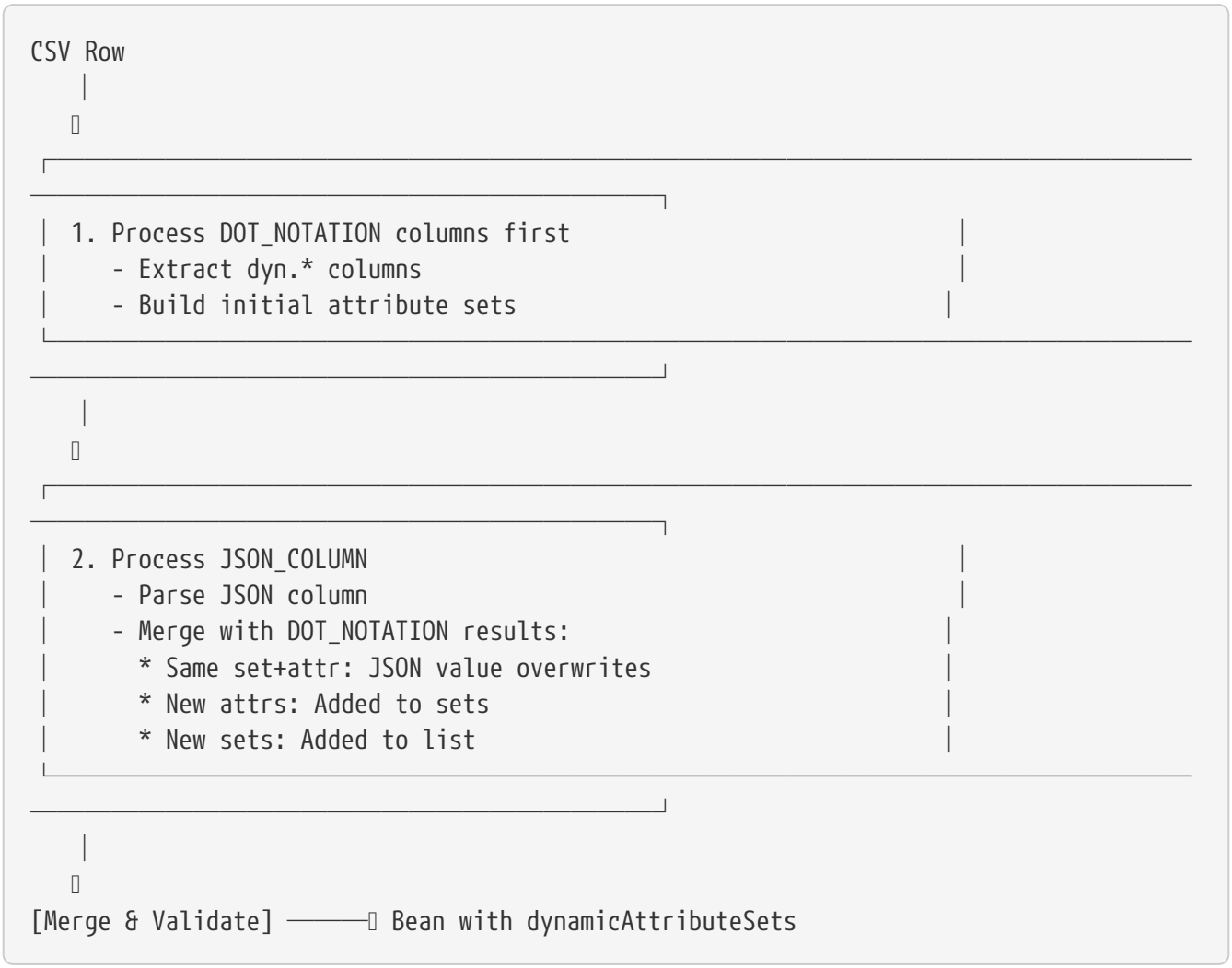
```
CSV Row
   │
   ▼
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 1. Locate JSON Column                                                                        │
│    - Find column by name (default: "dynamicAttributes")                                      │
│    - Skip if empty/null                                                                      │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
   │
   ▼
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 2. Parse JSON                                                                                │
│    - If COMPACT format:                                                                      │
│      Parse {"setName":{"attrName":value,...},...}                                            │
│      Infer types from JSON value types                                                       │
│    - If FULL format:                                                                         │
│      Parse as List<DynamicAttributeSet>                                                      │
│      Use explicit types from structure                                                       │
│    - Handle relaxed JSON if enabled                                                          │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
   │
   ▼
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│ 3. Build Attribute Sets                                                                      │
│    - Create DynamicAttributeSet for each entry                                               │
│    - Create DynamicAttribute for each nested entry                                           │
│    - Map JSON types to DynamicAttributeType                                                  │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
   │
   ▼
[Merge & Validate] ──────▶ Bean with dynamicAttributeSets
```

5.2.4. VERTICAL Processing Pipeline

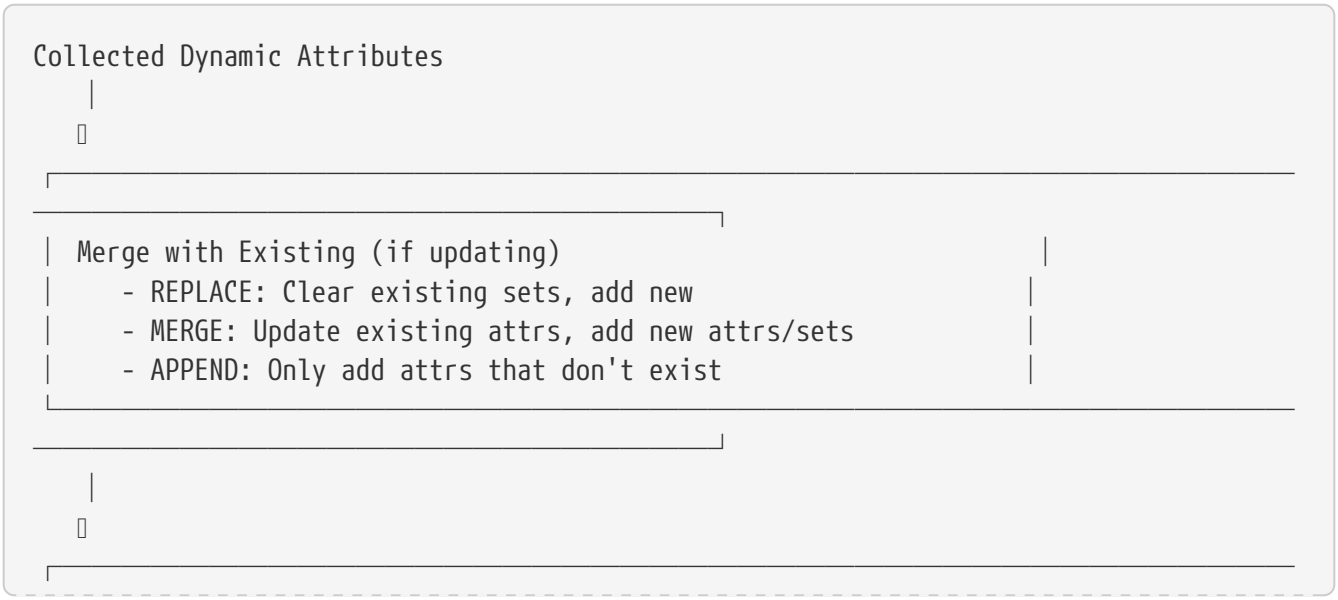
```
CSV File (grouped by entity)
   │
   ▼
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
```


1. Group Rows by Entity Key
- Read all rows
- Group by _entityKey column value
- Each group becomes one entity
2. For Each Entity Group
a. Extract entity fields (if allowEntityFieldsOnRows)
- Use first row's values for non-attribute columns
b. Collect attributes from all rows in group
- Read _attrSet, _attrName, _attrType, _attrValue
- Group by set name
c. Build DynamicAttributeSets
- Create set for each unique _attrSet value
- Add attributes to appropriate set
3. Entity Resolution
- If entity exists (lookup by entityKeyField):
Apply merge strategy with existing
- If entity is new:
Create with collected attributes

5.2.5. HYBRID Processing Pipeline



5.2.6. Common: Merge & Validate Stage



```
┌──────────────────────────────────────────────────────────────────────┐
│ Validation                                                           │
│    - If DynamicAttributeSetDefinition exists:                        │
│      a. Validate required attributes are present                     │
│      b. Validate types match definition                              │
│      c. Validate Select/MultiSelect values are allowed               │
│    - If strictDynamicAttributeValidation:                            │
│      d. Reject unknown attributes not in definition                  │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▯
Bean with populated dynamicAttributeSets
```

5.3. New SPI: DynamicAttributeProcessor

```java
package com.e2eq.framework.imports.spi;

/**
 * Processes dynamic attribute columns during CSV import.
 */
@ApplicationScoped
public class DynamicAttributeProcessor {

    @Inject
    DynamicAttributeSetDefinitionRepo definitionRepo;

    /**
     * Parse a dynamic attribute header.
     *
     * @param header the CSV header (e.g., "dyn.logistics.weight:Double")
     * @param prefix the dynamic attribute prefix (default "dyn.")
     * @return parsed header info, or null if not a dynamic attribute column
     */
    public ParsedDynamicHeader parseHeader(String header, String prefix) {
        if (!header.startsWith(prefix)) {
            return null;
        }

        String remainder = header.substring(prefix.length());
        // Parse: setName.attrName[:type]
        int typeIdx = remainder.lastIndexOf(':');
        String path;
        DynamicAttributeType explicitType = null;

        if (typeIdx > 0) {
            path = remainder.substring(0, typeIdx);
            String typeStr = remainder.substring(typeIdx + 1);
            try {
```

```
                explicitType = DynamicAttributeType.fromValue(typeStr);
            } catch (IllegalArgumentException e) {
                // Invalid type, treat as part of name
                path = remainder;
            }
        } else {
            path = remainder;
        }

        int dotIdx = path.indexOf('.');
        if (dotIdx <= 0 || dotIdx >= path.length() - 1) {
            return null; // Invalid format
        }

        String setName = path.substring(0, dotIdx);
        String attrName = path.substring(dotIdx + 1);

        return new ParsedDynamicHeader(header, setName, attrName, explicitType);
    }

    /**
     * Process dynamic attribute columns for a row.
     *
     * @param bean the target bean
     * @param rowData all column values
     * @param parsedHeaders map of column name to parsed header info
     * @param profile the import profile
     * @param context the import context
     * @return result with any errors
     */
    public ProcessResult processDynamicAttributes(
            BaseModel bean,
            Map<String, Object> rowData,
            Map<String, ParsedDynamicHeader> parsedHeaders,
            ImportProfile profile,
            ImportContext context) {

        // Check if bean has dynamicAttributeSets field
        List<DynamicAttributeSet> sets = getDynamicAttributeSets(bean);
        if (sets == null) {
            return ProcessResult.skip("Bean does not support dynamic attributes");
        }

        // Load definition if specified
        DynamicAttributeSetDefinition definition = loadDefinition(profile, context);

        // Group columns by set name
        Map<String, List<AttributeValue>> bySet = new LinkedHashMap<>();

        for (Map.Entry<String, ParsedDynamicHeader> entry : parsedHeaders.entrySet())
{
```

```
            Object value = rowData.get(entry.getKey());
            if (value == null || value.toString().trim().isEmpty()) {
                continue; // Skip empty values
            }

            ParsedDynamicHeader parsed = entry.getValue();
            DynamicAttributeType type = determineType(parsed, value, definition);
            Object typedValue = convertValue(value.toString(), type, profile);

            bySet.computeIfAbsent(parsed.getSetName(), k -> new ArrayList<>())
                    .add(new AttributeValue(parsed.getAttrName(), type, typedValue));
        }

        // Apply merge strategy
        applyMergeStrategy(sets, bySet, profile.getDynamicAttributeMergeStrategy());

        // Validate against definition if present
        if (definition != null) {
            List<String> errors = validateAgainstDefinition(sets, definition);
            if (!errors.isEmpty()) {
                return ProcessResult.error(String.join("; ", errors));
            }
        }

        return ProcessResult.success();
    }

    // ... helper methods ...
}

@Data
@AllArgsConstructor
public class ParsedDynamicHeader {
    private String originalHeader;
    private String setName;
    private String attrName;
    private DynamicAttributeType explicitType;
}
```

5.4. Type Inference Logic

```
public DynamicAttributeType inferType(String value) {
    if (value == null || value.isEmpty()) {
        return DynamicAttributeType.String;
    }

    // Boolean
    if ("true".equalsIgnoreCase(value) || "false".equalsIgnoreCase(value)) {
        return DynamicAttributeType.Boolean;
```

```
    }

    // Integer
    try {
        Integer.parseInt(value);
        return DynamicAttributeType.Integer;
    } catch (NumberFormatException e) { }

    // Long
    try {
        Long.parseLong(value);
        return DynamicAttributeType.Long;
    } catch (NumberFormatException e) { }

    // Double
    try {
        Double.parseDouble(value);
        return DynamicAttributeType.Double;
    } catch (NumberFormatException e) { }

    // Date (common formats)
    if (isDateFormat(value)) {
        return DynamicAttributeType.Date;
    }

    // DateTime
    if (isDateTimeFormat(value)) {
        return DynamicAttributeType.DateTime;
    }

    // Default to String
    return DynamicAttributeType.String;
}
```

5.5. Merge Strategy Implementation

```
private void applyMergeStrategy(
        List<DynamicAttributeSet> existingSets,
        Map<String, List<AttributeValue>> importedBySet,
        DynamicAttributeMergeStrategy strategy) {

    switch (strategy) {
        case REPLACE:
            existingSets.clear();
            // Fall through to add all imported
        case MERGE:
        case APPEND:
            for (Map.Entry<String, List<AttributeValue>> entry : importedBySet
.entrySet()) {
```

```
                String setName = entry.getKey();
                List<AttributeValue> importedAttrs = entry.getValue();

                DynamicAttributeSet existingSet = findSetByName(existingSets, setName
);
                if (existingSet == null) {
                    // Create new set
                    existingSet = new DynamicAttributeSet();
                    existingSet.setName(setName);
                    existingSet.setAttributes(new ArrayList<>());
                    existingSets.add(existingSet);
                }

                for (AttributeValue av : importedAttrs) {
                    DynamicAttribute existing = findAttrByName(existingSet
.getAttributes(), av.name);

                    if (existing == null) {
                        // Add new attribute
                        DynamicAttribute attr = new DynamicAttribute();
                        attr.setId(UUID.randomUUID().toString());
                        attr.setName(av.name);
                        attr.setType(av.type);
                        attr.setValue(av.value);
                        existingSet.getAttributes().add(attr);
                    } else if (strategy == DynamicAttributeMergeStrategy.MERGE) {
                        // Update existing
                        existing.setType(av.type);
                        existing.setValue(av.value);
                    }
                    // APPEND: skip if exists
                }
            }
            break;
    }
}
```

Chapter 6. CSV Examples

6.1. DOT_NOTATION Strategy Examples

6.1.1. Example 1: Simple Dynamic Attributes

```
refName,displayName,dyn.specs.color,dyn.specs.size,dyn.specs.weight:Double
prod-001,Red Widget,Red,Large,2.5
prod-002,Blue Gadget,Blue,Medium,1.8
```

6.1.2. Example 2: Multiple Attribute Sets

```
sku,name,dyn.logistics.weight:Double,dyn.logistics.hazmat:Boolean,dyn.compliance.ce_ma
rk:Boolean,dyn.compliance.rohs:Boolean
WGT-001,Widget Pro,2.5,false,true,true
GDG-002,Gadget X,1.2,false,true,false
```

6.1.3. Example 3: With Type Suffix

```
refName,dyn.dates.manufactured:Date,dyn.dates.expires:Date,dyn.counts.inventory:Intege
r
item-001,2024-01-15,2025-01-15,500
item-002,2024-02-20,2025-02-20,250
```

6.1.4. Example 4: Sparse Attributes (Optional)

```
refName,dyn.optional.note,dyn.optional.priority:Integer,dyn.optional.reviewed:Boolean
item-001,Important item,1,true
item-002,,,
item-003,Standard item,,false
```

In row 2, `note`, `priority`, and `reviewed` are all empty - these attributes won't be created. In row 3, only `note` and `reviewed` are set.

6.2. JSON_COLUMN Strategy Examples

6.2.1. Example 5: Compact JSON Format

```
refName,displayName,dynamicAttributes
prod-001,Red
Widget,"{""specs"":{""color"":""Red"",""size"":""Large"",""weight"":2.5}}"
prod-002,Blue
```

```
Gadget,"{""specs"":{""color"":""Blue"",""size"":""Medium""},""compliance"":{""ce_mark"
":true}}"
```

6.2.2. Example 6: Relaxed JSON (Single Quotes)

With `relaxedParsing: true`:

```
refName,displayName,dynamicAttributes
prod-001,Red Widget,"{'specs':{'color':'Red','weight':2.5}}"
prod-002,Blue Gadget,"{'specs':{'color':'Blue'},'compliance':{'rohs':true}}"
```

6.2.3. Example 7: Full Structure JSON

With `formatStyle: FULL`:

```
refName,displayName,dynamicAttributeSets
prod-001,Red
Widget,"[{""name"":""specs"",""attributes"":[{""name"":""color"",""type"":""String"","
"value"":""Red""},{""name"":""weight"",""type"":""Double"",""value"":2.5}]}]"
```

6.2.4. Example 8: Custom Column Name

With `jsonColumnConfig.columnName: "customAttrs"`:

```
refName,displayName,customAttrs
prod-001,Red Widget,"{""specs"":{""color"":""Red""}}"
```

6.3. VERTICAL Strategy Examples

6.3.1. Example 9: Basic Vertical Format

```
_entityKey,_attrSet,_attrName,_attrType,_attrValue
WGT-001,logistics,weight,Double,2.5
WGT-001,logistics,dimensions,String,10x5x3
WGT-001,logistics,hazmat,Boolean,false
WGT-001,compliance,ce_mark,Boolean,true
WGT-001,compliance,certification_date,Date,2024-01-15
GDG-002,logistics,weight,Double,1.2
GDG-002,logistics,dimensions,String,5x5x2
GDG-002,compliance,ce_mark,Boolean,true
```

6.3.2. Example 10: Vertical with Entity Fields

With `allowEntityFieldsOnRows: true`:

```
sku,name,_attrSet,_attrName,_attrValue
WGT-001,Widget Pro,logistics,weight,2.5
WGT-001,Widget Pro,logistics,dimensions,10x5x3
WGT-001,Widget Pro,compliance,ce_mark,true
GDG-002,Gadget X,logistics,weight,1.2
GDG-002,Gadget X,compliance,ce_mark,true
```

6.3.3. Example 11: Vertical with Type Inference

Type column is optional when `_attrType` is omitted:

```
_entityKey,_attrSet,_attrName,_attrValue
WGT-001,specs,color,Red
WGT-001,specs,weight,2.5
WGT-001,specs,active,true
WGT-001,specs,count,42
```

Types inferred: color=String, weight=Double, active=Boolean, count=Integer

6.3.4. Example 12: Custom Column Names

With custom `verticalFormatConfig`:

```
product_sku,set_name,attribute,type,value
WGT-001,logistics,weight,Double,2.5
WGT-001,logistics,hazmat,Boolean,false
```

6.4. HYBRID Strategy Examples

6.4.1. Example 13: Combined DOT_NOTATION + JSON

```
refName,displayName,dyn.specs.color,dyn.specs.size,dynamicAttributes
prod-001,Red Widget,Red,Large,"{""compliance"":{""ce_mark"":true,""rohs"":true}}"
prod-002,Blue Gadget,Blue,Medium,"{""compliance"":{""ce_mark"":true}}"
```

Result: `specs` set from DOT_NOTATION columns, `compliance` set from JSON.

6.4.2. Example 14: Hybrid with Override

JSON values override DOT_NOTATION for same attribute:

```
refName,dyn.specs.color,dynamicAttributes
prod-001,Red,"{""specs"":{""color"":""Dark Red"",""size"":""XL""}}"
```

Result: `color` = "Dark Red" (JSON wins), `size` = "XL" (from JSON only)

Chapter 7. ImportProfile Configuration Examples

7.1. DOT_NOTATION: Auto-Discovery Mode

```
{
  "refName": "product-import-dot-notation",
  "displayName": "Product Import with Dot Notation",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "DOT_NOTATION",
  "enableDynamicAttributeDiscovery": true,
  "dynamicAttributePrefix": "dyn.",
  "dynamicAttributeMergeStrategy": "MERGE"
}
```

7.2. DOT_NOTATION: Explicit Mapping Mode

Map arbitrary column names to dynamic attributes:

```
{
  "refName": "product-import-explicit-mapping",
  "displayName": "Product Import with Explicit Mappings",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "DOT_NOTATION",
  "enableDynamicAttributeDiscovery": false,
  "dynamicAttributeMappings": [
    {
      "sourceColumn": "weight_kg",
      "targetSetName": "logistics",
      "targetAttributeName": "weight",
      "type": "Double"
    },
    {
      "sourceColumn": "is_hazmat",
      "targetSetName": "logistics",
      "targetAttributeName": "hazmat",
      "type": "Boolean"
    },
    {
      "sourceColumn": "ce_certified",
      "targetSetName": "compliance",
      "targetAttributeName": "ce_mark",
      "type": "Boolean"
    }
  ],
  "dynamicAttributeMergeStrategy": "REPLACE"
```

```
}
```

7.3. JSON_COLUMN: Compact Format

```
{
  "refName": "product-import-json-compact",
  "displayName": "Product Import with JSON Column",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "JSON_COLUMN",
  "jsonColumnConfig": {
    "columnName": "dynamicAttributes",
    "formatStyle": "COMPACT",
    "relaxedParsing": true
  },
  "dynamicAttributeMergeStrategy": "MERGE"
}
```

7.4. JSON_COLUMN: Full Structure Format

```
{
  "refName": "product-import-json-full",
  "displayName": "Product Import with Full JSON Structure",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "JSON_COLUMN",
  "jsonColumnConfig": {
    "columnName": "dynamicAttributeSets",
    "formatStyle": "FULL",
    "relaxedParsing": false
  },
  "dynamicAttributeMergeStrategy": "REPLACE"
}
```

7.5. VERTICAL: Basic Configuration

```
{
  "refName": "product-attributes-vertical",
  "displayName": "Product Attributes Vertical Import",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "VERTICAL",
  "verticalFormatConfig": {
    "entityKeyColumn": "_entityKey",
    "entityKeyField": "sku",
    "setNameColumn": "_attrSet",
    "attrNameColumn": "_attrName",
    "attrTypeColumn": "_attrType",
```

```
    "attrValueColumn": "_attrValue",
    "allowEntityFieldsOnRows": false
  },
  "dynamicAttributeMergeStrategy": "MERGE"
}
```

7.6. VERTICAL: With Entity Fields on Rows

```
{
  "refName": "product-vertical-with-fields",
  "displayName": "Product Vertical with Entity Fields",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "VERTICAL",
  "verticalFormatConfig": {
    "entityKeyColumn": "sku",
    "entityKeyField": "sku",
    "setNameColumn": "attr_set",
    "attrNameColumn": "attr_name",
    "attrTypeColumn": "attr_type",
    "attrValueColumn": "attr_value",
    "allowEntityFieldsOnRows": true,
    "excludeFromEntityFields": ["attr_set", "attr_name", "attr_type", "attr_value"]
  },
  "dynamicAttributeMergeStrategy": "APPEND"
}
```

7.7. HYBRID: Combined Strategy

```
{
  "refName": "product-import-hybrid",
  "displayName": "Product Import with Hybrid Strategy",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "HYBRID",
  "dynamicAttributePrefix": "dyn.",
  "enableDynamicAttributeDiscovery": true,
  "jsonColumnConfig": {
    "columnName": "extraAttributes",
    "formatStyle": "COMPACT",
    "relaxedParsing": true
  },
  "dynamicAttributeMergeStrategy": "MERGE"
}
```

7.8. With Definition Validation

```
{
  "refName": "validated-product-import",
  "displayName": "Product Import with Strict Validation",
  "targetCollection": "com.example.Product",
  "dynamicAttributeStrategy": "DOT_NOTATION",
  "enableDynamicAttributeDiscovery": true,
  "dynamicAttributePrefix": "dyn.",
  "dynamicAttributeDefinitionRefName": "product-attributes-definition",
  "strictDynamicAttributeValidation": true,
  "dynamicAttributeMergeStrategy": "MERGE"
}
```

7.9. Complete Example with All Options

```
{
  "refName": "full-product-import-profile",
  "displayName": "Complete Product Import Profile",
  "description": "Demonstrates all dynamic attribute options",
  "targetCollection": "com.example.Product",

  "dynamicAttributeStrategy": "HYBRID",
  "dynamicAttributeMergeStrategy": "MERGE",

  "dynamicAttributePrefix": "dyn.",
  "enableDynamicAttributeDiscovery": true,

  "jsonColumnConfig": {
    "columnName": "jsonAttrs",
    "formatStyle": "COMPACT",
    "relaxedParsing": true
  },

  "dynamicAttributeMappings": [
    {
      "sourceColumn": "weight_lbs",
      "targetSetName": "logistics",
      "targetAttributeName": "weight_pounds",
      "type": "Double"
    }
  ],

  "dynamicAttributeDefinitionRefName": "product-attrs-def",
  "strictDynamicAttributeValidation": false,
  "defaultDynamicAttributeType": "String",
  "dynamicAttributeDateFormat": "MM/dd/yyyy",
  "dynamicAttributeDateTimeFormat": "MM/dd/yyyy HH:mm"
```

```
}
```

Chapter 8. Processing Order Integration

The dynamic attribute processing integrates into the existing CSV import pipeline:

```
1. CSV parsing with CellProcessors
2. Field calculators (timestamps, UUIDs, templates)
3. RowValueResolvers (arbitrary code with full row access)
4. DynamicAttributeProcessor (dynamic attribute import)
5. Pre-validation transformers
6. Bean validation
```

Chapter 9. RowValueResolver: Arc CDI Bean Integration

9.1. Overview

The `RowValueResolver` SPI provides a powerful mechanism for running arbitrary code during CSV import. Unlike static lookups or simple transformations, RowValueResolvers have access to:

- The entire row data (all columns)
- Full CDI injection capabilities (services, repositories, etc.)
- The import context (profile, realm, row number, session)

Key Feature: Since RowValueResolvers are Arc CDI beans, you can inject any other service and update multiple collections from a single CSV row.

9.2. Basic Implementation

```
@ApplicationScoped
public class SkuResolver implements RowValueResolver {

    @Inject
    ProductService productService;

    @Override
    public String getName() {
        return "skuResolver";
    }

    @Override
    public ResolveResult resolve(String inputValue, Map<String, Object> rowData,
ImportContext context) {
        String category = (String) rowData.get("category");
        String vendor = (String) rowData.get("vendor");

        String resolvedSku = productService.findOrCreateSku(inputValue, category,
vendor);

        if (resolvedSku != null) {
            return ResolveResult.success(resolvedSku);
        } else {
            return ResolveResult.error("Could not resolve SKU for: " + inputValue);
        }
    }
}
```

9.3. Multi-Collection Updates

Since RowValueResolvers are Arc beans, you can inject multiple repositories and update multiple collections from a single CSV row:

```
@ApplicationScoped
public class OrderLineResolver implements RowValueResolver {

    @Inject
    OrderRepo orderRepo;

    @Inject
    InventoryRepo inventoryRepo;

    @Inject
    AuditLogRepo auditLogRepo;

    @Override
    public String getName() {
        return "orderLineResolver";
    }

    @Override
    public ResolveResult resolve(String inputValue, Map<String, Object> rowData,
ImportContext context) {
        String orderId = (String) rowData.get("order_id");
        String productSku = (String) rowData.get("product_sku");
        Integer quantity = Integer.parseInt((String) rowData.get("quantity"));

        // Update inventory (different collection)
        inventoryRepo.decrementStock(productSku, quantity, context.getRealmId());

        // Create audit log entry (another collection)
        AuditLog log = AuditLog.builder()
            .action("IMPORT_ORDER_LINE")
            .entityRef(orderId)
            .details("Imported line for SKU: " + productSku)
            .build();
        auditLogRepo.save(log);

        // Return the value for the main entity being imported
        return ResolveResult.success(productSku);
    }
}
```

9.4. Available Injections

RowValueResolvers can inject any Arc/CDI bean:

Injection Type	Example
Repositories	`@Inject ProductRepo productRepo;`
Services	`@Inject EmailService emailService;`
Configuration	`@ConfigProperty(name="app.import.threshold") int threshold;`
Other Resolvers	`@Inject Instance<RowValueResolver> allResolvers;`
MicroProfile Clients	`@Inject @RestClient ExternalApiClient client;`

9.5. Configuration in ImportProfile

Configure a RowValueResolver in your ColumnMapping:

```
{
  "refName": "order-import-profile",
  "targetCollection": "com.example.Order",
  "columnMappings": [
    {
      "sourceColumn": "product_code",
      "targetField": "sku",
      "rowValueResolverName": "skuResolver"
    },
    {
      "sourceColumn": "line_data",
      "targetField": "lineInfo",
      "rowValueResolverName": "orderLineResolver"
    }
  ]
}
```

9.6. Result Types

Result	Usage
`ResolveResult.success(value)`	Set the field to the resolved value
`ResolveResult.passthrough(original)`	Keep the original value unchanged
`ResolveResult.nullValue()`	Set the field to null
`ResolveResult.skip()`	Skip this row (no error, just don't process)
`ResolveResult.skip(reason)`	Skip with a reason (logged but not an error)
`ResolveResult.error(message)`	Mark the row as an error

9.7. ImportContext Information

The `ImportContext` provides access to:

Method	Returns
`getProfile()`	The ImportProfile configuration
`getTargetClass()`	The class being imported
`getRealmId()`	The realm/tenant ID
`getRowNumber()`	Current row number (1-based)
`getSessionId()`	The import session ID

9.8. Advanced: Conditional Processing

```
@ApplicationScoped
public class ConditionalResolver implements RowValueResolver {

    @Inject
    ConfigService configService;

    @Override
    public String getName() {
        return "conditionalResolver";
    }

    @Override
    public boolean appliesTo(Class<?> targetClass) {
        // Only apply to Order entities
        return Order.class.isAssignableFrom(targetClass);
    }

    @Override
    public int getOrder() {
        // Run before other resolvers (lower = earlier)
        return 50;
    }

    @Override
    public ResolveResult resolve(String inputValue, Map<String, Object> rowData,
ImportContext context) {
        // Check configuration
        if (!configService.isFeatureEnabled("advanced-import", context.getRealmId()))
{
            return ResolveResult.passthrough(inputValue);
        }

        // Skip certain rows based on data
        String status = (String) rowData.get("status");
```

```
        if ("CANCELLED".equals(status)) {
            return ResolveResult.skip("Skipping cancelled orders");
        }

        // Process normally
        return ResolveResult.success(processValue(inputValue, rowData));
    }
}
```

9.9. Testing RowValueResolvers

```
@QuarkusTest
class SkuResolverTest {

    @Inject
    SkuResolver resolver;

    @Test
    void testResolveSuccess() {
        Map<String, Object> rowData = Map.of(
            "category", "electronics",
            "vendor", "acme"
        );
        ImportContext context = ImportContext.builder()
            .realmId("test-realm")
            .rowNumber(1)
            .build();

        RowValueResolver.ResolveResult result = resolver.resolve("PROD-001", rowData,
context);

        assertTrue(result.isSuccess());
        assertNotNull(result.getValue());
    }
}
```

Chapter 10. Edge Cases and Error Handling

10.1. Empty/Null Values

- Empty string → attribute not created (sparse support)
- Explicit null marker (configurable, e.g., "NULL") → attribute created with null value

10.2. Type Mismatches

- If type is explicit or from definition, conversion failure → error
- If type is inferred, use String as fallback

10.3. Invalid Header Format

- `dyn.` with no dot after set name → warning, skip column
- `dyn.setname` with no attribute name → warning, skip column

10.4. Missing Required Attributes

- If DynamicAttributeSetDefinition specifies `required=true`, validate presence
- Report all missing required attributes as validation errors

10.5. Unknown Sets/Attributes

- By default, allow any set/attribute names (schema-less)
- If `strictMode=true` in profile, only allow defined sets/attributes

Chapter 11. Implementation Components

11.1. New Model Classes

Component	Package	Description
DynamicAttributeImportStrategy	com.e2eq.framework.model.persistent.imports	Enum: NONE, DOT_NOTATION, JSON_COLUMN, VERTICAL, HYBRID
DynamicAttributeMergeStrategy	com.e2eq.framework.model.persistent.imports	Enum: REPLACE, MERGE, APPEND
DynamicAttributeMapping	com.e2eq.framework.model.persistent.imports	Configuration for mapping CSV columns to dynamic attributes
JsonColumnConfig	com.e2eq.framework.model.persistent.imports	Configuration for JSON_COLUMN strategy
JsonFormatStyle	com.e2eq.framework.model.persistent.imports	Enum: COMPACT, FULL
VerticalFormatConfig	com.e2eq.framework.model.persistent.imports	Configuration for VERTICAL strategy
ParsedDynamicHeader	com.e2eq.framework.imports.service	Parsed dynamic attribute header info for DOT_NOTATION

11.2. New Service Classes

Component	Package	Description
DynamicAttributeImportService	com.e2eq.framework.imports.service	Main orchestrator for dynamic attribute processing
DotNotationProcessor	com.e2eq.framework.imports.service	Processes DOT_NOTATION strategy columns
JsonColumnProcessor	com.e2eq.framework.imports.service	Processes JSON_COLUMN strategy
VerticalFormatProcessor	com.e2eq.framework.imports.service	Processes VERTICAL strategy with row grouping
DynamicAttributeTypeInferrer	com.e2eq.framework.imports.service	Type inference from string values
DynamicAttributeMerger	com.e2eq.framework.imports.service	Applies merge strategies (REPLACE, MERGE, APPEND)
DynamicAttributeValidator	com.e2eq.framework.imports.service	Validates against DynamicAttributeSetDefinition

11.3. Enhanced Existing Classes

Component	Package	Description
`ImportProfile` (enhanced)	`com.e2eq.framework.model.persistent.imports`	New fields for dynamic attribute configuration
`ImportProfileService` (enhanced)	`com.e2eq.framework.imports.service`	Integration with DynamicAttributeImportService
`CSVImportHelper` (enhanced)	`com.e2eq.framework.util`	Strategy detection and processor routing

Chapter 12. Future Considerations

1. **Nested Attributes**: Support for nested objects within attribute values (e.g., `dyn.address.coordinates.lat`)
2. **Multi-Value Attributes**: Support for MultiSelect with comma-separated values or JSON arrays
3. **Inheritance**: Respect `inheritable` flag when merging with parent entities
4. **Export Symmetry**: Ensure CSV export uses same conventions for round-trip imports
5. **Bulk Operations**: Streaming processor for VERTICAL strategy with very large files
6. **Mixed Entity Types**: VERTICAL format with multiple entity types in same file
7. **Attribute Templates**: Pre-defined attribute set templates for common patterns
8. **Change Detection**: Only update attributes that actually changed

Chapter 13. Summary

13.1. Dynamic Attribute Import Strategies

The design supports **four import strategies** for dynamic attributes, each suited to different use cases:

Strategy	Description
DOT_NOTATION	Spreadsheet-friendly with `dyn.setName.attrName[:type]` column headers. Best for human-edited CSV files with known attribute structures.
JSON_COLUMN	Single JSON column containing all attributes. Best for programmatic imports or complex/variable attribute structures.
VERTICAL	Multiple rows per entity with one attribute per row. Best for attribute-centric workflows or bulk attribute loading.
HYBRID	Combines DOT_NOTATION and JSON_COLUMN. Allows spreadsheet columns for common attributes with JSON for extras.

All strategies share common features:

- **Merge Strategies**: REPLACE, MERGE, APPEND for handling existing attributes
- **Type Handling**: Explicit types, definition lookup, or inference
- **Validation**: Optional validation against DynamicAttributeSetDefinition
- **Sparse Support**: Missing/empty values handled gracefully

13.2. RowValueResolver: Extensible Import Logic

The `RowValueResolver` SPI enables powerful, custom import logic:

- **Arc CDI Beans**: Full dependency injection support
- **Multi-Collection Updates**: Update multiple collections from a single row
- **Full Row Access**: Read any column value, not just the mapped column
- **Context Awareness**: Access to realm, session, row number, and profile
- **Flexible Results**: Return values, skip rows, or report errors

13.3. Implementation Summary

The implementation includes:

13.3.1. Model Classes (quantum-models)

- `DynamicAttributeImportStrategy` - NONE, DOT_NOTATION, JSON_COLUMN, VERTICAL, HYBRID
- `DynamicAttributeMergeStrategy` - REPLACE, MERGE, APPEND

- `JsonColumnConfig` - Configuration for JSON_COLUMN strategy
- `VerticalFormatConfig` - Configuration for VERTICAL strategy
- `DynamicAttributeMapping` - Explicit column-to-attribute mappings
- `ParsedDynamicHeader` - Parsed header info for DOT_NOTATION
- `DynamicAttributeSupport` - Interface for models supporting dynamic attributes

13.3.2. Service Classes (quantum-framework)

- `DynamicAttributeImportService` - Main orchestrator
- `DotNotationProcessor` - DOT_NOTATION strategy processor
- `JsonColumnProcessor` - JSON_COLUMN strategy processor
- `VerticalFormatProcessor` - VERTICAL strategy processor
- `DynamicAttributeTypeInferrer` - Type inference from string values
- `DynamicAttributeMerger` - Merge strategy implementation

13.3.3. Integration Points

- `ImportProfile` - Extended with dynamic attribute configuration
- `CSVImportHelper` - Integrated for automatic processing
- `RowValueResolver` - Arc CDI bean SPI for custom logic