Caliente 3.5

Filtering, Transformation, and Attribute Mappings

Operator Manual

Effective Date: July 2nd, 2014	**Approved By:** Management Team
Version Number: 0.1.0	**Contact Name:** Diego Rivera
Responsible Group: Armedia Technology CoP	**Asset Type:** Architecture Document
Area: Caliente, Content Management, Migration Tools	**Document Number:** TBD

Table of Contents

Document History

Name	Date	Reason For Changes	Version	Approval
Diego Rivera	2018-4-30	Initial Draft	0.1	

1 Introduction

1.1 Document Purpose

The purpose of this document is to describe the implementation syntax and details for Caliente's transformation and attribute mapping schemes. Both of these schemes are key when Caliente is used to translate content from one ECM engine to another, different one.

1.2 Audience

This document contains information aimed at technical staff that will be tasked with configuring and running Caliente for the purposes of data migration between ECM engines.

1.3 Concept Introduction

Caliente is an automated, large-volume content migration solution. It offers mechanisms through which an operator can instruct the underlying engine as to the ways that content will need to be modified from its source ECM's form in order to be ingested into the target ECM.

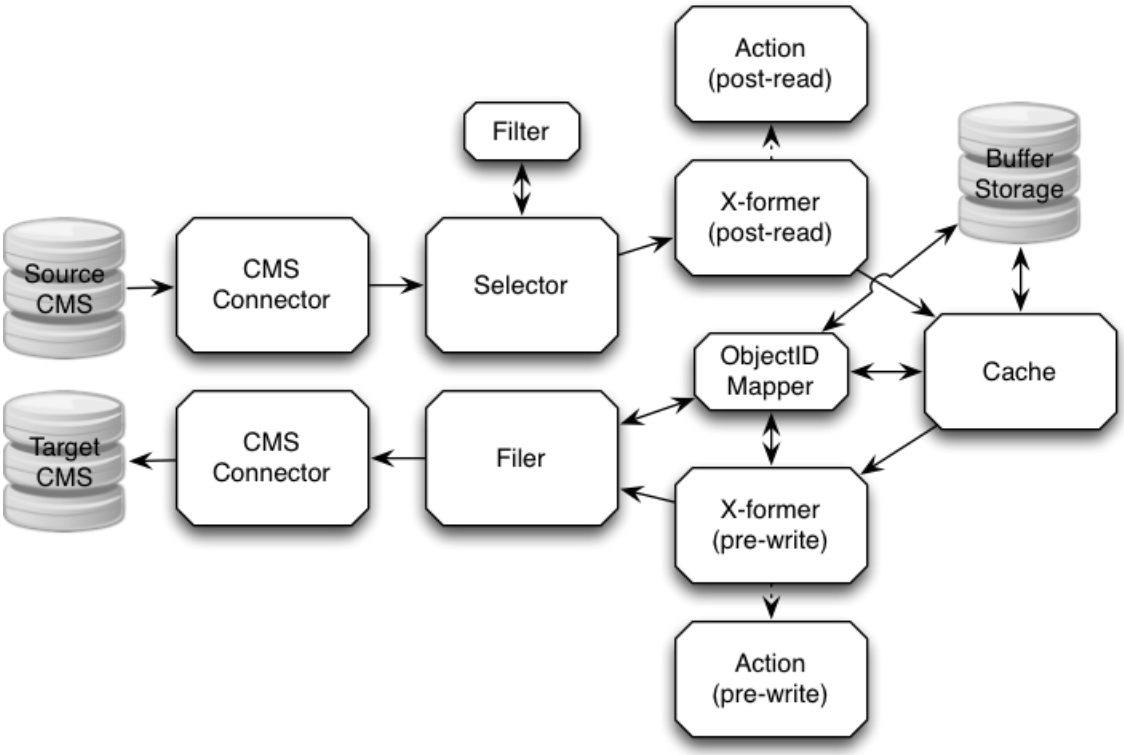
In order to facilitate re-use of code amongst different projects, Caliente employs an XML-based syntax in order to instruct its underlying transformation and attribute mapping components as to the tasks that need to be executed, their order, and the circumstances of execution.

There are pre-defined actions that are provided to satisfy the most commonly-used transformations, as well as the ability to integrate additional, as-yet-undefined complex actions written in Java, or using script engines such as JEXL, Groovy, Python (Jython), JavaScript, or BSF.

2 High-level Concept Architecture

2.1 Data Flow

Caliente's data flow roughly follows the following diagram:



2.1.1 Extraction

1. The reader component reads data from the source ECM
2. The data is fed through an ***export filter*** to allow the Caliente engine the opportunity to exclude the object from storage alongside the rest of the extracted data. An object excluded in this manner can be said to be ***SKIPPED*** by Caliente
3. Once the object has been accepted for processing, all its metadata is loaded into memory and Caliente begins the process of preparing that data for serialization into intermediate storage.
4. Just before the data is serialized, the transformation layer is afforded an opportunity to modify the data that is to be serialized. This is referred to as the ***export transformation***
5. The data is then serialized into Caliente's intermediate storage

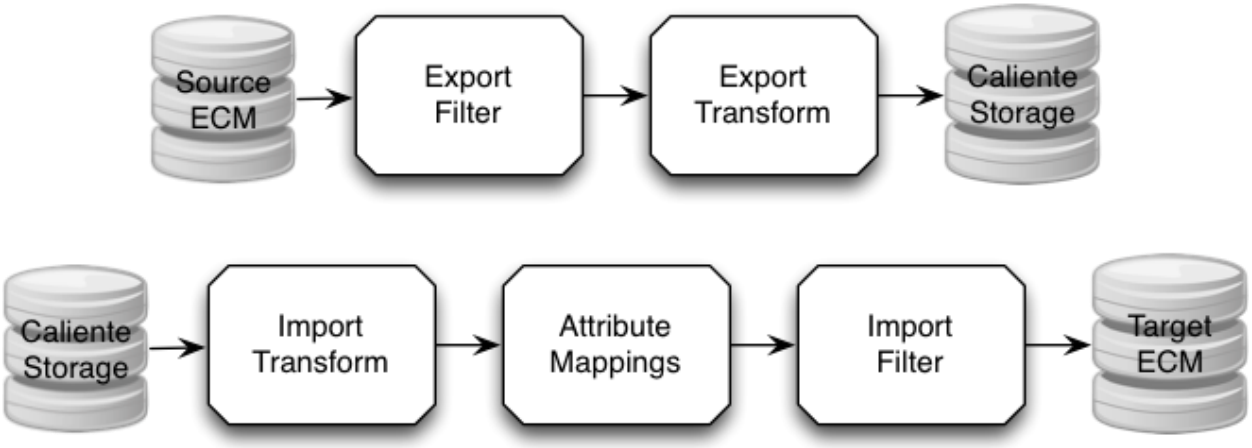
2.1.2 Ingestion

1. The Caliente engine begins to read back the data from intermediate storage
2. When an object is read from storage, all its metadata is loaded into memory, and Caliente applies what is known as an ***import transformation*** **BEFORE** the data is made available to the rest of the engine. That is to say: the data loading component immediately invokes the

transformation layer on each loaded object before the rest of the engine has a chance to look at it.

3. Immediately after this, the ***attribute mapping*** is applied in order to give the object its final form
4. Once each object is ready for processing, it is fed through an ***import filter*** to allow the Caliente engine the opportunity to exclude the object from the data being ingested. An object excluded in this manner can be said to be ***SKIPPED*** by Caliente.
5. Once an object has been accepted for processing, it is handed off to the ingestion engine which in turn will finish processing the object in whatever manner is appropriate

2.1.3 Execution Order Diagram



2.2 Caliente Internals

2.2.1 Data Types

Some filter conditions and transformation actions may reference specific data types for specific types of operations. Caliente supports the following data types:

- BOOLEAN
- INTEGER
- DOUBLE
- STRING
- ID
- DATETIME
- URI

- HTML
- BASE64_BINARY

2.2.2 Object Archetypes

Caliente supports the following object types (referred to as *archetypes*) in both actions and filter conditions:

- DATASTORE
- USER
- GROUP
- ACL
- TYPE
- FORMAT
- FOLDER
- DOCUMENT

An object ***cannot*** change its archetype under any circumstance during transformation.

2.2.3 Object Subtypes

In Caliente terminology, an object's ***type*** is its *archetype* (see above), where as its ***subtype*** is its specific object type in ECM terminology. For example: *dm_document*, *dm_folder*, *cm:content*, *arm:testDocument*, *dm_accounting_invoice*, etc.

An object ***cannot*** change its archetype under any circumstance during transformation, but its ***subtype*** can change as part of the transformation process. In fact, when moving data between different ECM engine types (i.e. Documentum to Alfresco), this is usually required.

2.2.4 Object Secondary Types

In Caliente terminology, an object can have multiple ***secondary types*** associated with it. In common ECM-speak, these are equivalent to *aspects*, as implemented in Alfresco or Documentum. Secondary types may be examined by condition filters, or modified by transformation actions freely including removing all assigned types at once.

2.2.5 Caliente Properties

Objects stored in Caliente may have properties associated with them that are similar in all respects to the object's inherent metadata attributes, but which exist solely to orient Caliente's execution internals. These properties may be manipulated or examined by filters and transformation actions, and will not normally be persisted into the target environment unless explicitly converted into regular metadata attributes.

These properties may be single- or multi-valued, and may even have the same name as regular metadata attributes. They will also have a specific data type (see Data Types, above).

2.3 Common Details for Transformation and Filtering

The following sections describe some common aspects referenced in the documentation for transformations and filters.

2.3.1 Scripting Languages

The transformation and filtering layers support the use of scripting languages within expressions embedded into the XML using the ***lang*** attribute (populated with one of the script language's aliases), and CDATA sections (if necessary to accommodate script syntax). In general, when the documentation references the term ***EXPRESSION***, it refers to a value that can either be a literal (i.e. no ***lang*** attribute is provided), or a script expression that calculates the value to use (if ***lang*** is provided and one of the valid language name values, see below). In a few cases, the expression must be a script expression using a default scripting language as specified in a case-by-case basis if ***lang*** is not specified.

Caliente supports the following scripting languages out of the box:

- JEXL 2
    - **Aliases:** JEXL2
- JEXL 3 (the default, where a default is required)
    - **Aliases:** JEXL, JEXL3
- JavaScript (via the JVM's embedded interpreter)
    - **Aliases:** JS, Rhino, JavaScript, ECMAScript
- BeanShell
    - **Aliases:** BeanShell, BSH, Java
- Groovy
    - **Aliases:** Groovy

Others can be added by putting the required JARs into the Caliente library directory at run-time. Language aliases are case-insensitive.

2.3.2 Cardinality

Caliente supports both single-valued and multi-valued attributes. In the case of multivalued attributes, some filters and transformation actions may affect or take into account only one of the values in the value collection. This is referred to as the action's or filter's ***cardinality***.

Elements that support this feature can use one of the following values (case-insensitive):

- ***all*** – affect all values (only used in transformative actions)
- ***any*** – compare against any of the values (only used in filter conditions)
- ***** - an alias for ***all*** or ***any*** depending on whether it's a transformative action or a filter condition
- ***first*** – only take the first value into account
- ***last*** – only take the last value into account

2.3.3 Comparisons

Some filter checks will inevitably have to resolve the issue of comparing between two values in order to render a ***TRUE*** or ***FALSE*** decision. These are the comparison modes available in Caliente out of the box:

- EQ – The checked value is ***equal*** to the given value
- GE – The checked value is ***greater than or equals*** to the given value
- GT – The checked value is ***greater than*** the given value
- LE – The checked value is ***less than or equals*** to the given value
- LT – The checked value is ***less than*** the given value
- CN –The checked value ***contains*** the given value
- RE – The checked value ***matches a regular expression***
- GLOB – The checked value ***matches a glob expression*** (i.e: like a filesystem wildcard)

Each of these may be prefixed with a letter "***N***" to negate its meaning (i.e. NEQ = Not Equals, NCN = Not Contains, etc.). Furthermore, you may add a letter "***I***" as a suffix to indicate that the comparision should be case-insensitive.

Comparison specifications are case-insensitive (i.e. Neq == NEQ == nEq == NeQ == neq == …).

These are some examples of these prefixes and suffixes in play:

- NEQI – Not Equals, case-insensitive
- GLOBI – Glob comparison, case-insensitive
- NRE – Does not match the Regular Expression
- NGT – Is not greater than
- LTI – Is less than, case-insensitive

- NCN – Does not contain
- NCNI – Does not contain, case-insensitive

As there are 8 core comparisions available, and there are two different suffixes which can modify each comparison's behavior, in reality there are a grand total of 32 possible comparisons that can be leveraged in the filter and action definitions.

3 Filters

Filters are a convenient mechanism for Caliente to select which content to extract from the source, or ingest into the target. Filters are specified using a simple XML syntax which supports arbitrarily complex decisions via easy-to-use constructs.

In particular, Caliente provides many ready-made conditions, as well as means for combining them into complex decision trees. It also supports customized check conditions written in Java (loaded dynamically), JEXL, BSF, Jython, Groovy, and JavaScript.

3.1 Sample Syntax

Here is an example XML from an actual filter implementation used in production:

```
<?xml version="1.1" encoding="UTF-8"?>
<filters xmlns="http://www.armedia.com/ns/caliente/engine">
    <filter>
        <if>
            <and>
                <is-type>DOCUMENT</is-type>
                <is-name comparison="globi">*.zip</is-name>
            </and>
        </if>

        <reject-object/>
    </filter>

    <default>ACCEPT</default>
</filters>
```

The above filter, when used for extraction, causes Caliente to ***skip*** (***reject***) every ***document*** object it finds whose filename ends with ***.zip***, while accepting all other objects.

This is a more general example of a filter XML, without using specific condition names, to highlight the base structure:

```
<?xml version="1.1" encoding="UTF-8"?>
<filters xmlns="http://www.armedia.com/ns/caliente/engine">
      <filter>
            <if>
                  <and>
                        <or>
                              <condition-1>...</condition-1>
                              <condition-2>...</condition-2>
                              <not>
                                    <condition-3>...</condition-3>
                              </not>
                        </or>
                        <xor>
```

```
                              <condition-4>...</condition-4>
                              <condition-5>...</condition-5>
                        </xor>
                        <condition-6>...</condition-6>
                        <!-- ... more conditions or groups? -->
                  </and>
                  <!-- ... more conditions or groups? -->
            </if>
            <filter-outcome/>
      </filter>
      <filter>
            <if>
                  <!-- ... filter conditions or groups ... -->
            </if>
            <filter-outcome/>
      </filter>
      <!-- ... more filter declarations ... -->
      <default>default-outcome-if-no-filter-matched</default>
</filters>
```

The syntax is designed to allow for multiple filters to be defined, each with its independent outcome and conditions. The syntax also defines a default outcome to apply to objects that do not match any of the defined filters, as a catch-all safety net.

The structure is designed around several components:

3.1.1 Filters

Filters can be thought of as containers which allow an operator to conveniently organize their filtering code. Multiple filters can be declared in a single filtering configuration.

Each filter is evaluated individually and independently upon each object in the order declared within the file, and the object's ***outcome*** will be that of the first filter that ***matches***.

A filter can be said to match an object when the condition defined in its ***<if>*** section evaluates to ***TRUE***.

3.1.2 Conditions

There are two kinds of conditions available: conditions and condition groups.

Conditions are components which perform specific, singular checks upon an object's metadata, and return a binary value of either ***TRUE*** or ***FALSE***.

Condition groups allow complex condition rules to be employed by combining conditions and returning ***TRUE*** or ***FALSE*** depending on the outcomes of its individual children, as per pre-defined rules for each condition group type. Condition groups may contain other condition groups in turn. It is through this nesting ability that arbitrarily-complex condition checks may be implemented.

3.1.3 Outcomes

An object's outcome dictates what Caliente will do with that object once the Filtering step is completed. There are only two possible outcomes: ***accept*** and ***reject***. An ***accepted*** object continues its course through Caliente's processing normally. A ***rejected*** object does not – processing for that object is stopped immediately, and its processing status is set to ***SKIPPED***.

Please note that rejected objects may in turn cause a cascade effect where other objects that depend on its correct processing may in turn fail to process correctly, or even be skipped in turn. The execution settings for Caliente will determine this, as well as any implied or explicit inter-object object dependencies.

3.2 Filter Condition Implementations

Filter conditions may be employed within filters' transformation actions' <if> constructs. This allows operators to employ a singular logic pattern when deploying complex filtering and transformation logic within their Caliente environment.

3.2.1 check-expression

Compare the two given expressions (specified via the ***left*** and ***right*** elements), converting each value to the type specified in the ***type*** element (default is ***STRING***), and using the given ***comparison*** (default ***EQ***). This is equivalent to evaluating: ***left COMPARISON right***, and returning the result.

```
<check-expression [type="STRING"] [comparison="eq"]>
      <left [lang="..."]> EXPRESSION </left>
      <right [lang="..."]> EXPRESSION </right>
</check-expression>
```

3.2.2 custom-check

Invoke a custom check written in Java, whose fully-qualified class name is specified by the given ***expression***. The class must implement the ***com.armedia.caliente.engine.dynamic.Condition*** interface, and must be available at launch time by adding JARs to Caliente's library path.

There must also be an instance of ***com.armedia.caliente.engine.dynamic.ConditionFactory***, for which the invocation of ***acquireInstance(String)*** with the custom class's fully qualified name will yield a valid ***Condition*** instance. This class will be located using ***java.util.ServiceLoader***.

```
<custom-check [lang="..."]>
      EXPRESSION <!-- should evaluate to a fully-qualified java class name -->
</custom-check>
```

3.2.3 custom-script

Implement a custom check using an ***expression*** in any of the supported languages. If no ***lang*** attribute is specified, the default is JEXL. The script's return value will be cast to a Boolean value as follows:

- If the return value is ***null***, or there is no return value, then it will evaluate to ***FALSE***
- If the value is a ***java.lang.Boolean***, then a direct cast to java.util.Boolean is performed

- If the value is a ***java.lang.Number***, then its long value (via Number.longValue()) will be converted to ***TRUE*** if it's a non-zero value, ***FALSE*** if the value is zero
- If the value is a string that can be parsed as a ***java.lang.Long***, then it is parsed as such and the value will be converted to ***TRUE*** if it's a non-zero value, ***FALSE*** if the value is zero
- Otherwise, the value will be cast to a String, and converted using ***java.lang.Boolean.valueOf(String)***

```
<custom-script [lang="..."]>
      EXPRESSION
</custom-script>
```

3.2.4 has-attribute

Evaluates to ***TRUE*** if the object contains an attribute whose name matches the given ***expression*** as per the designated ***comparison***.

```
<has-attribute [comparison="eq"] [lang="..."]>
      EXPRESSION
</has-attribute>
```

3.2.5 has-caliente-property

Evaluates to ***TRUE*** if the object contains a custom Caliente property whose name matches the given ***expression*** as per the designated ***comparison***.

```
<has-caliente-property [comparison="eq"] [lang="..."]>
      EXPRESSION
</has-caliente-property>
```

3.2.6 has-original-secondary-subtype

Evaluates to ***TRUE*** if the object originally had a secondary subtype that matches the given ***expression*** value as per the designated ***comparison***.

```
<has-original-secondary-subtype [comparison="eq"] [lang="..."]>
      EXPRESSION
</has-original-secondary-subtype>
```

3.2.7 has-secondary-subtype

Evaluates to ***TRUE*** if the object currently has a secondary subtype that matches the given ***expression*** value as per the designated ***comparison***.

```
<has-secondary-subtype [comparison="eq"] [lang="..."]>
      EXPRESSION
</has-secondary-subtype>
```

3.2.8 has-value-mapping

Evaluates to ***TRUE*** if Caliente's value mapping mechanism currently contains a mapping that matches the given criteria. The ***type*** element is one of Caliente's object types (**DOCUMENT**, **FOLDER**, **USER**, etc). The ***name*** element contains an ***expression*** that evaluates to the name the mapping was given when established. The ***from*** and ***to*** elements are mutually exclusive, and each may contain an ***expression*** that resolves to the **STRING** value to match the mapping's.

Example ***from***:

```
<has-value-mapping>
       <type>ARCHETYPE</type>
       <name [lang="..."]>EXPRESSION</name>
       <from [lang="..."]>EXPRESSION</from>
</has-secondary-subtype>
```

The above example looks for a mapping assigned to the given ***type***, with the given ***name***, whose ***from*** value (key) is equal to the given ***expression***, cast to a **STRING**

Example ***to***:

```
<has-value-mapping>
       <type>ARCHETYPE</type>
       <name [lang="..."]>EXPRESSION</name>
       <to [lang="..."]>EXPRESSION</to>
</has-secondary-subtype>
```

The above example looks for a mapping assigned to the given ***type***, with the given ***name***, whose ***to*** value (value) is equal to the given ***expression***, cast to a **STRING**

3.2.9 is-attribute-empty

Evaluates to ***TRUE*** if the object's attribute whose name matches the given ***expression*** as per the given ***comparison*** is empty. An empty attribute is any single-valued attribute whose value is either **null** or the empty string (""), or a multivalued attribute that has no values or has exactly one value and this value is either **null** or the empty string ("").

```
<is-attribute-empty [comparison="eq"] [lang="..."]>
       EXPRESSION
</is-attribute-empty>
```

3.2.10 is-attribute-repeating

Evaluates to ***TRUE*** if the object's attribute whose name matches the given ***expression*** as per the given ***comparison*** is a multivalued attribute.

```
<is-attribute-repeating [comparison="eq"] [lang="..."]>
       EXPRESSION
</is-attribute-repeating>
```

3.2.11 is-attribute-value

Evaluates to ***TRUE*** if the object's attribute whose name matches the given ***expression***, has a value as per the given value ***expression*** using the given ***comparison***, according to the chosen ***cardinality***.

```
<is-attribute-value [comparison="eq"] [cardinality="any"]>
      <name [lang="..."]>EXPRESSION</name>
      <value [lang="..."]>EXPRESSION</value>
</is-attribute-value>
```

3.2.12 is-caliente-property-empty

Evaluates to ***TRUE*** if the object's Caliente property whose name matches the given ***expression*** as per the given ***comparison*** is empty. An empty Caliente property is any single-valued property whose value is either **null** or the empty string (""), or a multivalued property that has no values or has exactly one value and this value is either **null** or the empty string ("").

```
<is-attribute-empty [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-attribute-empty>
```

3.2.13 is-caliente-property-repeating

Evaluates to ***TRUE*** if the object's Caliente property whose name matches the given ***expression*** as per the given ***comparison*** is a multivalued property.

```
<is-attribute-repeating [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-attribute-repeating>
```

3.2.14 is-caliente-property-value

Evaluates to ***TRUE*** if the object's Caliente property whose name matches the given ***expression***, has a value as per the given value ***expression*** using the given ***comparison***, according to the chosen ***cardinality***.

```
<is-caliente-property-value [comparison="eq"] [cardinality="any"]>
      <name [lang="..."]>EXPRESSION</name>
      <value [lang="..."]>EXPRESSION</value>
</is-caliente-property-value>
```

3.2.15 is-first-version

Evaluates to ***TRUE*** if this object is the first object in the version history.

```
<is-first-version/>
```

3.2.16 is-latest-version

Evaluates to ***TRUE*** if this object is the latest version in the object's version history.

```
<is-last-version/>
```

3.2.17 is-name

Evaluates to ***TRUE*** if the object's name matches the given ***expression*** as per the designated ***comparison***.

```
<is-name [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-name>
```

3.2.18 is-original-name

Evaluates to ***TRUE*** if the object's *original* name matches the given ***expression*** as per the designated ***comparison***.

```
<is-original-name [comparison="eq"] [lang="..."]>
      EXPRESSION
</is- original-name>
```

3.2.19 is-original-subtype

Evaluates to ***TRUE*** if the object's *original* subtype (object type) matches the given ***expression*** as per the designated ***comparison***.

```
<is-original-subtype [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-original-subtype>
```

3.2.20 is-reference

Evaluates to ***TRUE*** if this object is a **reference** object.

```
<is-reference/>
```

3.2.21 is-subtype

Evaluates to ***TRUE*** if the object's *current* subtype (object type) matches the given ***expression*** as per the designated ***comparison***.

```
<is-subtype [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-subtype>
```

3.2.22 is-type

Evaluates to ***TRUE*** if the object's *archetype* matches the given value. See above for a list of valid *archetype* names.

```
<is-type>ARCHETYPE</is-type>
```

3.2.23 is-variable-empty

Evaluates to ***TRUE*** if the Caliente context variable whose name matches the given ***expression*** as per the given ***comparison*** is empty. An empty variable is any single-valued variable whose value is either **null** or the empty string (""), or a multivalued variable that has no values or has exactly one value and this value is either **null** or the empty string ("").

```
<is-variable-empty [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-variable-empty>
```

3.2.24 is-variable-repeating

Evaluates to ***TRUE*** if the Caliente context variable whose name matches the given ***expression*** as per the given ***comparison*** is a multivalued variable.

```
<is-variable-repeating [comparison="eq"] [lang="..."]>
      EXPRESSION
</is-variable-repeating>
```

3.2.25 is-variable-value

Evaluates to ***TRUE*** if the Caliente context variable whose name matches the given ***expression***, has a value as per the given value ***expression*** using the given ***comparison***, according to the chosen ***cardinality***.

```
<is-variable-value [comparison="eq"] [cardinality="any"]>
      <name [lang="..."]>EXPRESSION</name>
      <value [lang="..."]>EXPRESSION</value>
</is-variable-value>
```

3.3 Condition Groups

3.3.1 AND

This group implements the logical AND-gate. It will only evaluate to ***TRUE*** when all its contained conditions evaluate to ***TRUE***. Otherwise, it evaluates to ***FALSE***.

3.3.2 OR

This group implements the logical OR-gate. It will only evaluate to ***TRUE*** when one or more of its contained conditions evaluates to ***TRUE***. Otherwise, it evaluates to ***FALSE***.

3.3.3 XOR

This group implements the logical XOR-gate. It will only evaluate to ***TRUE*** when an odd number of its contained conditions evaluates to ***TRUE***. Otherwise, it evaluates to ***FALSE***.

For a *true* mutex (mutually-exclusive) gate, look at ***ONEOF***, below.

3.3.4 NOT

This group implements the logical NOT-gate. It will invert the result of its single contained condition (or group). That is to say: if its contained condition (or group) evaluates to ***TRUE***, this group evaluates to ***FALSE***. Conversely, if its contained condition (or group) evaluates to ***FALSE***, this group evaluates to ***TRUE***

3.3.5 NAND

This group implements the logical NAND-gate. It will only evaluate to ***TRUE*** when at least one of its contained conditions evaluate to ***FALSE***. Otherwise, it evaluates to ***FALSE***. Identical to implementing ***NOT-AND***.

3.3.6 NOR

This group implements the logical NOR-gate. It will only evaluate to ***TRUE*** when all of its contained conditions evaluate to ***FALSE***. Otherwise, it evaluates to ***FALSE***. Identical to implementing ***NOT-OR***.

3.3.7 XNOR

This group implements the logical XNOR-gate. It will only evaluate to ***TRUE*** when an even number of its contained conditions evaluates to ***TRUE***. Otherwise, it evaluates to ***FALSE***. Identical to implementing ***NOT-XOR***.

3.3.8 ONEOF

This group implements a gate that evaluates to ***TRUE*** when exactly one of its contained conditions evaluates to ***TRUE***. Otherwise, it evaluates to ***FALSE***.

When the number of contained conditions is 2, ***ONEOF*** and ***XOR*** behave identically. Otherwise, ***ONEOF*** is a true mutex gate, whereas ***XOR*** is not.

4 Transformations

Caliente provides a very flexible and powerful transformation mechanism that supports almost anything that an operator or administrator may wish to do with their data while it's being transferred. There are limitations, however, due to the mechanics that Caliente implements and their inherent restrictions.

4.1 Limitations

4.1.1 Renaming objects

Because Caliente may not always have a full view of the data's end state, the transformation layer is not allowed to rename objects dynamically. This is to avoid generating new collisions as a result of the transformation process since at that point it's not possible to render a full, cohesive, and conclusive view of the data's end state. It is therefore impossible to detect these collisions pro-actively, and thus allowing renames may result in ingestion errors.

In layman's terms, Caliente can't see the future, and renaming files may negatively impact that future with no way for the engine to detect those problems proactively.

4.1.2 Changing object path

This is a corollary to the above point on renaming objects: Caliente can't safely change an object's path without risking filename collisions on the target system.

4.1.3 Changing the contents of data streams

Caliente's transformation only affects object metadata. There is currently no mechanism available to modify an object's data stream (i.e. document contents) during ingestion.

4.2 Sample Syntax

Here is an example XML from an actual transformation implementation used in production:

```
<?xml version="1.1" encoding="UTF-8"?>
<transformations xmlns="http://www.armedia.com/ns/caliente/engine">
	<transformation>
		<if>
			<is-type>DOCUMENT</is-type>
		</if>

		<set-subtype>
			<value>cm:content</value>
		</set-subtype>
		<remove-original-secondary-subtypes />
		<add-secondary-subtype>
			<value>ucm:document</value>
		</add-secondary-subtype>
		<end-transformation />
	</transformation>
```

```
	<transformation>
		<if>
			<is-type>FOLDER</is-type>
		</if>
		<set-subtype>
			<value>cm:folder</value>
		</set-subtype>
		<remove-original-secondary-subtypes />
		<add-secondary-subtype>
			<value>ucm:folder</value>
		</add-secondary-subtype>
		<end-transformation />
	</transformation>
</transformations>
```

The above transformation, when used for ingestion, causes Caliente to change the object type (***subtype***) for every ***DOCUMENT*** object to ***cm:content***, while also removing any associated secondary subtypes (a.k.a. aspects), adding an additional secondary subtype called ***ucm:document***, and stopping the transformation once that's done. It also change the object type (***subtype***) for every ***FOLDER*** object to ***cm:folder***, while also removing any associated secondary subtypes (a.k.a. aspects), adding an additional secondary subtype called ***ucm:folder***, and stopping the transformation once that's done.

This is a more general example of a filter XML, without using specific condition names, to highlight the base structure:

```
<?xml version="1.1" encoding="UTF-8"?>
<transformations xmlns="http://www.armedia.com/ns/caliente/engine">
	<transformation>
		<if>
			<!-- filter conditions -->
		</if>

		<action-1>
			<!-- action 1 parameters -->
		</action-1>
		<action-2>
			<!-- action 2 parameters -->
		</action-2>
		<action-3>
			<if>
				<!-- filter conditions for action 3 -->
			</if>
			<!-- action 3 parameters -->
		</action-3>
		<!-- ... more actions ... -->
```

```
            <action-N>
                  <!-- action N parameters -->
            </action-N>
      </transformation>

      <transformation>
            <!-- ... -->
      </transformation>

      <!-- ... more transformations ... -->

</transformations>
```

Every transformation can contain an ***<if>*** XML to which a filter condition or a filter condition group can be added to limit the occasions the contained transformations are executed. Every transformation block that matches a given object is applied to that object. A transformation block is said to match a given object when the filter conditions within its ***<if>*** evaluate to ***TRUE***. An empty or omitted ***<if>*** element always evaluates to ***TRUE***. The ***<if>*** block ***must*** be the first element in the declaration for each transformation.

4.3 Relationship with Filters

All actions support an ***<if>*** XML element to which a filter condition or a filter condition group can be added to limit the occasions the actions are executed. An empty or omitted ***<if>*** element always evaluates to ***TRUE***. The ***<if>*** block ***must*** be the first element in the declaration for each transformation. Through this mechanism, a high degree of flexibility can be attained in terms of the transformative actions performed upon an object at run-time.

Since Action Groups (see below) also support this functionality, the possible combinations are virtually limitless.

```
<action-name>
      <if><!-- filter conditions --></if> <!-- optional -->
      <!-- ... action configuration elements ... -->
</action-name>
```

4.4 Transformative Action Implementations

4.4.1 abort-transformation

Aborts processing the current object, causing the object to be ***FAILED***.

```
<abort-transformation/>
```

4.4.2 add-secondary-subtype

Adds a ***secondary subtype*** (a.k.a. aspect) to the current object.

```
<add-secondary-subtype>
      <name [lang="..."]>EXPRESSION</name>
</add-secondary-subtype>
```

4.4.3 apply-value-mapping

Applies a value mapping as defined either by Caliente's own internal mechanisms, or explicitly via ***set-value-mapping***. The mapping will be applied to every attribute whose name ***expression*** matches the given ***comparison***, as per the selected ***cardinality***. As mappings are defined as key-value pairs, each attribute's value is compared to the key, and if the current attribute value (as per the ***cardinality***) matches the key exactly, then it's replaced by the mapping's value. If no mappings match, and a ***<fallback>*** has been specified, the expression's value is used instead.

```
<apply-value-mapping>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <attribute-name [lang="..."]>EXPRESSION</attribute-name>
      <type>ARCHETYPE</type>
      <mapping-name [lang="..."]>EXPRESSION</mapping-name>
      <cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
      <fallback [lang="..."]>EXPRESSION</fallback> <!-- optional -->
</apply-value-mapping>
```

4.4.4 clear-value-mapping

Clears a value mapping as defined either by Caliente's own internal mechanisms, or explicitly via ***set-value-mapping***. The value is selected via the given ***type***, and mapping ***name expression***, using the given ***<from>*** element's ***expression*** as the value key.

```
<clear-value-mapping>
      <type>ARCHETYPE</type>
      <name [lang="..."]>EXPRESSION</name>
      <from [lang="..."]>EXPRESSION</from>
</clear-value-mapping>
```

Alternatively, you can clear the value mapping based on the mapped value using a ***<to>*** element:

```
<clear-value-mapping>
      <type>ARCHETYPE</type>
      <name [lang="..."]>EXPRESSION</name>
      <to [lang="..."]>EXPRESSION</to>
</clear-value-mapping>
```

4.4.5 copy-attribute

Creates a new attribute with the name specified by the ***<to>*** element expression as an exact copy of the attribute in the ***<from>*** expression. If the object already contains that target attribute, its value will be replaced.

```
<copy-attribute>
      <from [lang="..."]>EXPRESSION</from>
      <to [lang="..."]>EXPRESSION</to>
</copy-attribute>
```

4.4.6 copy-variable

Creates a new Caliente context variable with the name specified by the ***<to>*** element expression as an exact copy of the variable in the ***<from>*** expression. If the target Caliente context variable already exists, its value will be replaced.

```
<copy-variable>
      <from [lang="..."]>EXPRESSION</from>
      <to [lang="..."]>EXPRESSION</to>
</copy-variable>
```

4.4.7 custom-action

Invoke a custom action written in Java, whose fully-qualified class name is specified by the given ***expression***. The class must implement the ***com.armedia.caliente.engine.dynamic.Action*** interface, and must be available at launch time by adding JARs to Caliente's library path.

There must also be an instance of ***com.armedia.caliente.engine.dynamic.ActionFactory***, for which the invocation of ***acquireInstance(String)*** with the custom class's fully qualified name will yield a valid ***Action*** instance. This class will be located using ***java.util.ServiceLoader***.

```
<custom-action>
      <class-name> EXPRESSION </class-name> <!-- should evaluate to a fully-qualified java class name -->
</custom-action>
```

4.4.8 end-transformation

Ends the transformation run for the current object, allowing it to continue through the Caliente workflow.

```
<end-transformation/>
```

4.4.9 load-external-metadata

Changes pending…

4.4.10 map-attribute-value

Maps attribute values. An attribute is selected for value mapping if its name matches the given ***name*** as per the given ***comparison***. Once the attribute is selected for mapping, which of its values will be mapped is determined by both the given ***cardinality***, and their values matching the given ***case*** elements.

An attribute value matches a ***case*** element if its value matches the case's ***value*** expression as per the given case's ***comparison***. If the value matches, its replaced with the case's ***replacement*** expression. Only the first case mapping that matches the attribute value is applied. If no case elements match, and there's a ***default*** defined, then that value is used as a replacement instead (use wisely!).

There is no limit to the number of ***case*** elements that can be defined.

```
<map-attribute-value>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
      <cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
```

```
      <case [comparison="COMPARISON"]>
            <value [lang="..."]>EXPRESSION</value>
            <replacement [lang="..."]>EXPRESSION</replacement>
      </case>
      <!-- ... more <case> elements ... -->
      <default [lang="..."]>EXPRESSION</default> <!-- optional -->
</map-attribute-value>
```

4.4.11 map-original-subtype

Maps an object's subtype to a new value based on its original value, as opposed to its current value. An object's subtype may have been modified from its original one already by way other transformations. This transformation allows the operator to use the original data regardless of what may have happened to the object during the transformation pass.

Each ***case*** element is compared to the object's original subtype, and the first one that matches as per the given ***value*** and ***comparison*** is selected. Once a ***case*** is selected, the object's current subtype is set to be the result of matched case's ***replacement*** expression.

If no case matches, and a ***default*** expression is provided, its value is used instead.

```
<map-original-subtype>
      <case [comparison="COMPARISON"]>
            <value [lang="..."]>EXPRESSION</value>
            <replacement [lang="..."]>EXPRESSION</replacement>
      </case>
      <!-- ... more <case> elements ... -->
      <default [lang="..."]>EXPRESSION</default> <!-- optional -->
</map-original-subtype>
```

4.4.12 map-principal

Applies the principal mapping tables to individual attributes. An attribute is selected for mapping if its name matches any of the given ***name*** elements as per the given ***comparison***. Once the attribute is selected for mapping, which of its values will be mapped is determined by both the given ***cardinality***, and the ***type*** of principal mapping table to use. There are three types of principals: ***USER***, ***GROUP***, and ***ROLE***.

If no mappings in the chosen mapping table match, and there's a ***fallback*** defined, then that value is used as a replacement instead (use wisely!).

There is no limit to the number of ***name*** elements that can be defined.

```
<map-principal>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
      <!-- ... more <name> elements ... -->
      <type>PRINCIPAL_TYPE</type>
      <cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
      <fallback [lang="..."]>EXPRESSION</fallback> <!-- optional -->
</map-principal>
```

4.4.13 map-subtype

Maps an object's subtype to a new value based on its current value. Each ***case*** element is compared to the object's current subtype, and the first one that matches as per the given ***value*** and ***comparison*** is selected. Once a ***case*** is selected, the object's current subtype is set to be the result of matched case's ***replacement*** expression.

If no case matches, and a ***default*** expression is provided, its value is used instead.

```
<map-subtype>
	<case [comparison="COMPARISON"]>
		<value [lang="..."]>EXPRESSION</value>
		<replacement [lang="..."]>EXPRESSION</replacement>
	</case>
	<!-- ... more <case> elements ... -->
	<default [lang="..."]>EXPRESSION</default> <!-- optional -->
</map-subtype>
```

4.4.14 map-variable-value

Maps a Caliente context variable's values. A variable is selected for value mapping if its name matches the given ***name*** as per the given ***comparison***. Once the variable is selected for mapping, which of its values will be mapped is determined by both the given ***cardinality***, and their values matching the given ***case*** elements.

A variable's value matches a ***case*** element if its value matches the case's ***value*** expression as per the given case's ***comparison***. If the value matches, its replaced with the case's ***replacement*** expression. Only the first case mapping that matches the variable value is applied. If no case elements match, and there's a ***default*** defined, then that value is used as a replacement instead (use wisely!).

There is no limit to the number of ***case*** elements that can be defined.

```
<map-variable-value>
	<comparison>COMPARISON</comparison> <!-- optional, default EQ -->
	<name [lang="..."]>EXPRESSION</name>
	<cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
	<case [comparison="COMPARISON"]>
		<value [lang="..."]>EXPRESSION</value>
		<replacement [lang="..."]>EXPRESSION</replacement>
	</case>
	<!-- ... more <case> elements ... -->
	<default [lang="..."]>EXPRESSION</default> <!-- optional -->
</map-variable-value>
```

4.4.15 remove-attribute

Removes an object's attribute where its name matches the given ***name*** as per the given ***comparison***.

```
<remove-attribute>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
</remove-attribute>
```

4.4.16 remove-original-secondary-subtypes

Remove's all of an object's original secondary subtypes from its current subtype list, leaving intact any currently-assigned subtypes that aren't within that set. If a subtype that is part of the original subtype set is explicitly added to the object, it will also be removed.

```
<remove-original-secondary-subtypes/>
```

4.4.17 remove-secondary-subtype

Removes a secondary subtype from the current object as per the given ***name*** and ***comparison***.

```
<remove-secondary-subtype>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
</remove-secondary-subtype>
```

4.4.18 remove-variable

Removes a Caliente context variable from the current transformation context as per the given ***name*** and ***comparison***.

```
<remove-variable>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
</remove-variable>
```

4.4.19 rename-attribute

Renames an object's attribute as per the given ***from*** and ***to*** expressions. The attribute's name will be matched exactly.

```
<rename-attribute>
      <from [lang="..."]>EXPRESSION</from>
      <to [lang="..."]>EXPRESSION</to>
</rename-attribute>
```

4.4.20 rename-variable

Renames a Caliente context variable as per the given ***from*** and ***to*** expressions. The variable's name will be matched exactly.

```
<rename-variable>
      <from [lang="..."]>EXPRESSION</from>
```

```
        <to [lang="..."]>EXPRESSION</to>
</rename-variable>
```

4.4.21 replace-attribute

Apply a regular expression replacement to object attribute values. An attribute is selected for value replacement if its name matches the given ***name*** as per the given ***comparison***. Once the attribute is selected for replacement, which of its values will be acted upon is determined by both the given ***cardinality***, and their values matching the given ***regex*** element.

If an attribute's value matches the given ***regex***, then its replacement will be calculated via the given ***replacement*** expression. If there is none, then the empty string is used as a replacement. Regular expressions may include capturing groups, and these may in turn be referenced in the replacement as defined in **java.util.regex.Pattern**.

```
<replace-attribute>
        <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
        <name [lang="..."]>EXPRESSION</name>
        <cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
        <regex [caseSensitive="boolean"] [lang="..."]>EXPRESSION</regex>
        <replacement [lang="..."]>EXPRESSION</replacement> <!-- optional -->
</replace-attribute>
```

4.4.22 replace-secondary-subtype

Apply a regular expression replacement to object secondary subtype values. All secondary subtype values are considered for replacement. If a secondary subtype value matches the given ***regex***, then its replacement will be calculated via the given ***replacement*** expression. If there is none, then the empty string is used as a replacement. Regular expressions may include capturing groups, and these may in turn be referenced in the replacement as defined in **java.util.regex.Pattern**.

```
<replace-secondary-subtype>
        <regex [caseSensitive="boolean"] [lang="..."]>EXPRESSION</regex>
        <replacement [lang="..."]>EXPRESSION</replacement> <!-- optional -->
</replace-secondary-subtype>
```

4.4.23 replace-subtype

Apply a regular expression replacement to the object's subtype value. If a subtype value matches the given ***regex***, then its replacement will be calculated via the given ***replacement*** expression. If there is none, then empty string is used as a replacement. Regular expressions may include capturing groups, and these may in turn be referenced in the replacement as defined in **java.util.regex.Pattern**.

```
<replace-subtype>
        <regex [caseSensitive="boolean"] [lang="..."]>EXPRESSION</regex>
        <replacement [lang="..."]>EXPRESSION</replacement> <!-- optional -->
</replace-subtype>
```

4.4.24 replace-variable

Apply a regular expression replacement to a Caliente context variable's values. A context variable is selected for value replacement if its name matches the given ***name*** as per the given ***comparison***. Once the context variable is selected for replacement, which of its values will be acted upon is determined by both the given ***cardinality***, and their values matching the given ***regex*** element.

If a context variable's value matches the given ***regex***, then its replacement will be calculated via the given ***replacement*** expression. If there is none, then the empty string is used as a replacement. Regular expressions may include capturing groups, and these may in turn be referenced in the replacement as defined in **java.util.regex.Pattern**.

```
<replace-attribute>
      <comparison>COMPARISON</comparison> <!-- optional, default EQ -->
      <name [lang="..."]>EXPRESSION</name>
      <cardinality>CARDINALITY</cardinality>  <!-- optional, default any -->
      <regex [caseSensitive="boolean"] [lang="..."]>EXPRESSION</regex>
      <replacement [lang="..."]>EXPRESSION</replacement> <!-- optional -->
</replace-attribute>
```

4.4.25 reset-original-secondary-subtypes

Resets the object's assigned secondary subtypes to be exactly the original set of secondary subtypes the object was stored with. This effectively undoes all changes to the set of secondary subtypes for the given object.

```
<reset-original-secondary-subtypes/>
```

4.4.26 set-attribute

Sets an object attribute as per the tiven ***name***, ***type***, and with the given ***value***. Only single-valued attributes are supported at this time.

```
<set-attribute>
      <name [lang="..."]>EXPRESSION</name>
      <type>DATA_TYPE</type> <!-- optional, default STRING -->
      <value[lang="..."]>EXPRESSION</value>
</set-attribute>
```

4.4.27 set-subtype

Sets the object's current subtype to the result of the given ***name*** expression.

```
<set-subtype>
      <name [lang="..."]>EXPRESSION</name>
</set-subtype>
```

4.4.28 set-value-mapping

Defines a new value mapping that is made persistent throughout the entire extraction or ingestion process. The value is associated with the given archetype from ***type***, and as per the given ***name*** expression. If the ***type*** element is not given, the current object's archetype is used. The value

mapping is defined with a key as specified by ***<from>***, and a value as specified by ***<to>***. Caliente uses this mechanism to track mappings between source object IDs, and target IDs.

```
<set-value-mapping>
       <type>ARCHETYPE</type>
       <name [lang="..."]>EXPRESSION</name>
       <from [lang="..."]>EXPRESSION</from>
       <to [lang="..."]>EXPRESSION</to>
</set-value-mapping>
```

4.4.29 set-variable

Sets a Caliente context variable as per the tiven ***name***, ***type***, and with the given ***value***. Only single-valued context variables are supported at this time.

```
<set-variable>
       <name [lang="..."]>EXPRESSION</name>
       <type>DATA_TYPE</type> <!-- optional, default STRING -->
       <value[lang="..."]>EXPRESSION</value>
</set-variable>
```

4.5 Action Groups

Action groups are a special type of action which allows the operator to organize actions within a transformation script as a unit, and apply a common filter to all actions. Actions within the group will be executed one by one, and each action may in turn declare their own filter for further refinement of the execution logic. As with all other actions, the ***<if>*** block is optional, and if it's empty or omitted it will evaluate to ***TRUE***.

Action groups may also contain other action groups, and this nesting ability provides for arbitrarily complex logic to be applied during transformation.

```
<group>
       <if><!-- filter conditions --></if> <!-- optional -->
       <!-- ... contained actions ... -->
</group>
```

5 Attribute Mapping Syntax

… Still needs documenting …