Java/Java Component

Mulesoft Development Standards

Version 2.0

Java/Java Component
Mulesoft Development Standards

Version 2.0

Table of Contents

Disclaimer:

This Document intends to accurately provide aa substantial information regarding Java standards and guidelines for Mulesoft development. The information contained in this document is confidential and is for Mulesoft Consultants and project stakeholders only. This document must only be used internally within The Organization, and must not be used or made publicly available without permission. If you have any concern kindly contact the authors in the Document Control section.

1. Document Control

1.1. Document History

Date	Version	Amendments	Author
05/21/2020	1.0	Initial Draft	Julius Wilfred Leal
06/02/2020	1.0.1	Added component format specifications	Joanne Agustin
06/18/2020	1.0.2	Added Java Files Structure	Joanne Agustin
07/12/2020	1.0.3	Added Spring configurations	Joanne Agustin
07/30/2020	1.0.4	Applied WSL-CG template (Sharepoint compatible)	Joanne Agustin
08/04/2020	2.0	Applied CG Template and added a disclaimer	Marc Timothy Morales

1.2. Reviewers

Name	Role	Reference/Signature	Date
Marco Mate	Team Lead CoE		
Janos Rai Geronimo	Team Lead CoE		

1.3. Approvers

Name	Role	Reference/Signature	Date
John Anthony Bernardo	Director of Consulting - CoE		

2. Introduction

The purpose of this document is to provide substantial information regarding Capgemini | WhiteSky Labs - Java Standards that can help consultants to create a meaningful process for project development and design.

3. Java Standards

Java component in Mule 4, for some, may look like it reaches the point of no use since the evolution of dataweave. Dataweave language has been improving since its first release that it can practically do anything useful for integration. Because of this, many developers do integration with only dataweave as data transformers. This is not wrong but may also be not right.

According to MuleSoft:

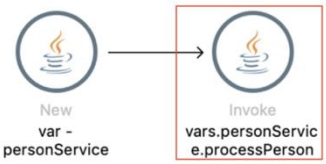
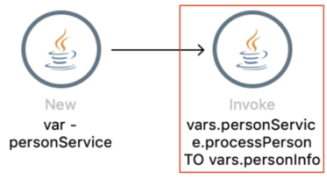
- If you want to extract, query, transform, or otherwise work with data in your flows, Dataweave expressions and transforms are the recommended tool.
- If you want to write custom logic, instantiate Java objects, or call arbitrary methods, MuleSoft recommends that you encapsulate this code into scripts or classes that can be injected and tested easily.

  By this definition, we cannot simply say Java components are of no use.

3.1. Operations

It's recommended to specify either a short description, or the class name and/or method, on the component's name to give the instantaneous idea of what the java component does.

Operation	Format	Description	Example
New	**Set as Payload:** - *{short_desc}* **<***{TypeName}***>** **Set as Variable:** - var - *{variable_name}* **<***{TypeName}***>**	Create a new instance of a class.	**Payload:**  **Variable Assignment:** 

Invoke	- *{instance}.{method}* **[** TO var.*{var_name}***]** - *{short_desc}* **[** TO var.*{var_name}***]** * Use var.*{var_name}* if the target is a variable. * Unlike the **New** java component, the type/class of the instance is not included in the name to (1) shorten it, (2) the instance method name is already included in the naming convention.	Invoke a java method given an instance.	**Payload:**  **Variable Assignment:** 
Invoke Static	*{classname.method | short_desc}***[** TO var.*{var_name}***]** * Use var.*{var_name}* if the target is a variable.	Invoke a static java method.	**Payload:**  **Variable Assignment:** 
Validate Instance	*{instance | object}* IS A [*{simple_classname}*]	Validate what class is a given instance.	

3.2. Format

Invoke | Invoke Static: ***{short_desc}***

New: Validate type: ***{instance | object}*** IS A [***{simple_classname}***]

3.3. Best Practice

1. Java Components are only recommended for custom logics. It is considered best practice to separate business logic from the workflow.
2. In creating packages, separate classes based on their uses like beans and POJOs, service interfaces, service implementations and utilities.

   Use pom **group ID** as a base package:

```
pom.xml
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http:/
3      <modelVersion>4.0.0</modelVersion>
4
5      <groupId>com.standards.examlpe</groupId>
6      <artifactId>standards-example</artifactId>
7      <version>1.0.0-SNAPSHOT</version>
8      <packaging>mule-application</packaging>
```

```
▼ src/main/java
  ▶ com.standards.example.bean
  ▶ com.standards.example.service
  ▶ com.standards.example.util
```

   Place service interfaces on service package and services implementations on service implementation package. Name the service implementation the same as its interface and add **impl** suffix:

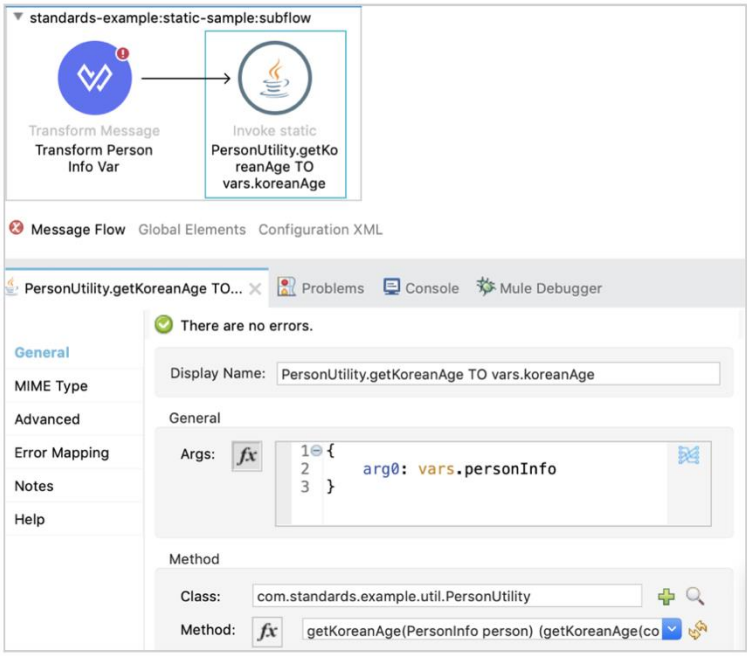
```
▼ src/main/java
  ▶ com.standards.example.bean
  ▼ com.standards.example.service
    ▶ PersonService.java
  ▼ com.standards.example.service.impl
    ▶ PersonServiceImpl.java
  ▶ com.standards.example.util
```

3. Make bean and pojo names short but descriptive and add **info** suffix.

```
▼ src/main/java
  ▼ com.standards.example.bean
    ▶ PersonInfo.java
```

4. Dataweave Module classes should follow dataweave standards. Please refer to the Modularization Java Module section in dataweave standards. (Add Link Here)

5. If the method is not dependent on instance variables (Usually Utility Class methods), make your method static so that there is no need to create an instance of the class in mule flow.



6. Take advantage of Object Orientation.

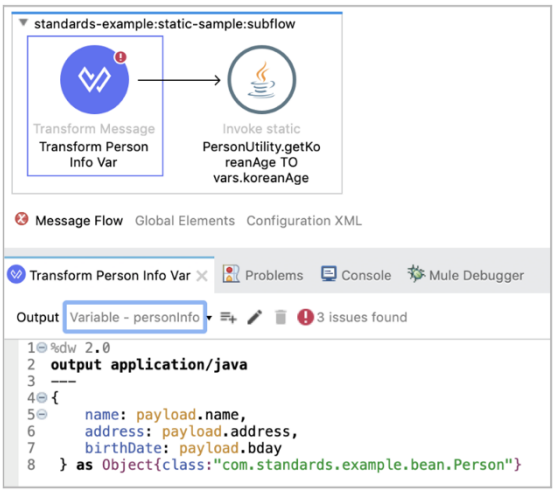
   Create methods accept Objects instead of accepting its properties and assembling it on your method.

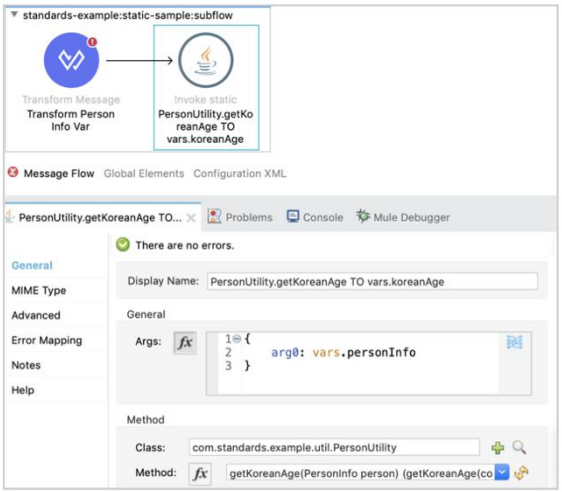
```
public class PersonServiceImpl implements PersonService{

        //bad practice
        @Override
        public void processPerson(String name, String address,Date birthDate) {
                PersonInfo person = new PersonInfo();
                person.setAddress(address);
                person.setBirthDate(birthDate);
                person.setName(name);
                //do something
        }

        //good practice
        @Override
        public void processPerson(PersonInfo p) {
                //do something
        }
}
```

Use Object Coercion for Java Object parameters:





4. Java File Structure

Java classes and interfaces should use the pom.xml **group id** as its base package.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.acme.retail</groupId>
    <artifactId>sys-retailer-abc-api</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>mule-application</packaging>

    <name>sys-retailer-abc-api</name>
```

Sub-package(s) must be added to further group the classes/interfaces by usage or type.
Recommended, but not limited to, sub packages and file conventions are:

- util/*Util.java
- bean/*Bean.java
- info/*Info.java
- service/*Service.java
- service.impl/*ServiceImpl.java
- exception/*Exception.java
- configuration/*Configuration.java

For test classes, apply the same structure suffixed with 'Test' under src/main/test/java/<package>.

5. Spring Configuration

Name	Format	Example
Spring Configuration	Spring_Config**[**_*{module_name}*] * This config loads the Spring xml beans definded in configuration file(s): **spring-beans[**-*{module_name}*]**.xml**. * Note that the filename under *file* attribute must be **spring-beans[**-*{module_name}*]**.xml** (location: **src/main/mule/common/**). This is the XML-based configuration data that describes or defines the Spring beans.	Spring_Config_Security Spring_Config **Code:** *<spring:config* *name="Spring_Config_Authentication"* *files="spring-beans-security.xml" />Å*
Security Manager, Security Provider	Security_Provider_Config**[**_*{auth_provider}*] * This enables support for Spring security by associating the Spring Authentication manager bean name that must be defined under **spring-beans[**-*{module_name}*]**.xml**.	Security_Provider_Config Security_Provider_Config_LDAP Security_Provider_Config_JDBC **Code:** <spring:security-manager> <spring:delegate-security-provider name="Security_Provider_Config_LDAP" delegate-ref="authenticationManager" /> </spring:security-manager>
Authorization Filter	Authorization Filter **[** - *{role}*] * Optionally add role	Authorization Filter Authorization Filter - Admin 

For details, see Spring Module documentation in Mulesoft.

6. References

- General Naming Conventions - https://capgemini.sharepoint.com/:w:/r/sites/MSABLInitiatives/Shared%20Documents/Standards/Capgemini%20Format/Naming%20Standards%20Quick%20View%20-%20Mulesoft%20Development%20Standards%20v2.0.docx?d=w8fdfbeb2063b4fa8be7b881082c30fdd&csf=1&web=1&e=k2bEJ9
- Mule Java Module - https://docs.mulesoft.com/connectors/java/java-module
- Mule Java Operation - https://docs.mulesoft.com/connectors/java/java-reference#operations
- Argument Transformation - https://docs.mulesoft.com/connectors/java/java-argument-transformation
- Using Java Throwable - https://docs.mulesoft.com/connectors/java/java-throwable

About Capgemini

A global leader in consulting, technology services and digital transformation, Capgemini is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. It is a multicultural company of over 200,000 team members in more than 40 countries. The Group reported 2018 global revenues of EUR 13.2 billion.

Learn more about us at www.capgemini.com

People matter, results count.