MuleSoft Coding Conventions
DRAFT

Table of Contents

.

1. Introduction

It is always our effort to provide clean and consistent codebase, and to be productive as a development team. This document provides general coding standards to be followed in Integration Development using MuleSoft.

It is required to share and to be in agreement with client before implementing these coding standards, and if there are any client provided guidelines, they should override the guidelines in this document.

1.1 Scope

This document outlines the naming convention guidelines for Mule Integraiton Development projects.

2. Mule Development Naming Conventions

Mule Component	Naming Details
Mule Project Name	The project is created for high level Business Process to be implemented. The project name should explain the high level **business purpose** of the project, and an optional version identifier if you want to maintain multiple versions of same project. The project name shall follow lower case as below syntax. **Syntax: <clientname/program>-<businesspurpose>-v1.0** Examples: 1. The Mule Integration that allows to manage order processing customer-orderprocessing 2. The Mule Integration that allows to manage products with version projectatlas-orderprocessing-v1.0
Mule Flow Name	The Mule configuration file name should explain the purpose of a flow within a Businesss Process. Use combination of Verb + Noun to define the purpose of a flow. Each of the Mule configuration file should refer to either of below 5 types:- 1. Main flow (Each interface in the project will have a main flow) 2. Error handling flow (Error handling strategy defined in this flow. Only one flow per project) 3. Config / Global flow (All the global connector and other configurations. Only one flow per project) 4. Common flow (Any reusable sub-flows across other interfaces. Only one flow per project) 5. API flow (This will appear only while using API kit router. This flow should have only routing logic, actual interface should be coded in individual interface main flow. API will also be only one flow per project) The name should follow lowercase, hyphen or underscore separated

.

	Syntax: <purposeof>-<flowdetails> Examples for the above mentioned flows: 1. The HTTP service to get token from SFDC get-oauthtoken-sfdc.xml 2. The main flow can contain source and target information app1-itemdetails-app2.xml 3. The RAML based main API kit router flow api.xml 4. The reusable sub-flows across other interfaces clientname-orderprocessing-common.xml 5. Similarily the config and error handling flows are customer-orderprocessing-errorhandling.xml customer-orderprocessing-config.xml
Endpoint Name(Global/Local)	The endpoint name should explain the operation that endpoint is performing. The name can follow “lower Camel Case” or “lower case hyphen separated”, but should be consistent across the project. **Syntax: <Action><Business Entity being processed>** Examples: 1. The file inbound endpoint to read employee data readEmployeeFlow 2. The file outbound endpoint to write product data write-product-privateflow 3. The HTTP Endpoint to get Sales Order from Request getSalesOrderSubflow
MUnit Test Suite	The Mule MUnit configuration file name should follow same name as the corresponding main interface flow. The name should follow lowercase, hyphen or underscore separated **Syntax: <purposeof>-<flowdetails>-testsuite** Example: 1. Test suite for get token from SFDC flow get-accountdetails-sfdc-testsuite.xml
MUnit Test Case	The individual test cases should provide test case description along with success or failure testing. The name should follow “lower Camel Case” or “lower case hyphen separated”. **Syntax: <positive/negative><Description><Test>** Examples: 1. Test case for one sample success test

	successGetAccountDetailsTest 2. Test case for one sample failure test failure-getaccountdetails-test
Property Files	The property files are usually created at the program level in a common project and usually injected as dependencies into individual projects under the program. However they can also be created at the project level, if the environment related properties are significantly different or not common across projects under the program. In anycase, the property files should be created at environment level and should follow the lower case syntax as below. They should be added as reference in the project config file using property placeholder. **Syntax: <clientname/program>-<env>.properties** Examples: 1. Properties file at the entire client / program level. customer-prod.properties 2. Properties file at the individual project level. customer-orderprocessing-dev.properties
Property File Values	The properties should be grouped based on the functionality and properties should start with a contextual name. None of the passwords should be put in plain text, they should be encrypted. The actual property should follow lower case syntax with DOT separated as below:- **Syntax: <application/entity>.<endpoint>.<property>=<value>** Examples: 1. Property for a bank salesforce Endpoint. bank123.sfdc.host=[host_name] bank123.sfdc.password=[encrypted_password] 2. Property for an application DB endpoint retek.db.host=[host_name] retek.db.port=[port_number]
Transformation Name (In-Built abstract Transformer as well DataMapper)	The transformation name should provide Input Entity Business Name, Input Structure, Output Entity Business Name, and Output Structure in the following pattern. The name should follow lower Camel Case. **Syntax: <Input Entity Name><Input Structure>2<Output Entity Name><Output Structure>** Examples: 1. The transformation of Sales Order OAGi model to SFDC Json model. salesLeadSOAP2LeadJSON 2. The transformation of Product CSV File to SAP Product Model.

	productCSV2ProductXML
Scopes	The scopes name should provide the entity on which the processing depends. The name should follow lowe Camel Case. **Syntax: <Scope Name><Entity Name>** Examples: 1. To process each record retrieved from Select Query from Database. forEachOrderRecord 2. To cache parameters if they already exist. cacheUserName
Components	The component name should explain the operation that component is performing on high level. The name should follow lower Camel Case. **Syntax: <Action><Business Entity being processed>** Examples: 1. To get OAUTH token by making a Rest call using Java. getOAuthToken 2. To make a SOAP call to external Order Processing web service. processOrder
Filters	The filter name should explain the action as well as the reference to the variable on which the processing depends. The name should follow lower Camel Case. **Syntax: <Action><Business Entity being processed>** Examples: 1. To filter the product records on isActive variable to process only the active. processActiveProduct 2. To filter specific message payload from HTTP inbound endpoint filterFavicon
Flow Control	The flow control name should provide the control activity with the criteria on which the processing depends. The name should follow lower Camel Case. **Syntax: <Action><Business Entity being processed>** Examples: 1. To spilt an xml containing multiple products. splitProducts 2. To choose the destination file endpoint depending on product family routeOnProductFamily
Global DataSource	The data source name should explain database business name, RDBMS type of

	database (Oracle, MS-SQL etc). The name should follow lower Camel Case. **Syntax: <Database Business Name><RDBMS Name>DS** Example: 1. Customer Core Oracle Database customerCoreOracleDS
Connector	The connector name should explain the datasource it connects to. The name should follow lower Camel Case. **Syntax: <DataSource Name-withoutDS>Connector** Example: 1. The connector for Customer Core Oracle Database customerCoreOracleConnector
Cloud Connector(Local)	The clould connector name should explain the operation it performs. The name should follow lower Camel Case. **Syntax: <Action><Business Entity being processed>** Example: 1. Create SalesLead using SalesForce Connector createLead
Global Component Configurations	The configuration name should explain the context object it configures. The name should follow lower Camel Case. **Syntax: <Action><Context Object>** Example: 1. The script configuration uses Groovy script to calculate total order payment amount calculatePaymentAmount
Logger	The logger component name should provide the name of currently processing entity and whether it is a request or response. The name should follow lower Camel Case. **Syntax: Log<Processing entity>Request/Response** Example: 1. Log incoming SalesLead request in OAG format. logOAGProcessSalesLeadRequest

2.1 General Naming Conventions

The below general naming convetion rules to be followed:

- Business processes or entities should be singular rather than plural. Example: CreateLead (NOT CreateLeads)
- Provide description of a flow in the flow description section.
- The name should avoid abbreviations. Example: OrderServiceFlow (NOT OrderServFlow)
- All MUNIT functional test clasess should have suffix with 'TestCase'. Example: CreatLeadTestCase, DeleteLeadTestCase
- All the variables names (session, flow, invocation, inbound, outbound) must be in mixed case starting with lower case. This rule is applicable to all message properties as well.
- Underscores and other special characters should NOT be used in variable names, method names or class names
- 'is' prefix should be used for boolean variables and methods. Example: isSet, isVisible, isFinished, isFound, isOpen
- Negated boolean variable names must be avoided. Example: boolean isError(NOT: isNoError)
- Do not use "mule" as prefix for project or deployment artifact name.
- Avoid being too specific whenever not required to limit re-use potential.
- Increment major version if you break backwards compatibility.

3. Common Development Naming Conventions

Component	Naming Details
JMS	The JMS resource names must be unique per resource type (ex. queues, topic, connection factories, and JNDI names). The name for JMS component should follow below pattern. This pattern provides unique identity for a JMS component, when there are multiple business groups and domains using a same JMS Server within an enterprise. The business group and domain should explain the logical business group and domain within an enterprise. Two different (business group + business domain) can have (message type + version + resource type) that can share the same name. This business group and domain within it is owner of the component. **Syntax:** **jms/<biz group>.<biz domain>.<message type>.<V99.9>.<resource type>** The syntax shall follow all lower case. The version number provides unique identity for a Queue or Topic, when there are multiple versions of the same application having diverse message models, or services exchanging messages.

	Do not include "application name" that provides this component, or "consumer" or "publisher" of the message into naming. These can change over time (ex. change or addition in consumer & publisher) although contract (and pattern) for business domain remains the same. Examples: 1. jms/ck.ckint.cif.dim.v1.0.queue 2. jms/ck.ckint.catalog.v01.0.queue
REST API	The well named APIs are insightful and so easier to use. The RESTful APIs is collection of URIs that is called over HTTP, and provides JSON or XML representation of resource. In RESTful API each piece of information is exposed as a resource having its own address or URI, so name each resource as noun as opposite to verb or action. URI should follow a consistent hierarchical structure to ease understanding and meaning of the resource for the consumer of an API. Examples: To create a new catalog POST http://www.elearning.com/catalogs To read a specific catalog with catalog Id #123 GET http://www.elearning.com/catalogs/123 The same URI would be used for PUT and DELETE, to update and delete catalog respectively. To get all registrations for a catalog 123 GET http://www.elearning.com/catalogs/123/registrations The same URI with POST would be used for creating registration for a catalog 123. To get all payments against registration 456 of a catalog 123 GET http://www.elearning.com/catalogs/123/registrations/456/payments Below are some of the rules for URI design - All REST service names should use strict lower case names - All REST service request and response fields should use strict lower case names - Build URI using hierarchy of entities (resources) that is being operated

	upon rather than the operation being performed - Use domain entitites names and their relation to define consistent hierarchy - Use plural of an entitiy - Use nouns, not verbs. The power of REST comes through the fact that there is a limited verb set (operations – GET, POST etc) combined with a large set of nouns (or resources) - Services offering a uniquely identifiable resource via a key should use basic REST notation (ex. /catalogs/(cataloged)) - Service having mandatory arguments over GET should have them as path variables (ex. /catalogs/(fromdate)/(todate)) - Services offering optional search/filtering capabilities should use query parameter (ex ?key1=value&key2=value) - Keep it intuitive. URIs should be human readable and guessable **REST API Versioning** There are multiple techniques to define version for REST API like - Include version into API URI - Use version attribute into HTTP Header - Use domain name into HTTP Header TODO: Best Technique for versioning for REST API or not to use Versioning.
SOAP WEB SERVICE	SOAP uses WSDL, XML document that represents the SOAP service contract including protocol, endpoint, methods and message structure. WSDL document uses the following elements in the definition of service: • **Message - an abstract, typed definition of the data being communicated.** Web service Message name should clearly indicate what type of data it carries. The Web service message name should follow Camel Case. **Syntax: <Verb><DataType><Request | Response | Fault>** Examples: 1. Messages to perform addition of product to product catalog. addProductRequest addProductResponse addProductFault 2. Messages to get product information. getProductInfoRequest getProductInfoResponse getProductInfoFault

- **Operation - an abstract description of an action supported by the service**.

Web service operation names should clearly indicate what they do from the client's point of view. The Web service operation name should start with verb and follow Camel Case.

Syntax: <Verb><Operation Purpose>

Examples:

1. To add product to product catalog.
   addProduct

2. To get product Information.
   getProductInfo

- **Port Type - an abstract set of operations supported by one or more endpoints.**

Port type name should be similar to service name appended with word Port. Port type name should follow Camel Case.

Syntax: <Service name><Port>

Examples:

1. Service to manage Product data.
   ProductPort

- **Service - a collection of related endpoints.**

The Web service name should be a business noun, concept or process.
Avoid verbs when naming Web services.

Examples:

1. To create Product service
   ProductService

Avoid names such as CreateProductDataService.

Web Service Versioning:

Web Service versioning is achieved by changing the version of namespace in WSDL and XSD.

It is NOT recommended to use version number in the Service Endpoint URL as this may result into frequent updates to the Endpoint URL configured in service consumers programs.

Major Version Change Scenarios:
The term major update is reserved for updates that break existing service consumers – i.e. clients, or invalidates previously shared (and consumed) documents.

Major version updates have to be shared with all consumers of the service to avoid the failure of existing interfaces that are using the service.

Updates to an XML Schema document are considered to result in a major update if:

- There are changes to the type of a local element
- There are changes to a local element or an element reference requiring that it become mandatory (i.e. changing the definition from optional to require).
- Adding or removing an enumeration value
- Renaming or removing a global type or element
- Changing the type of a global element

Updates to a WSDL document are considered to result in a major update if:

- Updates are made to existing binding, service or portType definitions
- Updates are made to a message element with a reference to a schema type
- message, message part, portType, binding is removed from service

Major updates may NOT be backwards compatible for existing clients. Hence always share update WSDL/XSD with all consumers whenever major version change.

Syntax: <Service Namespace></V><Version Number><.0>

Example:
http://www.productdata.com/retail/integration/service/ProductDataService/V2.0

Minor Version Change Scenarios:
The term minor update is reserved for backward-compatible updates of an interface which does not break existing clients or existing documents such that they remain valid if continuing to use a minor update to an existing document.

Minor version updates have to be shared with only those consumers of the service for whom the change has been implemented. All other consumers can still continue to use the previous version of the service.

Updates to an XML Schema document are considered to result in a minor update if:

	• There are changes to a local element or element reference making it optional (i.e. changing from the definition from required to optional). • Adding a global element or type • Adding documentation or comments Updates to a WSDL document are considered to result in a minor update if: • A new operation, possibly with message and message parts definitions is added • A new portType, binding or a service to a WSDL is added • A new, optional, behavior is added without changing the message signatures or types Minor updates MUST be backwards-compatible for existing clients. **Syntax: <Service Namespace></V><Version Number><.n>** Example: http://www.productdata.com/retail/integration/service/ProductDataService/V2.5

4. JAVA Components Conventions

Mule IDE studio should be configured for Sonar coding guideline tools. Use default Sonar rules.
Please refer to the below link for Java Coding Guidelines. This link also includes XML Schema Best Practices.

- https://github.com/mulesoft/mule/blob/3.7.2/STYLE.md
- https://docs.mulesoft.com/mule-contributors-guide/v/3.4/mule-coding-conventions

5. References

Sr. No.	Document Name
1	https://docs.mulesoft.com/mule-contributors-guide/v/3.4/mule-coding-conventions
2	MuleSoft Alliance Partner Portal documentation
3	http://www.w3.org/TR/wsdl
4	