**MULE ESB BEST PRACTICES FOR MULE PROJECT**

1. **Naming conventions.**

Mule flow by itself and its components should have such names, which would give a clear understanding of what function they have. Standard names of components are allowed for transformation elements, but for other elements it is preferred to use more coherent names. For example:

* flow name should provide insight into processes in this flow;
* the basis, according to which the condition is built can be pointed for Choice element;
* if it is possible, addresses and the way of data input/output can be pointed for connectors;
* For logging elements, if it is possible, the name should represent which information is being logged.

There is no specific rules for naming Conventions for Mule Flows. It’s always the best if you name them from maintainability and readability perspective.

**Here are the best practices:**

**Naming Standards and Conventions:**

* Application name: In lowercase(Ex: myapplication)
* Config.xml file name (app/...): Camel case. ex. myApplication.xml
* Flow name: Camel case. Reflects the purpose of a flow and ends with word Flow i.e myApplicationFlow
* Subflow name: Camel case. Reflects the purpose of a sub flow and ends with word sub\_flow/subFlow i.e connectDBSub\_flow/connectDBSubFlow
* Variable name: Camel case i.e bookPrice
* Transformation file name should begin with 'xform' word. Example: xform<Operation>.xsl
* XSD nodes containing repetitive elements should be Suffix with List/Type i.e OrderListType
* Invoking Flows/ Subflows should be called with Operation name

**Package Naming standard and namespace:**

* Common data objects: com.<orgname>.<project/program name>.api.dataobjects i.e com.capgemini.bmrs.api.dataobjects
* Data objects with respect to API: com.<orgname>.<project/program name>.api.<specific\_api>.dataobjects i.e: com.capgemini.bmrs.api.b602.dataobjects
* API Service package: com.<orgname>.<project/program name>.bmrs.api.service
* API Service Implementation package: com.<orgname>.<project/program name>.bmrs.api.service.impl
* API util package: com.<orgname>.<project/program name>.bmrs.api.util
* Common Project util package: com.<orgname>.<project/program name>.bmrs.util
* Common project Service package: com.<orgname>.<project/program name>.bmrs.services
* SOAP Client classes package: com.<orgname>.<project/program name>.bmrs.<servicename>.client
* Mock SOAP Client classes package: com.<orgname>.<project/program name>.bmrs.<servicename>.mock.client
* Other package java class name: as per java naming conventions

**Documentation and Logging:**

* Comments needs to be provided at the flow documentation, with a brief description.
* All flows and services defines should have an entry log, exit log, unique transaction id log and error log.
* All flows defines should have a custom business events for each logs.
* Check the document flows label Nodes, sequences, Case constructs are readable.
* All if constructs should be labeled and documented logically.
* All identifiable business keys should be logged in the each log.
* Comments for all Java code logic is must.

**Code Consistencies:**

* Validate the input for mandatory fields.
* Use Properties file to make port, host, etc configurable
* Check for error code sequence is orderly defined and appropriate error category(Business/Technical) is used for new Error Codes
* All Operations should have a response header with code, type and description.
* All functional flows that can be reused should be defined as an independent flows and invoked.
* Maintain all the constants in the constants/property file.

1. **The whole code**, which can be used in other projects,**should be brought out to the specific project/library.** Common utility should be added as JAR in library.
2. **Standard in using property:**

* All hardcode should be brought out to the settings files.
* All settings should be described in comments in detail.

1. **Powerful expression language in Mule**, which allows making Mule components more dynamic. For example, **Mule Expression Language** allows dynamically defining addresses for http component. For this purpose, in the host field of http connector it is enough to use expressions of the following form:

* host="#[xpath3('/request/host')]" – in the case if different data come to the input of the http connector in the xml format.
* host=”#[payload]” – in the case if the address needs to be withdrawn from payload.
* host=”#[flowVars.host]” - in the case if the address needs to be withdrawn from flow variables.  
  Mule expressions can be used almost in all text-fields of the Mule components.

1. **For every flow** should be realized its own **exception strategy** for correct exception handling.

Example of mule xml:

<set-payload value="#[exception]" doc:name="Set Payload as Exception"/>

<component doc:name="IncomingExceptionHandler">

<spring-object bean="IncomingExceptionHandler"/>

</component>

NOTE: IncomingExceptionHandler is a Java component (class) where the handling of the appeared exception and the formation of the answer can take place.

1. [**SFTP connector**](https://docs.mulesoft.com/mule-user-guide/v/3.8/sftp-connector) best practices.

Besides the standard settings of the sftp-server connectivity, also can be useful the following settings:

* Response timeout.
* Reconnection policy.
* Archive directory.

1. [**Dataweave transformation**](https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave)is a powerful data process tool, which allows converting data from/into CSV, Excel, XML, JSON, Java Object and other formats. One can specify mappings and default values, add conditional logic, filter data, group and build transformation chains.  
   The following operations on arrays are possible: selectors (for choosing elements), operators (map, concat, contains and etc.)  
   Also Dateweave language for simple transformations can be used.

For example:

#[dw("{'payload': payload, 'interface':'CompInvoice'}","application/json")]

1. **Work with Queues.**

Mule allows working with different queues, which settings have a lot of common (name of the queue, operation – read/record), but with its specific features:

* [**VM**](https://docs.mulesoft.com/mule-user-guide/v/3.8/vm-transport-reference)is an in-memory queue, support transactions, but it can be used only for the message exchange among threads of an application, and not among several applications.
* [**AnypointMQ**](https://docs.mulesoft.com/anypoint-mq/) is a queue, provided by Mulesoft for the commercial usage and integrated into the Anypoint platform. It does not require a specific server for installation, but it does not support transactions.
* [**ActiveMQ**](https://docs.mulesoft.com/mule-user-guide/v/3.8/activemq-integration)is a popular open-source solution, which supports JMS. It supports transactions and message exchange among applications but it requires a specific server for installation.

1. **Profiling service**.

A standard java/oracle utility jvisualvm.exe can be used for profiling. It can be found by the following way Windows C:\Program Files\Java\jdk1.8.0\_65\bin\jvisualvm.exe(in the bin catalogue of the installed jdk). This utility allows tracking resources which are used by java applications). The application can be loaded and resources spends can be tracked under the concrete loadings.

1. **MUnit tests**

Mule flows should be covered by MUnit tests. Testing should be carried out on the different levels. If flow contains several subflows, then it is desirable to test each of subflows separately and the flow, which uses them. In this case MUnit tests are a very good example of using flow and its separable parts. Basic algorithms need to be covered by unit tests. We recommend to use [Data Driven Testing](https://jazzteam.org/en/technical-articles/data-driven-testing/) if it is possible.

1. **Load testing.**

JMeter can be used for Mule application load testing. This tool allows checking application work in the conditions of a big amount of incoming requests from many users and providing testing results in the convenient form for an analysis.

1. **Xml formatting.**

* An extra component should be created for a smart/convenient text displaying in xml logs. When using this component it should be taking into consideration that it can change xml by adding to its spaces/inputs etc. that, as a result, might make xml invalid for web-services and various utilities for working with xml, so before logging one should save the source xml, and after logging one should return the previously saved xml.
* Sometimes, in xml request there are symbols of & lt type instead <> brackets, for correct work with xml, it is necessary to do the operation of data symbols change to <>. In this regard an additional class may be used, which will change escape symbols.

1. **Deploy of the application.**

Mule Plugin for Maven server can be used to deploy the application to the CloudHub. Such parameters, as Mule version, user name, password, environment, business group, application name, etc. can be configured.

1. **Spring Bean Components**.

All components are recommended to perform by spring beans.

Example:

