Anypoint Naming Conventions

- 1 Naming Conventions
- 2 Mule Anypoint Folder Structure

Naming Conventions

Below table captures the naming convention to be followed while executing Anypoint platform based projects.

#	Item	Description	Naming Convention	Examples
1	Application / Project Name for Experience Layer	Mule Application or Project name for user experience layer, where 'x' stands for user eXperience.	*x-<hyphen separated, all lower case program/project name>[-<hyphen separated, all lower case entity/function name>]*	*x-loyalty-program* *x-loyalty-program-pos*
2	Application / Project Name for Process Layer	Mule Application or Project name for process/orchestration layer, where 'p' stands for process.	*p-<hyphen separated, all lower case program/project name>[-<hyphen separated, all lower case entity/function name>]*	*p-loyalty-program* *p-loyalty-program-pos*
3	Application / Project Name for System Layer	Mule Application or Project name for system connectivity layer, where 's' stands for system.	*s-<hyphen separated, all lower case program/project name>[-<hyphen separated, all lower case system/function name>]*	*s-loyalty-program* *s-loyalty-program-comarch*
4	Application / Project Name (non-API)	Mule Application or Project name for non-API projects like common services, framework, etc.	*<hyphen separated, all lower case program/project name>[-<hyphen separated, all lower case system/function name>]*	*audit-logger* *audit-logging-framework* *audit-logger-service* *common-logging* *common-utilities*
5	RAML File Name	For API layer a RAML file is created in API Designer. This needs to follow standard naming convention.	*<x/p>-<hyphen separated, all lower case program/project name>[-<hyphen separated, all lower case entity/function name>]-api.raml*	*x-loyalty-program-api.raml* *p-loyalty-program-pos-api.raml*
6	Mule Project Structure	Any Mule Project should comply with a satndard project structure	*Refer sheet "Mule Folder Structure"*	*Refer sheet "Mule Folder Struc*
7	Mule Config - Global Elements	All connector configurations, property placeholders, etc. should be defined in a separate Mule Config file where no flows should be defined. It should only contain global elements.	*global-config.xml*	*global-config.xml*
8	Mule Config - Default	Whenever a Mule project is created a default Mule config xml file gets created with same name as the project name	*<Project Name>.xml*	*x-loyalty-program.xml* *p-loyalty-program.xml*
9	Mule Config - Default - API	Whenever a Mule project is created a default Mule config xml file gets created with same name as the RAML file, if provided, else project name. If RAML file provided, please change the name to project anme.	*<Project Name>.xml*	*x-loyalty-program.xml* *p-loyalty-program.xml*
10	Mule Config - Exception Handler	There should be a separate Mule Config only containing exception flows across the project.	*<Project Name>-error-handling.xml*	*x-loyalty-program-error-handlir* *p-loyalty-program-error-handlir*

11	Mule Config - Additional	Any additional Mule config xml files created should follow a standard naming convention	*<Project Name>-<hyphen separated, all lower case function name>.xml*	*p-loyalty-program-common-se* *p-loyalty-program-create-custo*
12	Flow Name - Main Flow	A Main flow is the one which exposes inbound interface to external applications. One Mule config can hold more than 1 main flow.	*<Project Name>-main[-<hyphen separated, all lower case function name if more than 1 main flows>]*	*p-loyalty-program-main* *p-loyalty-program-main-create-* *s-loyalty-program-comarch-ma* *s-loyalty-program-comarch-ma*
13	Flow Name - Main Flow - API	All flows including main flow would be created by default when a project is created with APIKit components enabled and/or RAML file. One Mule config can hold more than 1 main flow. The flow can be named with a verb (as per the HTTP method) followed by optional domain and the noun and then in the end will come the keyword 'flow', all small case separated by hyphen.	*<Project Name>-main[-<hyphen separated, all lower case function name if more than 1 main flows>]*	*x-loyalty-program-main* *x-loyalty-program-api-main-cre*
14	Flow Name - Subflow/non-main flows	The flow other than main flows are the ones which do not expose inbound interface to external applications, they could only be called from other flows using flow-ref.	*<Project Name>-flow-<hyphen separated, all lower case function name>*	*p-loyalty-program-flow-update-* *p-loyalty-program-flow-create-* *s-loyalty-program-comarch-flo* *s-loyalty-program-comarch-flo*
15	Flow Name - Subflow/non-main flows - API	The flow other than main flows are the ones which do not expose inbound interface to external applications, they could only be called from other flows using flow-ref.	*<Project Name>-flow-<hyphen separated, all lower case function name>*	*x-loyalty-program-flow-update-* *x-loyalty-program-api-flow-crea*
16	Flow Name - Exception/Error Handling	A common error handler should be used for the project to improve re-use. This service could raise Custom Business Events, Alerts, send emails or process errors from Message Queue.	*<Project Name>[-<hyphen separated, all lower case function name, if more than one error handler>]-error-handling-flow*	*x-loyalty-program-error-handli*
17	Flow Components	All components a flow/subflow comprises of should follow standard naming convention	*<hyphen separated, all lower case component type>-<hyphen separated, all lower case task definition>*	*set-payload-to-request* *logger-entry* *logger-exit* *logger-process-customer-detai*

18	Logger Statements - Critical touch points	All critical touch points / interfacing points in a flow should be logged to normal logging as well as Audit logging with INFO level. The different touch points are ENTRY into the flow, EXIT from the flow, CALLING external services from the flow, CALLED external services response received, PROCESSING intermidiate logging statements should follow a standard and consistent log statements and log level. NOTE: In case of Normal Logging, please do not log the entire payload for this, unless small or critical, always log only important fields out of the request/response which are enough for tracking/troublshooting. In case of Audit Logging, you could log the entire payload for each request and response, but a word of caution for batch processing.	*"ENTRY >> ApplicationName: #[app.name], FlowName: #[flow.name], RequestId: #[message.inboundProperties.pRequestId != null ? message.inboundProperties.pRequestId: sessionVars.sRequestId], MessageId: #[message.mRootId], CorrelationId: #[message.mCorrelationId], Message: #[message.payloadAs(java.lang.String)]" with log level as "INFO" and category of logging as "<com>.<mycompany>.<module1>"*	*INFO com.whishworks.pos EN x-loyalty-program, FlowName: RequestId: nnnnn, MessageId: Message: abcdefg*
19			*"EXIT << ApplicationName: #[app.name], FlowName: #[flow.name], RequestId: #[message.inboundProperties.pRequestId != null ? message.inboundProperties.pRequestId: sessionVars.sRequestId], MessageId: #[message.mRootId], CorrelationId: #[message.mCorrelationId], Message: #[message.payloadAs(java.lang.String)]" with log level as "INFO" and category of logging as "<com>.<mycompany>.<module1>"*	
20			*"CALLING >> ApplicationName: #[app.name], FlowName: #[flow.name], RequestId: #[message.inboundProperties.pRequestId != null ? message.inboundProperties.pRequestId: sessionVars.sRequestId], MessageId: #[message.mRootId], CorrelationId: #[message.mCorrelationId], Target: #[message.mTargetName], Message: #[message.payloadAs(java.lang.String)]" with log level as "INFO" and category of logging as "<com>.<mycompany>.<module1>"*	
21			*"CALLED << ApplicationName: #[app.name], FlowName: #[flow.name], RequestId: #[message.inboundProperties.pRequestId != null ? message.inboundProperties.pRequestId: sessionVars.sRequestId], MessageId: #[message.mRootId], CorrelationId: #[message.mCorrelationId], Target: #[message.mTargetName], Message: #[message.payloadAs(java.lang.String)]" with log level as "INFO" and category of logging as "<com>.<mycompany>.<module1>"*	
22			*"PROCESSING >> ApplicationName: #[app.name], FlowName: #[flow.name], RequestId: #[message.inboundProperties.pRequestId != null ? message.inboundProperties.pRequestId: sessionVars.sRequestId], MessageId: #[message.mRootId], CorrelationId: #[message.mCorrelationId], Message: #[message.payloadAs(java.lang.String)]" with log level as "INFO" and category of logging as "<com>.<mycompany>.<module1>"*	
23	Flow Variables	Variables defined and used withtin a flow scope	*camelCase as in Java variables but always starts with "v"*	*vLocationId vIsStoreCache vStoreId*
24	Flow Properties (Inbound/Outbound)	Properties, inbound and outbound, from message header	*camelCase as in Java variables but always starts with "p"*	*pLocationId pIsStoreCache pStoreId*
25	Flow Session Variables	SHOULD NOT BE USED UNLESS CRITICAL NECESSITY	*camelCase as in Java variables but always starts with "s"*	*sLocationId sIsStoreCache sStoreId*
26	Message Properties	Properties passed as part of message	*camelCase as in Java variables but always starts with "m"*	*mLocationId mIsStoreCache mStoreId*

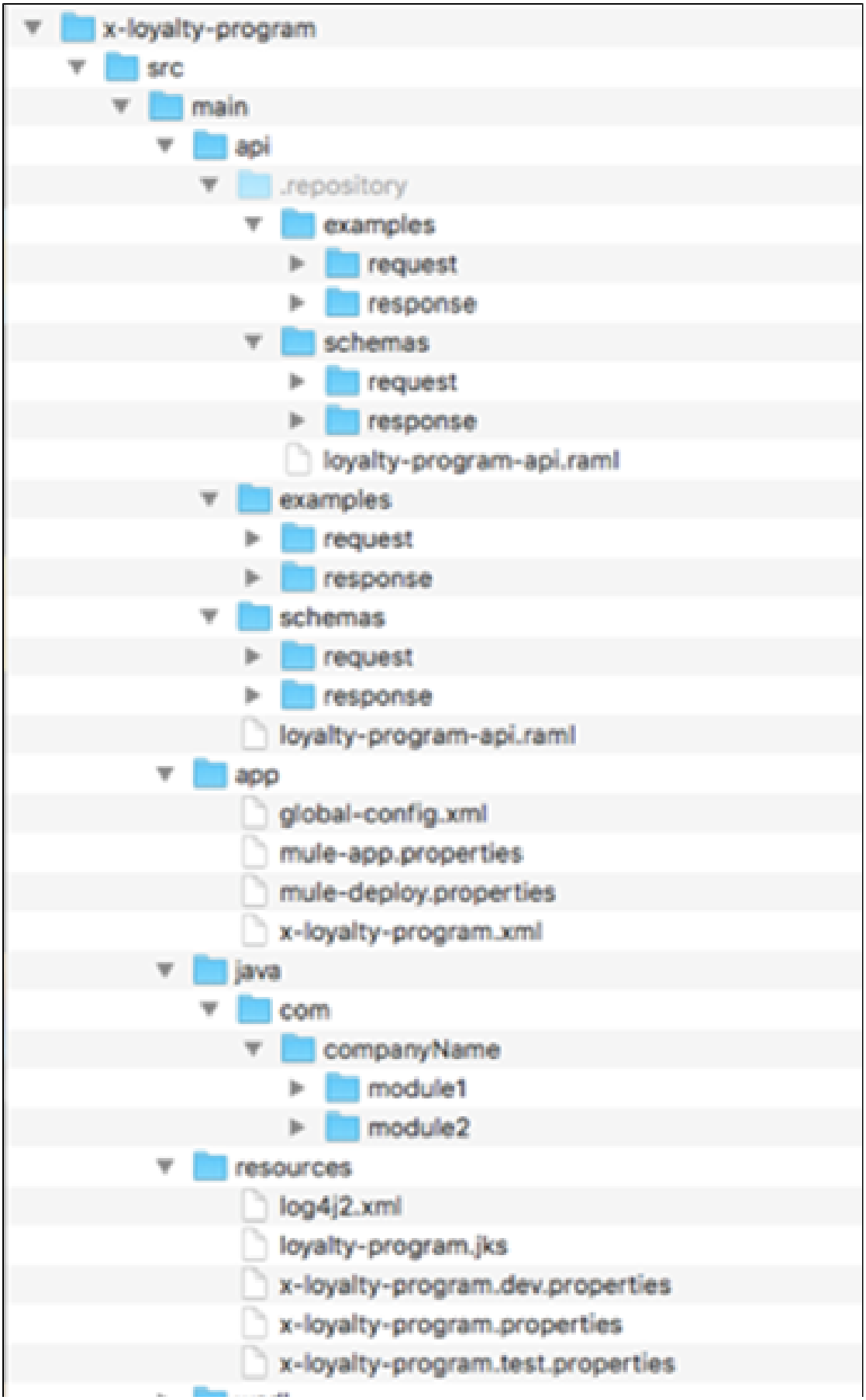
27	Application Properties	Properties used by application	*FQN (Fully Qualified Name) with lower case separated by dot/period. In cases more than one properties for similar systems/instances then use either numeric like 1,2,.. or name of the function.*	*db.audit.insert.query* *db.audit.insert.order.query* *salesforce.user.id* *salesforce.user.token* *salesforce.customer.user.id* *salesforce.customer.user.toke*
28	Application Property File	Property file holding all the application properties should follow standard naming convention and structure	*/src/main/resources/properties/<RAML file name / Project Name>-<environment>.properties*	*x-loyalty-program-prod.propert* *x-loyalty-program-dev.propertie* *x-loyalty-program-test.propertie* *x-loyalty-program-common.pro* *properties)*
29	Data Weave Files	DataWeave Mapping files, if complex and needs to be externalized, should follow standard naming convention and a structure	*/src/main/resources/dwl/<hyphen separated, all lower case functionality name>.dwl*	*order-xml-to-json.dwl* *price-xml-to-map.dwl*
30	Integration Examples	Examples of source and destination formats used in DataWeave, or any other places	*/src/main/resources/examples/<hyphen separated, all lower case functionality name>.<xml/json/csv/psv>*	*create-order-request.xml* *list-customers-response.csv*
31	API Examples	Examples of API request and response used in RAML	*/src/main/api/examples/<hyphen separated, all lower case functionality name>.<xml/json>*	*create-order-request.xml* *update-order-response.json*
32	Integration Schemas	XSD, JSON, etc. schemas of source and target applications or systems in integration	*/src/main/resources/schemas/<hyphen separated, all lower case functionality name>.<xsd/json>*	*order.xsd* *customer.json*
33	API Schemas	XSD, JSON, etc. schemas used in the RAML definition	*/src/main/api/schemas/<hyphen separated, all lower case functionality name>.<xsd/json>*	*order.xsd* *customer.json*
34	WSDL	SOAP WSDL files of API if exposed as SOAP services	*/src/main/wsdl/<hyphen separated, all lower case API/service name>.wsdl*	*stores.wsdl* *orders.wsdl*
35	MUnit Test Suite	MUnit Test Suite for testing MUnit flows and coverage	*<Corresponding Mule Config Name>-test-suite.xml*	*x-loyalty-program-test-suite.xm* *p-loyalty-program-test-suite.xm*
36	MUnit flow names	Each MUnit Test Suite will have different MUnit Test flows for testing all Mule Flows	*<Corresponding Flow Name>-[<hyphen separated, all lower case functionality name>]-test[n]*	*x-loyalty-program-main-test* *x-loyalty-program-api-main-cre* *x-loyalty-program-api-main-cre*
37	API Service naming	API Service name should be a concrete noun or entity and not abstract		*stores* *orders*

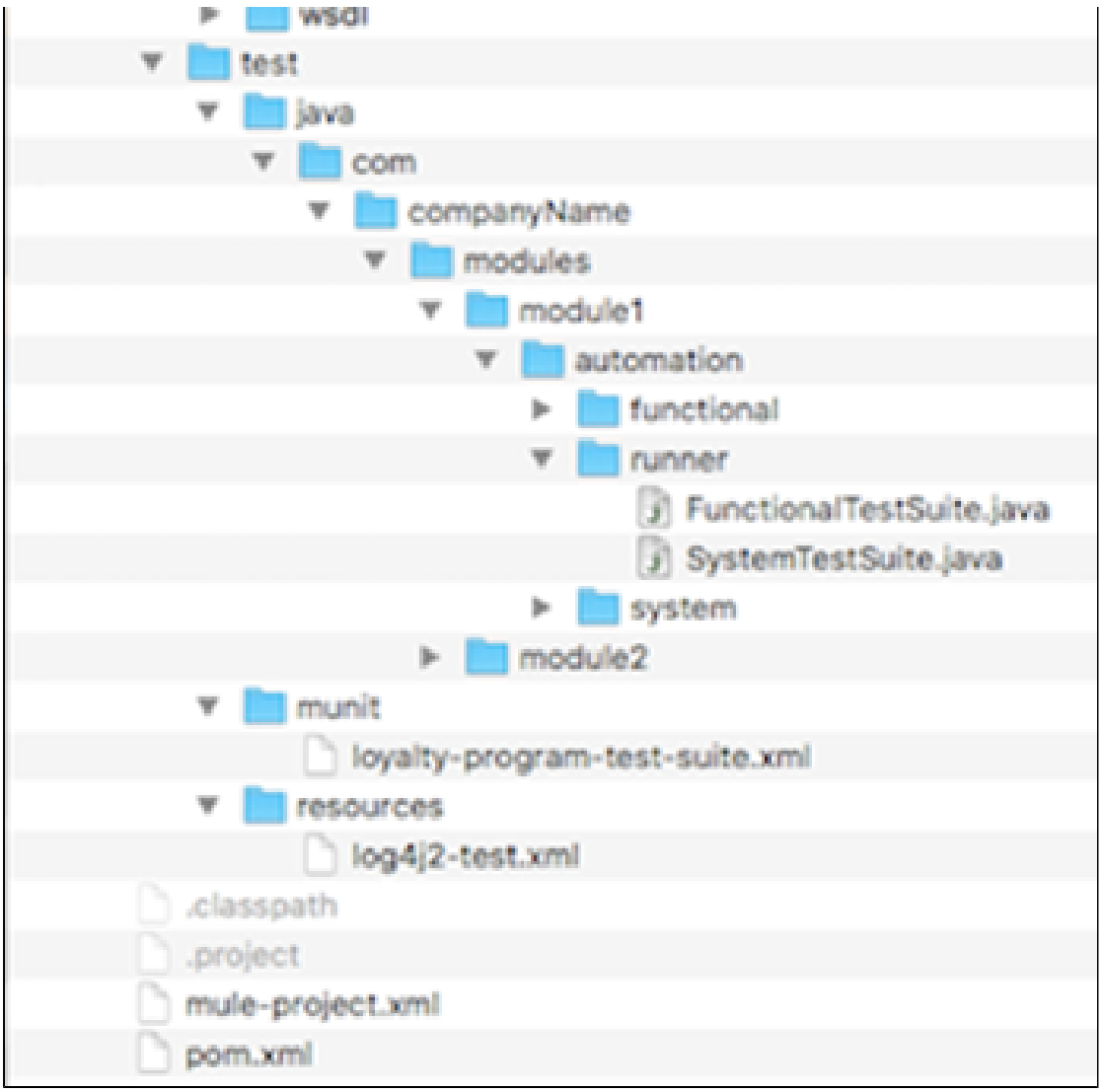
38	URI	The API resources should be named using concrete nouns and not verbs in their base URL. The verbs (or actions) should be implemented as HTTP methods POST (create), GET (Retrieve/Read), PUT/POST (Update), DELETE (Delete). Avoid making a resource name too abstract like /services. The endpoint URL for a service, which is the address that a consumer uses to locate and invoke a service, must reflect the major version number only. This is because the inclusion of a minor version number would cause the address of the service to change for each minor version release, thus making the service appear to have been removed, from an existing consumer's perspective. For the next backward compatible release there is no change in the URI, but for a backward incompatible major release the URI should be updated to next major version.	*/v<n>/<domain or category>[/<sub-domain or sub-category>]*	*/v1/stores* */v1/stores/locations* */v1/orders* */v2/orders* */v1/security/tokens* */v1/customers?id=12345 (with customer and DELETE for delt* */v1/customers (with HTTP met customer and PUT/POST for u body)*
39	API URL - complete (Base URI + API URI)	Complete URL for an API should be structured for a better categorization of APIs across functional domains, regions, access (public or private) and helps define relationships (hierarchical). A good URI also helps to govern the lifecycle of the API through versioning practices.	*<http/https://[<env>.][<access>.]<company web domain>.<region>/<API/Service Name>/v<n>/<resource/entity/noun>[/<resource-id>/][<sub-resource>/][?<query-params>]*	*GET https://sandbox.api.whish 5* *GET https://api.whishworks.co ddress*
40	API - Filtering	For search type of API's which return list of entities/resources, the filtering should be implmented in standard way	*HTTP GET method with query parameters*	*GET /orders?state=shipped*
41	API - Sorting	For search type of API's which return list of entities/resources, the sorting should be implmented in standard way by using generic query parameter 'sort'	*/v<n>/<resource>?sort=[-]<field to sort>[,<another field to sort>]* *- indicates descending and no sign indicates ascending order*	*GET /orders?sort=-date,produ*

42	API - Partial Resources	In some cases, the consumer might not need all the fields of a resource. To allow for obtaining only a partial resource the API URL could be design to take a list of 'fields' as a query parameter, and return only the fields that are includes in that list.	*/v<n>/<resource>?fields=<field to return>[,<another field to return>]*	*GET /orders/1?fields=date,tota*

Mule Anypoint Folder Structure

Below captured is a snapshot of Mule Anypoint folder structure


x-loyalty-program
src
main
api
.repository
examples
request
response
schemas
request
response
loyalty-program-api.raml
examples
request
response
schemas
request
response
loyalty-program-api.raml
app
global-config.xml
mule-app.properties
mule-deploy.properties
x-loyalty-program.xml
java
com
companyName
module1
module2
resources
log4j2.xml
loyalty-program.jks
x-loyalty-program.dev.properties
x-loyalty-program.properties
x-loyalty-program.test.properties


wsdl
test
java
com
companyName
modules
module1
automation
functional
runner
FunctionalTestSuite.java
SystemTestSuite.java
system
module2
munit
loyalty-program-test-suite.xml
resources
log4j2-test.xml
.classpath
.project
mule-project.xml
pom.xml