API & Development Naming Standards



Contents

[Introduction 3](#_Toc527378385)

[Definitions 3](#_Toc527378386)

[Summary 3](#_Toc527378387)

[Files & Content 3](#_Toc527378388)

[Palette Components 4](#_Toc527378389)

[Mule Project Names 5](#_Toc527378390)

[Mule Configuration Files 6](#_Toc527378391)

[File Names 6](#_Toc527378392)

[Standard Files 6](#_Toc527378393)

[Functional Files 7](#_Toc527378394)

[Mule Flows 7](#_Toc527378395)

[Mule Variables 7](#_Toc527378396)

[ARM Domain Names 7](#_Toc527378397)

[Custom HTTP Header Names 8](#_Toc527378398)

[Queue & Topic Names 8](#_Toc527378399)

[API URLs 9](#_Toc527378400)

[CloudHub Shared Load Balancer 9](#_Toc527378401)

[Cloudhub VPC Dedicated Load Balancer or Company Load Balancer 9](#_Toc527378402)

# Introduction

This document describes the naming standards for API and application development on the Anypoint Platform and Mule runtimes.

# Definitions

* **Mule Project:** Name of the Mule project for the application. This is the name of the top-level folder of the Mule app and is stored in source control. When deployed to CloudHub, it is also the name of the application’s zip file.
* **Runtime Manager Domain:** This is the name of the domain in Anypoint Runtime Manager to which the application is deployed. The Mule project name and domain name are NOT linked and don’t have to be named the same thing. One domain is linked to only one Mule app instance. This name must be globally unique across all applications, environments, and clients in Runtime Manager.
* **URL:** This is the Mule application’s API URL.
* **API:** This is the API defined in API Designer & API Manager.
* **App**: This is the implementation running in Runtime Manager. It can be linked to an API in API Manager but does not have to be.
* **ARM**: Anypoint Runtime Manager
* **SoR**: System of record. This refers to the back-end non-mule systems that are the systems of record of the data flowing through APIs.

In API-Led design, there are different application types, depending where the app lines up in the API layers. The application types are listed below and will be referenced in other sections.

* **Experience**: consumer apps in the experience layer.
* **Process**: business orchestration apps in the process layer.
* **System**: apps in the system layer in wrap systems of record.
* **Service**: common service.
* **Batch**: application that does batch or independent processing. Usually triggered by scheduler and not API.

# Summary

A name for any item in Anypoint eco system should follow the guidelines below.

* Descriptive: a name should clearly describe the purpose or mechanism of the item being named.
* Concise: a name should be brief with a recommended length less than 32 characters. Although this is not length limit on a Mule flow or Mule palette activity, frequently using long names would make the visuals inside MuleSoft Studio harder to visually process.
* Consistent with Tools: a name should be consistent with what MuleSoft tools already offer. For example, both MuleSoft Design Center and APIkit both generate some technical artifacts. The defined naming convention should not significantly deviate from what these MuleSoft tools generate.
* Follow the default: a name should follow the “all lower cases with dash and descriptive” pattern, if the item naming is not described in this document.

Following the defined naming standards will make the Mule projects, APIs, flows and other items more readable and understandable.

## Files & Content

The table below shows the general pattern for the files and contents within a Mule project. Specific items are detailed in subsequent sections.

|  |  |
| --- | --- |
| **Category** | **General Pattern** |
| **Mule Projects** | lower case, being descriptive, hyphen |
| **Maven Group Id** | com.<company>.<owning-business-entity>.<code-module-name> |
| **Mule config xml files** | lower case, hyphen  {significant-name}.xml |
| **RAML Files** | lower case, separated by hyphen  {significant-name}.raml |
| **Mule properties files** | lower case, separated by hyphen  {application}-${env}.yaml |
| **Global Elements** | lower case, separated by hyphen  {significant-name} |
| **Dataweave Scripts** | camel case  {significantName}.dwl |
| **Mule Flows** | File name + camel case  {file-name}-{significantDescription} |
| **API Base Path** | /v<version number>/\* |

## Palette Components

Starting from Studio 7, once a developer renames a palette, the palette type name is displayed on top of the display name in gray font. Therefore, the developer does not need to keep the palette activity type in the component name if not required to be descriptive.

|  |  |  |
| --- | --- | --- |
| **Component** | **Description** | **Example** |
| **DataWeave/ Transform** | **Display Name:**<Purpose of the transformation>   * Example: Prepare XML Request for SOAP EAI |  |
| **Set Payload** | **Display Name:**  Set <content name of payload>   * + Example: Set Pricing |  |
| **Choice** | **Display Name:** <Descriptive name for the choice logic>. Ask the question.  Examples:   * + Boolean: Is null?   + Case: Pricing type? |  |
| **For each** | * **Display Name:**FOREACH <Element type in the Collection>   Example: FOREACH commodity |  |
| **Logger** | * **Display Name:**<Purpose of this logger>   + Example:     - Response from Argus pricing call |  |

# Mule Project Names

A Mule project is the application’s source files used in MuleSoft Anypoint Studio and stored in source control. The project should be named for the service it provides, including the type of service. The top-level folder name and project name in source control should take the form below. It should be checked into source control upon creation.

The purpose portion of the name below contains different information depending on the application type. LOB usually only appears in batch or system APIs but those do not have to be specific to a LOB. Separate all words with hyphens.

* **Experience**
  + Pattern: <consumer grouping>-exp
  + Example: vendors-orders-exp
* **Process**
  + Pattern: <lob>-<enterprise data/process entity>-proc
  + Example: commercials-orders-proc
* **System**
  + Pattern: <lob>-<system of record>-sys
    - Add information model or other categorization to name if multiple APIs for single SoR.
  + Example: commercials-sfdc-sys
* **Service**
  + Pattern: <common service name>-service
  + Example: property-encryption-service, notification-service
* **Batch**
  + Pattern: <lob>-<functionality>-bat
  + Example: commercials-accounts-sync-bat

This pattern matches the ARM domain naming convention without the company and environment portions. This helps developers understand the project to ARM domain mapping easily.

Best Practices

* Set Git repository name to the same value as the Mule project name.
* Set Mule project’s Maven artifactId to the same value as the Mule project name.
* Limit the Mule project name to 30 characters. CloudHub has a limit of 42, 11-12 of which are taken by the unique naming standard used for it (prefix & suffix to project name).

# Mule Configuration Files

## File Names

Mule configuration files are the XML files in the Mule project under src/main/mule that define the Mule application. You should create different files for different functionality in a Mule app. This keeps the flows separated so it is easy for maintenance and allows for moving files to different projects if necessary. Generally, the inbound (API), processing, and outbound, global configuration, and error handling portions are put in different files.

The naming pattern is flexible, though the file should be named for the functionality of the flows within. Each file should only contain flows whose purpose is denoted by the file’s name.

Application XML file names are lowercase and separated by hyphens or underscores.

### Standard Files

These are files that every project will have. There can be multiple of each type as necessary. If that is the case, add a post-fix to distinguish between them.

* api-routes.xml
  + Contains the APIKit and routing. Routing flows consist of logging and flow references to flows in other files that implement the functionality.
* error-handling-api.xml
  + Contains the top-level error handling for the API.
* global.xml
  + Contains all the global configurations for the app
  + Global declarations include connector configs and processing strategies.

Functional Files  
These files are created for new functionality added to the app. Each API operation should get its own functional file.

* Syntax: *< functionality>.xml*
* Examples:An API for products that can create, modify, and delete would at least have the files below.
  + create-products.xml
  + update-products.xml
  + delete-products.xml

## Mule Flows

Flow names are lower camel-case. The name of the flow should be prefixed with the name of the file containing the flow. Pattern: <Mule config file name>-<flow name in camel case>. Example: a flow that reads a csv file in the create-products.xml file is named *create-products-readCsvFile*.

One best practice is to have a main orchestration flow, *create-products* in our example, that encompasses the high-level flow and contains flow references to specific functionality. This is like programming styles used in languages like Java. This should be the first flow in the file and have the same name as the file, so it is very clear the purpose of the flow and that it is the starting point.

## Mule Variables

Variables are lower camel-case, such as customerId, xmlPayload, and clientEmailAddress.

Best Practices

* Delete unused content (flows, processors, etc) before committing to version control. This includes commented-out items.
* All flows, sub-flows, components, processors, scopes need brief, descriptive names. Do not leave any default names.
* Do not use any acronyms unless globally-known and industry standard, such as SFDC for salesforce.
* Items should be named such that reading the flows in Studio gives the reader a good idea of what is happening.
* Name properties and variables clearly, concisely, and consistently.

# ARM Domain Names

Domain names in ARM must be globally unique. This applies regardless of whether the app is deployed in Cloudhub or on-premise. Names in ARM have a 42-character hard limit. They should be named for the service they provide. In addition, they should include the environment suffix, so the same application can be deployed into each environment. A single domain applies to a single Mule application instance, though the Mule app’s name and the domain name are NOT linked.

The domain is used for identifying and managing the app in Anypoint Runtime Manager. The domain may or may not be used to access the app. This is because apps running in CloudHub will not be available over the CloudHub public load balancer and can only be contacted through the dedicated load balancer, which will have a completely different domain/URL.

The CI/CD deployment will add the prefix (company-) and suffix (-environment) to the app name. Developers should NOT add these as part of the project or Git names.

* **Pattern:** *<company>-<app name>-<environment>*
* **Example:** *acme-sfdc-sys-dev*

Best Practice

* If you are only using a single Production environment, then omit the environment suffix for the production CloudHub domain. Example: *acme-sfdc-sys.*

# Custom HTTP Header Names

API developers should avoid creating custom headers as possible. In the rare case that a custom HTTP header is required, it should follow the standards below.

* Each word should be lower case and separated by hyphens.
* The header must have the company prefix.
* The pattern to follow: <company>-<optional category>-<field>
  + Category is the type of information in the header, such as security. For headers that are destined for specific SoRs, such as a SFDC OAuth token, the category can be the SoR.
  + Field is the specific name of the header.
  + Example: acme-security-oauth or acme-sfdc-OAuth (specifically sfdc token).
* Do not use the “x-” prefix as that is deprecated standard.
* Any header value where specific spacing and encoding must be maintained should be base64-encoded.

# Queue & Topic Names

Many apps use queueing software such as ActiveMQ, Anypoint MQ, and RabbitMQ. The queues and topics should follow the general standards below.

* Use the dot character '.' as a separator.
* Use only alphabet characters and dash character in name sections. Only use numbers if necessary.
* Be aware of system limitations.
* Anypoint MQ limits name to 127 characters and must be unique to business group/environment/region combination.
* Name reads left to right, moving from most general to most specific.
* The publisher owns the topic, so the topic name should be in that context.
* The consumer owns the queue, so the queue name should be in that context.
* Do not add regional or type portions in the name.

# API URLs

A Mule app’s API access is provided through a load balancer. This can be either the shared load balancer or dedicated VPC load balancers in Cloudhub or a company's load balancer solution when running on-premise. The app’s URL follows a specific pattern based into which region and environment the app is deployed. The pattern for the app’s URL is listed below. Note that <app name> refers to the app’s name in the pom.xml, which should be the same as the Mule project name and the Bitbucket repo name.

## CloudHub Shared Load Balancer

The [shared load balancer](https://docs.mulesoft.com/runtime-manager/dedicated-load-balancer-tutorial#shared-load-balancers) simply uses the application's ARM domain as the URL.

* **Pattern** 
  + Non-prod: *https:/<company>-<app name>-<env>.cloudhub.io*
  + Prod: *https://<company>-<app name>.cloudhub.io*
* **Example**
  + Dev: *https://acme-commercials-sfdc-sys-dev.cloudhub.io*
  + Prod: *https://acme-commercials-sfdc-sys.cloudhub.io*

## Cloudhub VPC Dedicated Load Balancer or Company Load Balancer

Each VPC in Cloudhub can have one or more [dedicated load balancers](https://docs.mulesoft.com/runtime-manager/dedicated-load-balancer-tutorial#dedicated-load-balancers). These follow a similar pattern as company load balancers as any domain can be given to them.

* **Pattern** 
  + Non-prod: *https://<loadbalancer>.<custom domain>/<app name>-<env>*
  + Prod: *https://<loadbalancer>.<custom domain>/<app name>*
* **Example**
  + Dev: *https://dev-us-e1.<custom domain>/commercials-sfdc-sys-dev*
  + Prod: *https://prod-us-e1.<custom domain>/commercials-sfdc-sys*

The API’s path would then be added onto the base domain and app name. See the full pattern below.

* **Pattern:** *https://<loadbalancer>.<custom domain>/<app name>/<path>*
* **Example:** *https://prod-us-e1.<custom domain>/commercials-sfdc-sys/v1/cases*

Best Practices

* Name load balancers according to region so they are easy to identify, consume, and manage in a multi-region environment.
* Many variations on this can be done based on the company's domain naming and management policies.