MuleSoft Integration Coding Standards

Department for Education

Version 0.1, 2022-01-27

Table of contents

[1. Document Control 2](#_Toc94185193)

[1.1. Version History 2](#_Toc94185194)

[1.2. Review History 2](#_Toc94185195)

[2. Introduction 3](#_Toc94185196)

[2.1. Audience 3](#_Toc94185197)

[3. MuleSoft Coding Standards 4](#_Toc94185198)

[3.1. General 4](#_Toc94185199)

[3.2. Naming Conventions 4](#_Toc94185200)

[3.3. Application Deployments 5](#_Toc94185201)

[3.4. Formatting 5](#_Toc94185202)

[3.5. Comments 5](#_Toc94185203)

[3.6. Global Declarations 5](#_Toc94185204)

[3.7. Dataweave Standards 5](#_Toc94185205)

[3.8. JSON Standards 5](#_Toc94185206)

[3.9. Logging Standards 6](#_Toc94185207)

[3.10. Git Ignore File 6](#_Toc94185208)

[3.11. MUnits 6](#_Toc94185209)

[3.12. Error Handling 6](#_Toc94185210)

[3.13. Project Structure 6](#_Toc94185211)

[4. RAML Standards 9](#_Toc94185212)

[4.1. RAML Files 9](#_Toc94185213)

[4.2. RAML Header 9](#_Toc94185214)

[4.3. Standards 9](#_Toc94185215)

[4.4. RAML Structure 10](#_Toc94185216)

[4.5. Error Messages 10](#_Toc94185217)

# Document Control

## Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 0.1 | 2022-01-27 | Steven Revill | Initial version |

## Review History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Reviewer** | **Comments (if any)** |
| 0.1 |  |  |  |

# Introduction

This document defines the coding standards required when developing MuleSoft applications. A large codebase is easier to understand and maintain when all the code is in a consistent style.

## Audience

This document’s audience primarily includes architects, consultants, and developers engaged in architecting, designing and developing, MuleSoft integration applications.

# MuleSoft Coding Standards

## General

* Unused flows/sub-flows must be removed before committing to version control
  + Do not comment flows, sub-flows, global-declarations, transformers or components without a dated note explaining why it has not been deleted.
* All flows, sub-flows, components, transformers, scopes (of any kind, including transactions) must be given brief, descriptive names.
  + When a transformer is setting variables, add the variable name to the description
* Do not use custom acronyms - use only industry-standard acronyms unless the custom acronyms are well-known and well-understood across the business
* Do not leave connectors, scopes etc. with the default naming. The aim is to allow maximum comprehension of flow logic while minimizing the amount of drill-down needed.
* Add secure properties to the mule-artifact.json secureProperties section, add the prefix secure:: for encrypted values
* Ensure timeouts are correctly set, in API-Led, the EAPI should expire last, then PAPI, then SAPI (eg SAPI should expire first)
* Ensure to send the correlation ID between APIs (set sendCorrelationId to "ALWAYS”)
* Externalise variables, do not hard code strings or numbers (no magic numbers)
* All passwords and credentials are encrypted

## Naming Conventions

* Project names are lowercase, hyphen-separated
* Identify the type of API in the project name (eg experience, process or system)
* Application XML file names are lowercase, hyphen-separated: impl-board-transactions-requests.xml
* Flow names are lower case and hyphen-separated
* Flow variables are lower camel-case: customerId
* Flow references should include the file, flow name and type of flow as the display name: financial-process-api:get-financial-trial-balance-data:subflow
* Properties files are lowercase and hyphen-separated
* Properties files must include the environment name and .properties extension: app-config-**dev**.properties

## Application Deployments

* Within the IDP infrastructure, the deployment name is all in lowercase hyphen-separated with the full name of the API e.g., board-experience-api
* For CloudHub deployments the deployment name is prefixed with the department’s acronym, hyphen separated with name of the API along with the type of API acronym, for example:
  + dfe-board-eapi
  + dfe-salesforce-papi
  + dfe-ceds-db-sapi

## Formatting

* Where possible indentation is 2 whitespaces (e.g. for dataweave, json)
* Separate unrelated parts of the flow vertically by one blank line to enhance readability

## Comments

* Use doc:name and doc:description attributes to clearly and concisely document functionality
* Add useful comments to dataweave logic

## Global Declarations

* All global declarations are to be kept in a dedicated ‘global’ configuration file.

## Dataweave Standards

* For large dataweave transformations, place them in a .dwl file.
* Dataweave files to be place in the dwl directory
* Dataweave files names are lower case hyphen-separated
* Variable names are declared in camel case, eg. objectCode

## JSON Standards

* Data within the platform should use a Common Data Model where possible
* Data communication within the platform should use JSON and the keys should be in camel case

## Logging Standards

* Do not log the payload
* Do not log Personally identifiable information (PII)
* Add unique logs at the start and end of each flow to track the journey of the requests
* Log messages should be concise in normal case with no special characters (e.g. >>>>)
* Log messages are for support and need to be meaningful to help debug production issues

## Git Ignore File

* The Anypoint Studio .gitignore file includes most of the correct files, add reports/

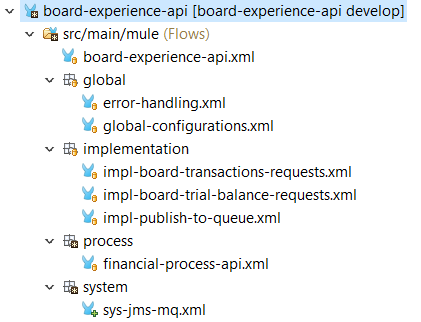
## MUnits

* All applications must contain MUnits with at least 80% test coverage (application and resource)
  + This is to be enforced with a parent-pom in the future
* MUnits should be self-contained with no consumption of external services

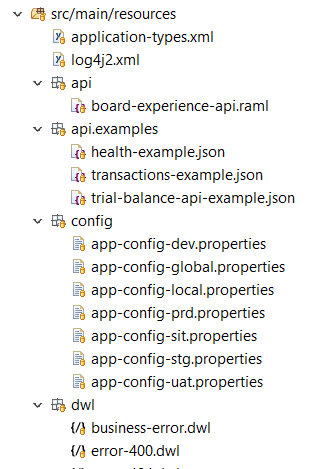
## Error Handling

* Use the standard error-handling.xml and dwl files
* Within the IDP, the practice is to send errors to the Error API via JMS
  + To be updated for CloudHub via Anypoint MQ

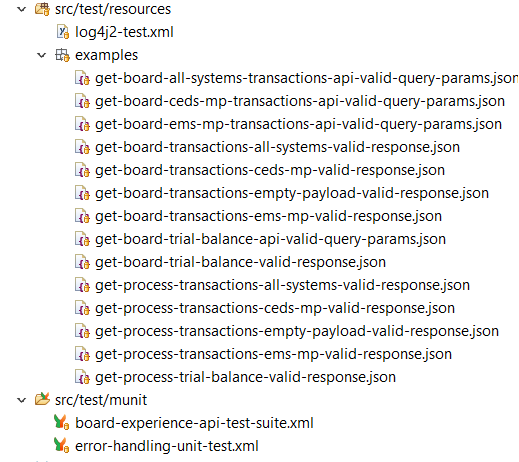
## Project Structure

The following folders are used for the main flows (all in lowercase)

* global
  + Contains common and global files such as the global-configuration.xml
* implementation
  + Contains the main implementation logic flows
* process
  + Contains any calls to process APIs
* system
  + Contains any calls to system APIs or downstream applications

**Resources:**

* api
  + examples
  + traits
  + dataTypes
* configs
* dwl

**MUnits/Tests:**

* test
  + resources
    - examples
  + munit

# RAML Standards

## RAML Files

RAML file names are in lowercase, alphanumeric characters using kebab-case (i.e. hyphen separated).

* For example:
  + account-example.raml
  + board-experience-api.raml

## RAML Header

**API Titles**

The title of the API should be a short, plain-text label for the API.

For example:

title: Account API

**Version**

Each API should have a corresponding alphanumeric version number.

For example:

version: 1.0.0

**Protocols**

The optional protocols node specifies the protocols that an API supports. If the protocols

node is not explicitly specified, the protocol in the baseUri node is used.

For example:

protocols: [HTTP, HTTPS]

## Standards

* File names and query parameters are in lowercase separated with hyphens, eg. last-updated-on
* Examples must be supplied
* Add data types if possible
* Indent by 2 whitespaces
* Resource names must follow RESTful best practice, in general, using plural nouns is preferred unless the resource is clearly a singular concept
* Separate words with hyphens

## RAML Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource** | **Path** | **Suffix** | **Example** |
| API Definition | / | .raml | board-experience-api.raml |
| Traits | /traits/ | .raml | cacheable.raml |
| Resource Types | /resourceTypes/ | .raml | collection.raml |
| Security Schemes | /securitySchemes/ | .raml | oauth2.raml |
| Data Types | /dataTypes/ | .raml | account.raml |
| .schema.json | account.schema.json |
| .xsd | account.xsd |
| Documentation | /documentation/ | .raml | account-doc.raml |
| Examples | /examples/ | .raml | account-example.raml |
| .json | account-example.json |
| .xml | account-example.xml |

## Error Messages

Every API should use a common error message schema for when an error needs to be returned in

a response payload:

The common error type schema contains the following attributes:

* code – The HTTP Status code of the response
* description – A sort message describing the error
* correlationId - The unique ID associated with the transaction in which this API is involved
* additionalDetails - A detailed message describing the problem in relation to the error type