Mule Development Standards

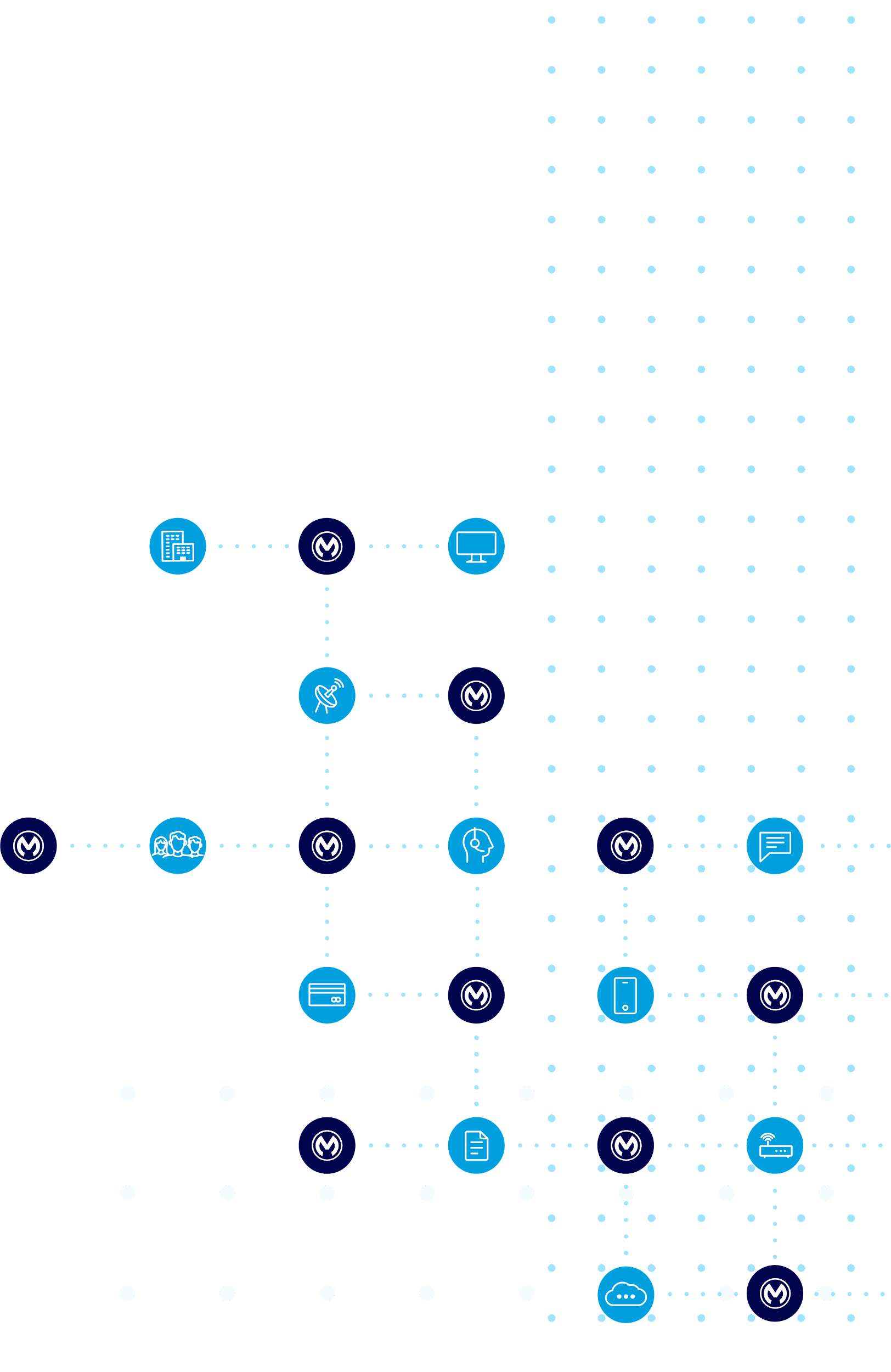


Table of contents

[1 Introduction 3](#_Toc90311855)

[1.1 Audience 3](#_Toc90311856)

[1.2 Overarching desired outcomes 3](#_Toc90311857)

[2 Development standards 4](#_Toc90311858)

[2.1 API specification guidelines 4](#_Toc90311859)

[2.1.1 Project naming conventions 4](#_Toc90311860)

[2.1.2 Other naming conventions 4](#_Toc90311861)

[2.2 API implementation guidelines 5](#_Toc90311862)

[2.2.1 Anypoint Studio project structure 5](#_Toc90311863)

[2.2.2 Project naming conventions 6](#_Toc90311864)

[2.2.3 Mule naming conventions 7](#_Toc90311865)

[2.2.3.1 Flow files 7](#_Toc90311866)

[2.2.3.2 Flows and sub-flow names 7](#_Toc90311867)

[2.2.3.3 Message processors and scopes 8](#_Toc90311868)

[2.2.3.4 Acceptable naming variations 10](#_Toc90311869)

[2.2.4 Global configuration elements 10](#_Toc90311870)

[2.2.4.1 Configuration filename 11](#_Toc90311871)

[2.2.4.2 Configuration element names 11](#_Toc90311872)

[2.2.5 Configuration and properties files 11](#_Toc90311873)

[2.2.6 CloudHub property overrides 12](#_Toc90311874)

[2.2.7 Hard-coded configurations 13](#_Toc90311875)

[2.2.8 APIs and auto-generated flows 13](#_Toc90311876)

[2.2.9 Imported resources 14](#_Toc90311877)

[2.2.10 Example project structure 14](#_Toc90311878)

[2.2.11 Anypoint Studio Project Structure 15](#_Toc90311879)

[Appendix A: Example DLB mapping rules 18](#_Toc90311880)

[Appendix B: Example of Display Naming Convention 19](#_Toc90311881)

[Appendix C: Case option descriptions 24](#_Toc90311882)

# Introduction

This document aims to capture recommended standards to be adopted when developing Mule applications. The adoption of such standards will result in a consistent codebase that is easier to maintain, of a higher quality and will enable collaboration across as well as within development teams. Without such standards, developers will rely on personal preferences resulting in difficulties and potential conflicts when attempting to contribute to code written by another developer.

This document does not consider general development best practices such as API versioning, packaging granularity, and keeping dependencies up to date. Such best practices are covered in a separate document.

These standards represent the opinions of MuleSoft and are reflected in the assets contained within Knowledge Hub. It is expected that this document will be duplicated and tweaked to meet the needs of a specific context. Ideally there would only be one version of this document within an organization. The degree to which these standards are enforced is outside the scope of this document.

## Audience

Mule developers and API designers.

## Overarching desired outcomes

* All lower kebab-case URLs
* Consistency
* Reduced maintenance effort
* Fewer bugs
* Ability to perform static code analysis

# Development standards

## API specification guidelines

The API taxonomy should follow generally accepted RESTful API design standards. This means that top-level resources defined in API specifications should represent business domain model resources wherever possible (e.g. employees, orders, contracts, etc.), and are thus defined as nouns. In Experience and Process APIs, it is allowable to use verbs as sub-resources when required to expose processes (e.g. trigger a batch job) to consumers.

### Project naming conventions

The names of API Specification projects will adopt the following pattern:

{context}-{name}[-{layer}|{identifier}]

Where:

* **{context}**: short code that is unique to the domain that this API belongs to, e.g. hr
* **{name}**: logical name of the application, e.g. employee. The {name} value can also be broken into multiple components, each separated with "-", e.g. employee-benefits
* **{layer}**: the API layer, one of eapi, papi or sapi; not relevant for interface fragment projects
* **{identifier}**: interface fragment project identifier, can be blank, typical values include common, traits and types, e.g. hr-employee-types; not relevant for interface specification projects.

### Other naming conventions

1. Project name components are always separated by hyphens (“-”)
2. Folder names should use camelCase, e.g. dataTypes
3. The *main* RAML file should have the same name as the project, e.g. hr-employee-benefits-eapi.raml
4. All other API specification files should use TitleCase, e.g. example JSON payloads, RAML named examples
5. Resource names should:
   1. be plural when they represent collections (e.g. /customers) otherwise singular (e.g. /cart);
   2. use kebab-case when there are multiple components.
6. Query and URI parameter names should use camelCase
7. Data type names should use TitleCase
8. Data attribute names should use camelCase
9. Use kebab-case namespace definitions when importing libraries and other resources via the uses declaration (e.g. product-types, global-responses).

## API implementation guidelines

### Anypoint Studio project structure

The Anypoint Studio project layout will adopt the following structure:

* /src/main/mule (lower-kebab-case)
  + {application-name}-main.xml
  + common
    - z-global-config.xml
      * (Non secured connector configuration)
    - z-global-secured-config.xml
      * (Secured connector configuration and the key name must be added in the mule-artifact.json for obfuscating the value)
    - z-common-error.xml
      * Place for API Kit errors, common error handling
    - z-common-config.xml
      * Place for common or utility flows for reusability (ie: code translation, calculation, etc)
    - Note that the “z-” prefix is there for template generated files and to distinguish from other common files.
  + implementation
    - [get|put|post|delete|patch]-{domain-context}.xml (e.g. post-account-benefits.xml)
* src/main/resources
  + dwl (lower-kebab-case)
    - p-<dataweave-payload-transformation>.dwl (for Payload)
    - v-<dataweave-variable-transformation>.dwl (for Variables)
    - c-<dataweave-common-utility>.dwl (for Common or Utility)
  + examples (TitleCase)
    - {EntityName}[Input|Output].[json|xml|csv]
  + properties (lower-kebab-case)
    - config[-secure][-{env}].[properties|yaml]

### Project naming conventions

The names of API Implementation projects will adopt the following pattern:

{domain}-{app}-{layer}

Where:

* **{domain}**: short code that is unique to the domain that this API belongs to, e.g. hr
* **{app}**: logical name of the application, e.g. employee. The {app} value can also be broken into multiple components, each separated with "-", e.g. employee-benefits
* **{layer}**: the API layer, one of eapi, papi or sapi.

Full example: hr-employee-benefits-papi.

### Mule naming conventions

#### Flow files

* Must be named using all lower-case names separated by hyphens (dashes). No underscores are to be used.
* The “main” flow file (either generated by APIKit or a main entry point) must exactly match the name of the interface specification (e.g. hr-employee-benefits-papi.xml for hr-employee-benefits-papi.raml). Note that there can be multiple “main” flow files if the project implements multiple interface specifications.
* The “implementation” flow must have the following pattern:
  + [get|post|put|delete|patch]-{context}.xml (e.g. get-employee-benefits.xml)

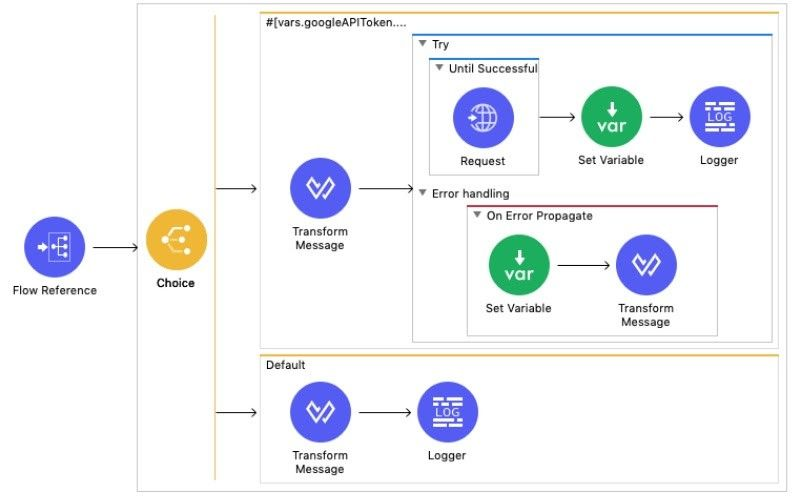
#### Flows and sub-flow names

* Must be named as {method}-yyyy, where {method} is the flow file name (excluding the .xml extension) and yyyy is the actual flow or sub-flow name, with name components separated by hyphens (e.g. get-employee-benefits). This causes flows to be grouped together in dropdown and autocomplete menus, making them easier to navigate.
* Flow names must be descriptive, concise, and cannot have embedded underscores or special characters (e.g. get-employee-benefits).
* The flow name in the main xml file must exactly match the root file name (e.g: hr-employee-benefits.raml) of the API specs from RAML (again, excluding the .raml extension) and have the term -main appended to it (e.g. hr-employee-benefits-main).
* Use the Notes tab to provide a brief description of the flow or sub-flow.

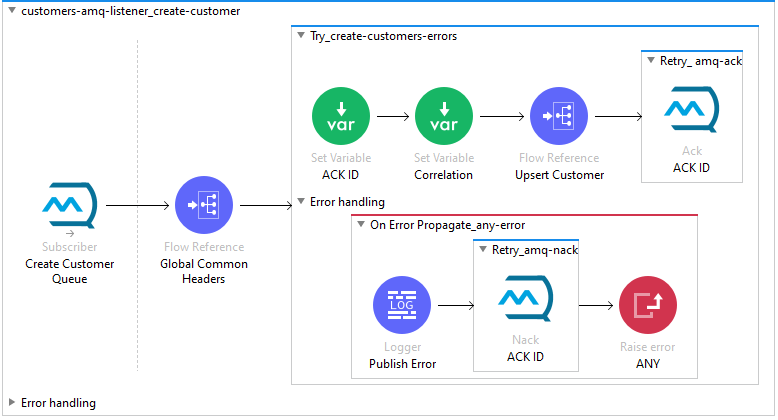
Note: The names of flows generated by APIKit scaffolding must not be modified; otherwise, the router will fail to find the flow implementing a particular endpoint.

#### Message processors and scopes

* Must be named concisely, indicating purpose, and cannot be left as the Anypoint Studio default names.
* Since the type of processor always remains visible, the display name should add to the overall description. For example, the display name for a Set Variable processor should indicate the actual variable being set (e.g. Response Status) , while for a Transform Message processor it should be the type of transform being done (e.g. Map Response Message).
* For scopes, keep the name of the scope but add an underscore and a brief description of the scope (e.g. For Each customer-entry). Name components follow the same convention as for flows, above.
* For a more complete example, the names in the following image are undesirable because they are all left at their Studio default values:



* Contrast that with the following, where clear names are provided on all processors and scopes - demonstrating the ease with which a casual reader can comprehend the overall purpose and implementation of the flow:



#### Acceptable naming variations

* For flow reference processors, the complete name of the flow (i.e. doc:name) may be used as the processor name. This makes it clear which sub-flow is being called.
* For set variable processors, the actual name of the variable can be used (e.g. httpStatus). Again, this makes it clear which variable is being set.
* Default scope names can be converted to the usual flow name format (e.g. for-each\_customer-record) for the sake of being consistent.

### Global configuration elements

All global configuration elements in applications must be defined in the flow src/main/mule/common/z-global-config.xml. This includes all configuration items such as HTTP listeners, ObjectStore references, connector configs, etc. There should be no configuration elements in any other flow files within the project. This means that, after generating a skeleton flow from an API, the HTTP connector and Router definitions must be moved out of the default location and into the global config file.

#### Configuration filename

For libraries, the global configuration file should be called global-{library}.xml, where {library} is the short name of the library. For example, if the org-common-core library requires some global resources, they would be defined in a file named global-common.xml.

#### Configuration element names

The display name for configuration elements should be in the formation {description}\_{type}, where {description} may consist of multiple names separated by hyphens. For example:

* main\_http-listener
* oms\_http-requestor
* env\_properties
* oauth\_object-store

One exception are *Global Property* elements, where the display name should reflect the name of the variable being set (e.g. mule.env). Configuration elements defined in libraries should be prefixed with the name of the library (e.g. common-selfsigned\_tls-context).

Note some configuration elements include both the name and doc:name properties in their declarations; these should generally be set to the same value.

### Configuration and properties files

All deployment-specific properties, such as hosts, ports, access keys, etc., are to be externalised in properties files located under the src/main/resources/properties folder. For Mule properties, use .yaml files; for Java properties only, use .properties files.

At a minimum, each project should contain the following files:

* config.yaml: Unsecured properties common to all environments
* config-{env}.yaml: Unsecured properties specific to individual environments
* config-secure-{env}.yaml: Secured properties specific to those environments.

The deployment environment identifier {env} represents the target environment in CloudHub (typically one of dev, test, or prod) that is used for application deployments. Note that it does not need to match the exact name of the CloudHub environment, e.g. test rather than tst. For local workstation deployments, use local; config-local files are typically not checked into source code. Note that there are generally no common secured properties.

If additional organization of properties files is required, perhaps because there are separate files for mapping or translating values in each environment, group them under sub-folders by environment identifier (e.g. src/main/resources/properties/dev). This makes them easy to reference using the same property as for the file names themselves.

User defined property keys should use camelCase and be stored in a hierarchical structure rather than a flat one, e.g. retry.maxCount and retry.interval rather than retryMaxCount and retryInterval.

### CloudHub property overrides

All the MuleSoft deployment models allow for overrides to any of the properties configured in the yaml files. RTF and Hybrid deployments are able to toggle the masking of the property values. CloudHub on the other hand utilizes a file in the project called mule-artifact.json. All CloudHub projects should add a “secureProperties” section to this file, adding property names into this array will keep the corresponding values masked when viewed in CloudHub. If the property value is contained in a secured properties file then the entry must include the “secure::” prefix. More information is available [here](https://docs.mulesoft.com/runtime-manager/secure-application-properties).

Example:

|  |  |
| --- | --- |
| {  "minMuleVersion": "4.3.0",  **"secureProperties": [**  **"mule.key",**  **"anypoint.platform.client\_id",**  **"anypoint.platform.client\_secret",  "secure::db.password”**  **]**  } |  |

### Hard-coded configurations

All global configuration elements must use parameterized values (i.e. properties) whenever those values may change from environment to environment or when they are intended to be customizable at deployment time. This includes but is not limited to:

* Protocol, host and port of HTTP listener configs.
* Auto-discovery API Id and base path.
* Deployment environment name and encryption key.
* Back-end connector host and port.
* Retry and timeout settings.
* Passwords, tokens and secrets.

### APIs and auto-generated flows

When an API is added to a project and scaffolded, Studio will generate a config file with default implementation flows; these flows may also be regenerated when an API specification dependency has been updated. The following changes should be made to ensure that the generated file has no flows or logic other than what the APIKit generator creates:

1. Replace the hardcoded example response transforms with flow-references
2. Improve the auto-generated API error-handling by editing or by including (via import) a reusable, global error handler.

The flow references will be to sub-flows defined in separate implementation files.

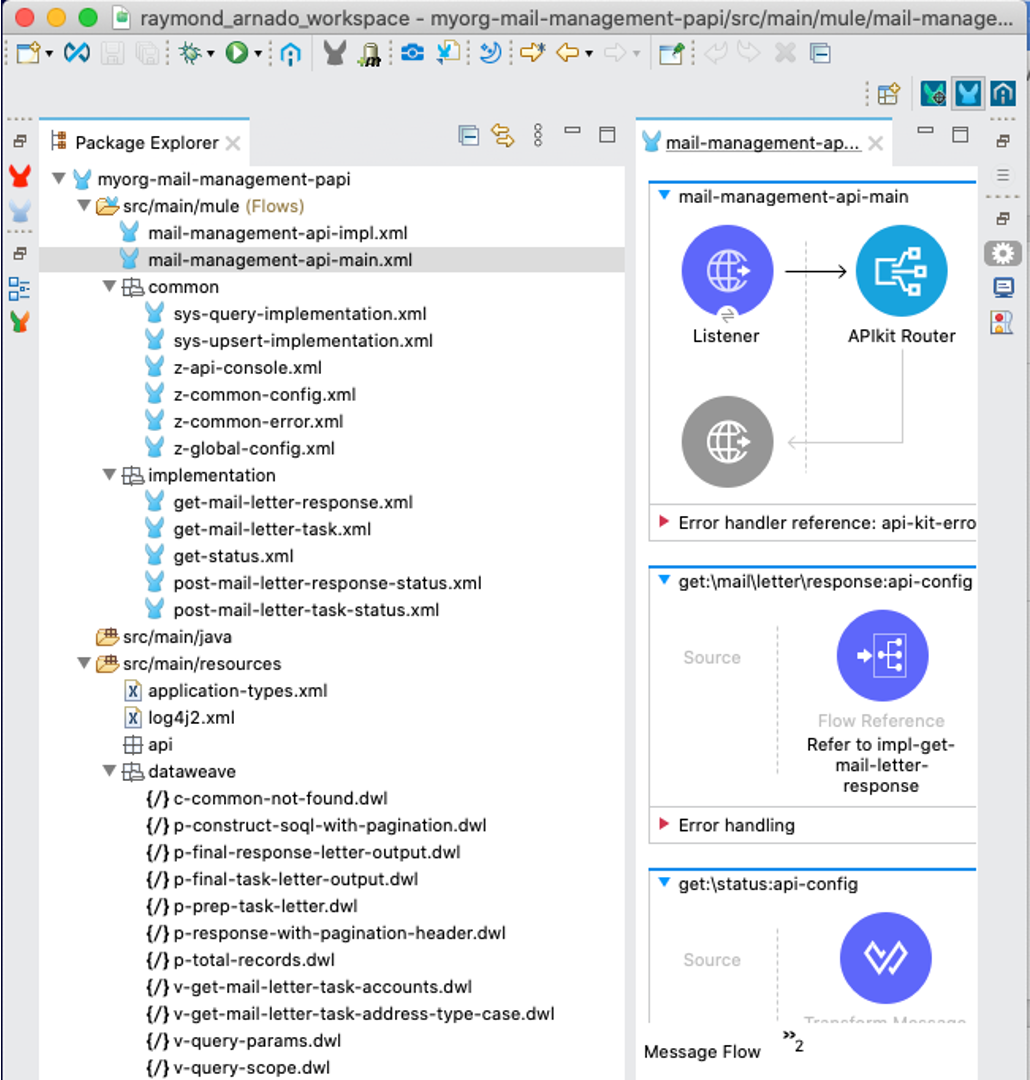
### Imported resources

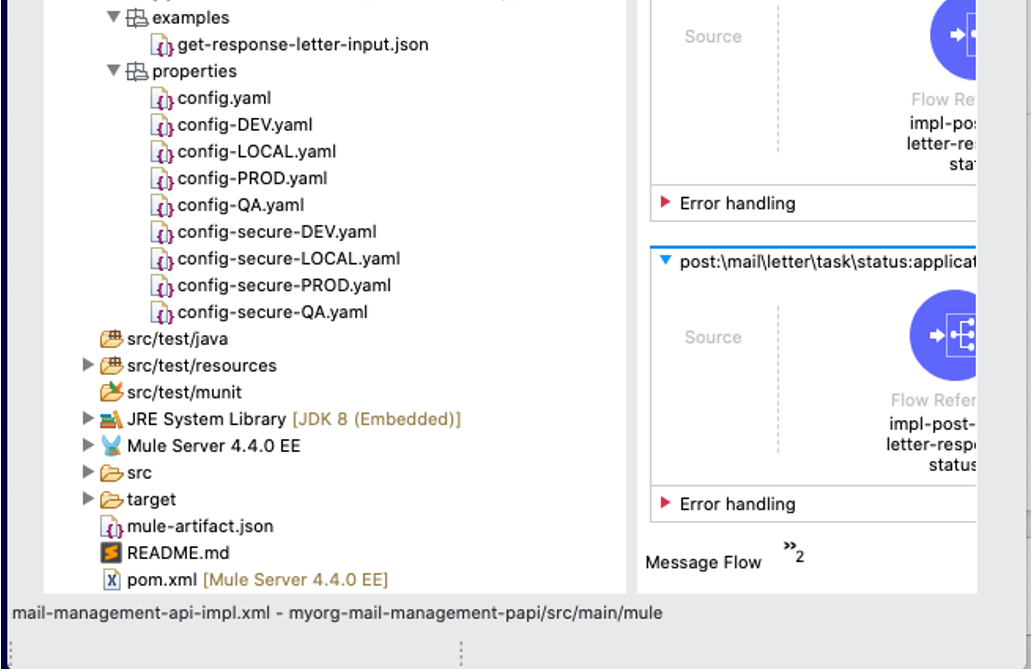
Any common files, which are intended to be included in multiple applications via the import mechanism, should be defined in a shared library. Mule configuration files should be located under the usual src/main/mule folder, while properties files, key stores, scripts, etc. should be located under src/main/resources. Although library projects are not intended to be runnable on their own, they may also contain MUnit and/or other testing resources under the src/test/munit and src/test/resources folders, respectively.

### Example project structure

The following diagram illustrates many of the conventions defined above:

### Anypoint Studio Project Structure





Deployed application name

The Dedicated Load Balancer mapping rules mandate the following deployed application name pattern:

«org»-{context}-{app}-{layer}-{version}-{env}

Where:

* **«org»:** prefix abbreviation for the organization. This is critical for CloudHub deployments as they need to be globally unique.
* **{context}**, **{app}**, **{layer}**: as per the project naming convention
* **{version}**: major version number prefixed with “v”, e.g. v1
* **{env}**: environment code, valid values: dev, tst, prd

Note that the length of the deployed application name must not be more than 42 characters.

Appendix A: Example DLB mapping rules

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Prod / Non-prod** | **Ext / Int** | **Hostname** | **Input Path** | **App** | **Output Path** |
| Non-prod | Ext | {env}.api.«org».com | /{context}/{app}/{version} | «org»-{context}-{app}-eapi-{version}-{env} | /api |
| Non-prod | Int | {env}.api-int.«org».com | /{context}/{app}-{layer}/{version} | «org»-{context}-{app}-{layer}-{version}-{env} | /api |
| Prod | Ext | api.«org».com | /{context}/{app}/{version} | «org»-{context}-{app}-eapi-{version}-prd | /api |
| Prod | Int | api-int.«org».com | /{context}/{app}-{layer}/{version} | «org»-{context}-{app}-{layer}-{version}-prd | /api |

Appendix B: Example of Display Naming Convention

|  |  |  |
| --- | --- | --- |
| Component | Description | Example |
| Dataweave | Display Name: DW <Purpose of the transformation>  o Example: DW Map Payload with Required Fields |  |
| For Each | Display Name: For Each <Name of the Collection>  o Example: For Each order-items |  |
| Logger | Display Name: LOG [DEBUG/ERROR] <Define the purpose of this log>  o Example:  o For common LOG:   * LOG DEBUG Payload Response from Salesforce   o For error LOG:   * LOG ERROR After Invoking SIMS US SOAP WS |  |
| Cache Scope | Display Name: CACHE <Object by CacheKey>  o Example: CACHE Account by accountId |  |
| Validator Component | Display Name: VALIDATE <Object>  o Example: VALIDATE Payload |
| Common Flow | Display Name: cf-<Name of the flow lower case>  o Example: cf-logger-init |  |
| Main Flow | Display Name: mf-<Name of the flow>  o Example: mf-impl-post-accounts |  |
| Sub Flow | Display Name: sf-<Name of the flow>  o Example: sf-validate-login |  |

Appendix C: Case option descriptions

* kebab-case
* snake\_case
* SNAKE\_CASE\_CAPS
* camelCase
* TitleCase (aka PascalCase)

|  |  |  |
| --- | --- | --- |
| **Case** | **Description** | **Example** |
| Kebab | All lowercase with words separated by “-” | hr-employee-name |
| Snake | All lowercase with words separated by “\_” | hr\_employee\_name |
| Snake (Caps) | All uppercase with words separated by “\_” | HR\_EMPLOYEE\_NAME |
| Camel | First word is all lower-case, the first letter of subsequent words is capitalized with no separator character between words | hrEmployeeName |
| Title / Pascal | The first letter of all words is capitalized, all other characters are lowercase with no separator character between words | HrEmployeeName |

About MuleSoft, a Salesforce company

MuleSoft, provider of the world’s #1 integration and API platform, makes it easy to connect [data](https://www.mulesoft.com/integration-solutions/dataweave-integration) from any system – no matter where it resides – to create connected experiences, faster. Thousands of organizations across industries rely on MuleSoft to realize speed, agility and innovation at scale. For more information, visit <https://www.mulesoft.com>.

*MuleSoft is a registered trademark of MuleSoft, LLC, a Salesforce company. All other marks are those of respective owners.*