[API Naming Convention 3](#_Toc517964063)

[Objective 3](#_Toc517964064)

[Global and Enterprise Impacts of Naming Convention Standards 3](#_Toc517964065)

[Cross Integration or Tech Platform Support 3](#_Toc517964066)

[Cross Deployment Models 3](#_Toc517964067)

[Vendor Agnostic API Naming Convention 4](#_Toc517964068)

[VA Background Requirements 4](#_Toc517964069)

[Naming Criteria 4](#_Toc517964070)

[Naming Convention Summary 5](#_Toc517964071)

[URL Examples 6](#_Toc517964072)

[Enviornment Naming Convention 6](#_Toc517964073)

[API Type 7](#_Toc517964074)

[Organization Name 7](#_Toc517964075)

[Major/Minor Versions 8](#_Toc517964076)

[API in Path 8](#_Toc517964077)

[Canonical Business Object Name 8](#_Toc517964078)

[RAML File Naming Convention 9](#_Toc517964079)

[Use Case 9](#_Toc517964080)

[Naming Criteria 9](#_Toc517964081)

[Naming Convention Summary 9](#_Toc517964082)

[General Naming Convention Guidelines 10](#_Toc517964083)

[MuleSoft specific API Naming Convention 11](#_Toc517964084)

[MuleSoft API Manager 11](#_Toc517964085)

[MuleSoft Application Naming Convention 11](#_Toc517964086)

[Use Case 11](#_Toc517964087)

[Naming Criteria 11](#_Toc517964088)

[MuleSoft Module Naming Convention 11](#_Toc517964089)

[Use Case 11](#_Toc517964090)

[Naming Criteria 11](#_Toc517964091)

[Naming Convention Summary 12](#_Toc517964092)

[Flow and Sub Flow Naming Convention 12](#_Toc517964093)

[Use Case 12](#_Toc517964094)

[Naming Criteria 12](#_Toc517964095)

[Naming Convention Summary 12](#_Toc517964096)

[Property Naming Convention 12](#_Toc517964097)

[Use Case 12](#_Toc517964098)

[External Property Placeholders 12](#_Toc517964099)

[Property File Naming Convention 12](#_Toc517964100)

[Naming Convention Summary 13](#_Toc517964101)

[Formatting 13](#_Toc517964102)

[Optional vs Required Elements for Property Files 14](#_Toc517964103)

[Extension of Blueprint 15](#_Toc517964104)

# API Naming Convention

## Objective

This blueprint defines an API naming convention for the Veteran’s Affairs enterprise, supporting all middleware and API management solutions and supporting all Veteran’s Affairs organization units and lines of businesses. Most importantly, define an API URL standard.

### Global and Enterprise Impacts of Naming Convention Standards

The naming conventions standards contained herein are applicable across any centralized or geographically distributed implementations of Integrations and API’s.

### Cross Integration or Tech Platform Support

The naming convention standards contained herein for API are applicable across integration platforms, as they are agnostic of the underlying technology stack. For example, if the API’s are developed on MuleSoft, 3Scale, Kong, CA Layer, or Spring, the API naming conventions should remain consistent and continue to be relevant no matter of the technology platform used to build and host these APIs. Specific vendor applicability is noted in separate sections.

### Cross Deployment Models

The naming convention standards contained herein for API are applicable across deployment and hosting models, whether the API is deployed to a Cloud (IaaS, PaaS, or SaaS) or On Premise, as the naming conventions either accounts for the deployment model or is agnostic of the underlying hosting and cloud model. The conventions account for deployment in the VA Enterprise Cloud (VAEC) deployment model to ensure uniqueness across the VA.

The VA naming convention standard requires the API name to be unique no matter the region, locale, or infrastructure environment due to being deployed in a domain that supports the cloud hosted environment across clouds, as well as On Premise.

Note, whether an API is Internal facing, External facing, or both for Consumer use, is addressed and controlled via API policies, and not addressed via API naming conventions. For more information please see **Playbook: API Policies framework.**

Note, the scope of this API Naming Convention document is to focus on the major API naming convention elements, including and up to the application name. This blueprint document does not address granular SOAP and REST operation naming conventions. **Please see Playbook API Operations for guidelines on such.**

## Vendor Agnostic API Naming Convention

### VA Background Requirements

Table : VA API Naming Conventions Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Framework Impacts |
| 1 | Veteran’s Affairs has a preference to host API’s, applications, and IT solutions in the Veterans Affairs Enterprise Cloud (VAEC). Some API’s may still be hosted On Premise. | The VA Naming Convention accounts for Cloud or On-Premise solutions. |
| 2 | VA does a majority of their business in USA. | The Naming Convention will not account for a region or locale in the naming convention because the majority of VA is USA based.  Even for other regions outside of the USA, the API Naming conventions does factor in the different Organization Name into the naming standard and therefore could allow the API’s in other countries to be differentiated by the Organization Name  Should VA business model change the future, where there is more globalization and geographic differences outside of the USA, the Naming Conventions shall be adjusted at that time. |
| 5 | VA has many Use Cases that involve Salesforce.com and enterprise systems (although, these are not the only Use Cases, and this framework is a broader, enterprise level framework) | The Naming Conventions shall account for the Canonical representation of business objects, such as “Veteran, “Address”, and others. |

### Naming Criteria

Following the industry standards on web resource definition, any API URL always consists of the protocol, host, port, base path, RESTful operation, and, finally, parameters to the API t.

With the API URL naming convention, the following required and optional criteria should be met:

1. Designate the environment hosting the API (dev, qa, production, uat, etc.).
2. Define whether the API is Experience, Process, or System API. **(See Playbook section on “API Layers”).**
   1. Experience API is targeted to a specific UI platform such as mobile app, desktop portal, or tablet.
   2. Process API is used to orchestrate and provide business logic across two or more system APIs.
   3. System API is used to integrate with a back office or external system or platform (e.g., a database or an ERP system). Typically, follows a Create/Read/Update/Delete model with system interaction.
3. Distinguish the Veteran’s Affairs enterprise Organization Name who owns the API. For project teams where, multiple teams are developing and contributing to the API, the Organization Name who has ownership of the API should be noted in the naming convention. If the API is not owned nor is functionally tied to an Organization, then it should not be part of the URL. Instead, the word “Enterprise” can be substituted in for the Organization Name.
4. Define the Veteran’s Affairs enterprise application serving this API. I.E. what system is the “Provider” of the API?
5. Identify the key Canonical Business Object that the API represents for the enterprise.
6. Define the base API path, /api, needed to differentiate all APIs from other HTTP resources throughout the enterprise at Veteran’s Affairs.
7. Provide ability by the API consumer to easily determine what version of the API is being consumed by incorporating version numbering into the API URL itself.
8. Define the RESTful operation and parameters (**See Playbook section: REST Operation Naming Convention section).**

Below is a table documenting which criteria are required and which are optional in the VA API naming convention standard:

### Naming Convention Summary

Table : API Naming Convention Summary

| Category | Req/Opt | Example |
| --- | --- | --- |
| Environment | Required | dev  qa  uat |
| API Type  (Experience, Process, System) | Required | exp for Experience  proc for Process  sys for System |
| Organization Name | Required | Vet360  DAS  iHub  VistA  Enterprise (Not associated with an Organization)  C4E (owned by the API Governance team) |
| API Path | Required | /api in the path to differentiate the APIs |
| API Version | Required | v1, v2, v3 |
| Application | Required | veteranportal (experience API)  address (process API)  addressdetails (system API) |
| Canonical  (Enterprise Business Object) | Optional | customer  account  address |

### URL Examples

Below are visual examples of the API URL elements associated with VA standard. The convention defined herein shall be used for API’s developed outside of a specific API or vendor platform.

VA API URL:

Figure 1: VA API Example



Note, the naming standard shall enable unique API names across all of VA enterprise.

### Enviornment Naming Convention

The naming convention for the API’s deployed in VA should incorporate environment names always, (The assumption is that the API shall be deployed across multiple environments).

As an example, the environment should be the prefix for each of the deployed API’s except for Production, which doesn’t have the environment prefix.

Normally, the continental region is also included in the prefix of an API via industry best practice, but since VA pre-dominantly operates in in the US, the current naming convention doesn’t have any global region requirements (The assumption is all APIs are US based).

Table 3: VA Environment Naming Convention Summary

|  |  |  |
| --- | --- | --- |
| **Environment** | **API name** | **Application URL (App URL)** |
| DEV | dev-exp-vet360 | dev-exp-vet360.va.gov:8081 |
| QA | qa-exp-vet360 | qa- exp-vet360.va.gov:8081 |
| PROD | exp-vet360 | exp-vet360.va.gov:8081 |

### API Type

Table 4: API Type applied to API Naming Convention

| Category | Definition | Convention | Example |
| --- | --- | --- | --- |
| Experience Application | Experience application, targeted to the mobile, desktop, or tablet experience | **exp-<Organization Name>** | exp-vet360 |
| Process Application | Process application, designed for business logic orchestration and facilitation across multiple system API’s | **proc-<Organization Name>** | proc-vet360 |
| System Application | System application, targeted for interacting with one or multiple backend systems | **sys-<Organization Name>** | sys-vet360 |

### Organization Name

The main VA Organization Name is always part of the API URL path. For example, the Organization Name, “Enterprise Services” or “Vet360”, is added as part of the API base path.

The framework is flexible in that it allows for any VA Organization Names to be added or subtracted from the API path naming convention, if the Organization Name is unique\*.

If an API is not associated, owned, or managed by a single VA Organization, then it must be reflected in the naming convention as “Enterprise”. This means multiple teams own the API and ownership cannot be delineated to a single VA Organization. If the C4E or the Integration governance team owns the API, then it’s reflected as “C4E” naming convention in place of the Organization Name.

In this context, the Organizational Names examples for Veteran’s Affairs and are provided below:

Examples of Line of Business Team Name for Veteran’s Affairs, include:

* + Enterprise Services (API URL path element: “/EnterpriseServices”)
  + Vet360 (API URL path element: “/Vet360”)
  + DAS (API URL path element: “/DAS”)
  + C4E (API URL path element: “/C4E”)
  + No Team or Shared (API URL path element: “/Enterprise”)

(\* Note, the Organization Name should be unique across teams, but the naming conventions framework could still allow for duplicate Organization Names, if API name uniqueness is applied by using another parameter from the URL path, such as the URL elements: “Application”, “Canonical”, or something else in the URL path.

### Major/Minor Versions

We recommend having only the major versions in the API Base URL naming convention (e.g., “v1”), as opposed to showing major/minor versions (e.g., “v1.0” or even “v1.0.0”). The consumer of the API should be able to easily determine by the API URL what major version of the API is being consumed. The consumer of the API mostly won’t be concerned with the minor release version from the actual API URL. This granular version information should be provided to the API consumer in the detailed release notes, provided to the consumer or Veteran’s Affairs partner via the API Portal or via written release notes.

The API Portal or API Manager should capture additional metadata about each public API, such as major/minor API version, all system integrations for that API, owner of the API, examples of consuming the API within the internal Veteran’s Affairs platforms, and other detailed information about the API.

On the contrary, the API Base URL should provide clear and targeted information about that API, its owner, line of business, API type, major released version, and the application.

Note, this “Naming Conventions Framework” content does not replace the need for another framework for “Version Control Framework”. **Please see Playbook section on: Version Control**

### API in Path

“/api” will be part of the API URL base path as illustrated above as a required element in the API Base URL path. The reason for having /api always being part of the path is two-fold:

* Having /api in the path helps configure load balancing solutions (i.e., for L7 path-based balancing of load across multiple instances), as well as to configure any type of internal gateway and firewall policies for all Veteran’s Affairs Enterprise APIs.
* Having the word /api in the path instructs the consuming applications that this URL is an API URL, which follows certain API configuration and naming standards.

## 

As described in other sections, it is recommended to include a version number in the base path of an API to provide flexibility when it does come time to upgrade consumers to a latest version.  With a versioned API, you will have the option to deploy a latest version of the API while continuing to support consumers of the outdated version because you don’t have to immediately deprecate the outdated version. You can have two concurrent versions of the

### Canonical Business Object Name

If an API is associated with a Canonical Business Object, that should be captured in the API Naming Convention URL path. Some API’s may not be associated with a Canonical Business Object, so this criterion is “optional”.

The following decision tree can be applied to decide when it is required vs optional:

1. Is the API associated with a Canonical Business Object? **Yes/No**
   1. **Yes** – add the Canonical Business Object name to the API naming convention
   2. **No** – don’t add the Canonical Business object name into the API naming convention

Final API URL should look like this:

https://dev-exp-vet360.va.gov:8081/api/v1/veteranportal/address (with optional Canonical Business Object element named “address”)

https://dev-exp-vet360.va.gov:8081/api/v1/veteranportal (without optional Canonical element)

### RAML File Naming Convention

#### Use Case

Provide a consistent and standard naming convention for the RAML files.

#### Naming Criteria

All RAML files should follow standard naming convention of a standard file prefix “RAML” with the rest of the name tied to the name of the API or application.

#### Naming Convention Summary

The name of each RAML file should be:

**RAML**-<Organization Name>-<Application Name>

For example:

**raml**-Vet360-VeteranPortal

### General Naming Convention Guidelines

Some general guidelines and rules when choosing naming convention standards, which were applied throughout this blueprint, are listed below:

* Names should be descriptive and encode the functionality of the IT asset being identified.
* Shorter names are preferred over longer names to ensure simplicity, less clutter, and avoid byte constraints across systems.
* Extremely short names such as 1-character names are discouraged.
* Names should be descriptive so that they can be interpreted and understood quickly, preferably without having to review reference guides often.
* Any IT asset named should follow CamelCase practices, which is the practice of writing compound words or phrases so that each next word or abbreviation begins with a capital letter. CamelCase starts with a capital.
* Periodically spot check auto generated names from the commercial toolsets to ensure there aren’t any names that could be difficult to manage. Take periodic sample sets to spot check so you’re not spending lots of time.
* Since the name of an API deployed within the VA domain, it could span VA Clouds or On-Premise environments, but still need to be unique across the entire VA domain. Since the API will represent a domain name (DNS entry), some care must be taken to ensure these names are unique and orderly across the VA enterprise.

## MuleSoft specific API Naming Convention

The MuleSoft Naming Conventions framework includes the following MuleSoft components to apply naming standards to:

* MuleSoft Application (API) Naming Standards
* MuleSoft Property File Naming Standards
* MuleSoft Module Naming Standards
* MuleSoft Flow and Sub Flow Naming Standards

### MuleSoft API Manager

Within the MuleSoft API Manager, there is API URL field on the screen that needs to be populated with the API Base URL, following the naming convention defined in this document. Examples of sample APIs that could be defined in the API Manager are provided below:

Figure : Veteran’s Affairs - API URL setting in MuleSoft API Manager

INSERT SCREENSHOT

### MuleSoft Application Naming Convention

#### Use Case

Standardize MuleSoft application names across the Veteran’s Affairs enterprise. The name of the application should be meaningful and should provide as much details about the purpose and use of the application as possible.

#### Naming Criteria

Since all MuleSoft applications are separated into Experience, Process and System applications, the naming convention should be aligned to these types of applications.

INSERT SCREENSHOT

### MuleSoft Module Naming Convention

#### Use Case

Standardize Mule module name across the Veteran’s Affairs enterprise. The name of the module should be indicating whether this is a MuleSoft component, library, or connector, as well as providing a clear indication on the purpose and name of that module.

#### Naming Criteria

All MuleSoft modules can be separated into libraries (i.e., packaged as JAR) or connectors.

Shared libraries (i.e. compiled binaries) should be packaged as JARs and deployed into a shared module registry, such as Artifactory (3rd-party product to MuleSoft). The JAR can then be easily integrated (i.e. consumed) into any Mule process flow or sub-flow via Maven or Gradle reference. **See Playbook: Shared Components** for more information on consuming shared modules.

#### Naming Convention Summary

Table 5: Mule Module Naming Convention Summary

| Category | Definition | Convention | Example |
| --- | --- | --- | --- |
| MuleSoft Library | MuleSoft library of reusable flows, sub-flows or resources | **lib-common-<MuleSoft library name>** | lib-common-errorHandling |
| Connector | MuleSoft custom connector | **connector-<Line of Business>-<connector name>** | connector-vet360-VeteranContactDatabase |

### Flow and Sub Flow Naming Convention

#### Use Case

Standardize MuleSoft flow and sub-flow naming convention, to allow for consistent naming of them across Veteran’s Affairs enterprise MuleSoft projects.

#### Naming Criteria

All MuleSoft flows and sub-flows should be properly prefixed with “flow- “prefix.

#### Naming Convention Summary

All flows should have “flow- “prefix.

Example: flow-<flow name>

INSERT SCREENSHOT

### Property Naming Convention

#### Use Case

Can create a property file for each target Mulesoft environment and deploy the same artifact across multiple environments (e.g., dev, qa, production), without modifying the keys of the property parameters. Standardize on property file and property key naming conventions.

#### External Property Placeholders

The proposed solution externalizes environment specific information to property files located on

a specific server path (e.g. /opt/mule/mule-ext-configs). These property files can/will be picked up by each Mule application using a global configuration named: <context:property-placeholder>

In practice, property-placeholders work exactly the same as the default mule-app.properties file

included on every Mule application and properties defined on property files (as key=value pairs)

can be read in the same way: ${myproperty}

#### Property File Naming Convention

Given these files can contain pretty much anything, it is recommended to define a naming convention that would provide enough granularity, so that each application could reference only what is needed

without adding too much complexity.

The current approach takes the following naming conventions:

<Organization Name>\*-<source>-<source-type>\*-<credentials>\*-<component>\*.properties

Below is a table that explains each property file name convention part, with examples:

#### Naming Convention Summary

Table 6: Mule Property Naming Convention Summary

| Category | Definition | Req/Opt | Example |
| --- | --- | --- | --- |
| Organization Name | Represents VA Organization Name | **Optional** | * **Vet360-**mongodb.propeties |
| Source | Represents the source system for which you are externalizing the properties (e.g. mongodb) | **Required** | * **mongodb** .properties |
| Source-Type | Provides ability to specify the specific type of source (e.g. amqp with configuration of type A, B or C). | **Optional** | * amqp-**A**-producer.properties * amqp-**B**-producer.properties |
| Credentials | For some systems there may be a completely  separate set of credentials for each component being build. In this situation, it is recommended to define multiple credential-specific property files instead of having a full copy of all the  configuration for every component | **Optional** | * amqp-**credentials**-enricher.properties * amqp-**credentials**-selector.   properties   * amqp-**credentials**-listener.properties |
| Component | Occasionally, there is a need to have a set of properties per component, in which case it is recommended to define it at the end of the name. | **Optional** | * amqp-credentials-**enricher**.properties * amqp-credentials-**selector**.properties |

Final Property File naming convention should look like this:

* <source>-<source-type>-<credentials>-<component>.properties

Notice that only <source> is required, with the rest being optional.

#### Formatting

Each property file should start by defining which system it’s meant for surrounded by

triple hashtags (### <system> ###).

Next, each set of related properties should have a sub-header describing that group

starting with a hashtag (# <description>)

Example:

### Vet360 amq ###

# Connection

aaa.csf.connectionNameList=svm2038pdv.dev.va.gov(1414),svm2038pdv.dev.va.gov(1415),svm2038pdv.dev.va.gov(1416)

cat.csf.clientReconnectTimeout=3600

cat.csf.queueManager=SVM2038PDV

cat.csf.transportType=1

cat.csf.targetClientMatching=true

# Vet360 SSL Config

cat.wmq.sSLCipherSuite=TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256

cat.wmq.ssl.keyStorePassword=password

#### Optional vs Required Elements for Property Files

The following decision tree can be applied to decide when there are required vs optional elements for Property Files:

1. Is the property file owned and maintained by a single Organization? **Yes/No**
   1. **Yes** – add the Organization Name to the property file naming convention
   2. **No** – don’t add the Organization Name into the property file naming convention
2. Does the source of the property file have multiple source types associated with that source? For example, mongodb is the source but it has multiple source types, based on specific purpose or function. **Yes/No**
   1. **Yes** – add the Source Type to the property file naming convention
   2. **No** – don’t add the Source Type to the property file naming convention
3. Is there a separate set of credentials for this source that requires separate property file to maintain? For example, one set of credentials for service accounts, and the other for users with elevated access rights. **Yes/No**
   1. **Yes** – add the Credentials to the property file naming convention
   2. **No** – don’t add the Credentials to the property file naming convention
4. Are there multiple unique components in Mulesoft for this source that requires separate property file to maintain? For example, an Enricher component requiring this source with different properties vs a Selector component, requiring separate set of properties? **Yes/No**
   1. **Yes** – add the Component to the property file naming convention
   2. **No** – don’t add the Component to the property file naming convention

## Extension of Naming Convention Blueprint

Naming conventions can include additional API or integration assets that need a name associated to it that are beyond just the API and other components described herein. For example, 3rd-party artifacts and assets, field names, testing tool naming conventions, database naming conventions, all of which are outside of API or the API platform, but still should have a standardized naming convention. This blueprint can be extended and content added to this document to include these other IT asset names associated with integration initiatives, including but not limited to:

* Specific components for additional platforms beyond MuleSoft (EX: API’s built in other platforms) flows, modules, and applications/packages.
* Naming fields in an API Contract or Middleware Canonical
* Naming supporting databases and structures to support the API Platform
* Naming API Platform environments
* Naming support assets (certificates, cache stores, queues, etc.)
* Naming Test Cases in Testing Tools or Unit Test Cases in platform tools.
* Adding additional API contracts, such as OpenAPI Swagger and associated naming conventions.