# API Naming Convention

## Objective

This blueprint defines an API naming convention for the VA Enterprise, supporting all middleware and API management solutions and supporting all VA 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 APIs.

### Cross Integration or Technical Platform Support

The naming convention standards contained herein for APIs are applicable across integration platforms, as they are agnostic of the underlying technology stack. The intention is to provide a vendor neutral convention for MuleSoft, 3Scale or any other API development and hosting platform. Where product-specific guidance is required, these are addressed in independent sections.

### Cross Deployment Models

The naming convention standards contained herein for APIs are applicable across deployment and hosting models, whether an API is deployed to a Cloud (e.g. IaaS, PaaS, or SaaS) or On Premise, as the naming conventions either account 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 unique names across the entire VA.

The VA naming convention standard requires the API name to be unique no matter the region, locale, or infrastructure environment to avoid possible future collisions as APIs may be rehosted to different environments.

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.[[TBD]]**

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.****[[TBD]]**

### Discoverability

API names should be syntactically valid DNS names (as per [RFC 1035](http://www.ietf.org/rfc/rfc1035.txt)) which can be resolved to one or more network addresses. If an API is composed of several services or APIs, they should be named in a way to help discoverability. One way to do this is for the API Names to share a common prefix. For example, the services build.VAappname.com and buildresults.VAappnameapis.com are both services that are part of the VAappname Build API.

## Vendor Agnostic API Naming Convention

### To provide consistent developer experience across many APIs and over an extended period of time, all names used by an API should be:

### simple

### intuitive

### consistent

### VA Background Requirements

Table 1: VA API Naming Conventions Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Framework Impacts |
| 1 | VA has a preference to host APIs, applications, and IT solutions in the Veterans Affairs Enterprise Cloud (VAEC). Some APIs may still be hosted On Premise. | The VA Naming Convention accounts for Cloud or On-Premise solutions. |
| 2 | VA does most 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 APIs in other countries to be differentiated by the Organization Name  Should VA’s 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. |
| 3 | 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.

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

1. Designate the environment hosting the API (e.g. dev, qa, production, uat).

Define whether the API is Experience, Process, or System API See: [[03.00 ASG\_API Playbook\_API Layer\_Section]!03.00 ASG\_API Playbook\_API Layer\_Sectio

* 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), and typically follows a Create/Read/Update/Delete model with system interaction.

1. Distinguish the VA 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 for the Organization Name.
2. Define the VA Enterprise application serving this API (i.e. what system is the “Provider” of the API?).
3. Identify the key Canonical Business Object that the API represents for the enterprise.
4. Define the base API path, /api, needed to differentiate all APIs from other HTTP resources throughout the VA Enterprise.
5. 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.
6. Define the RESTful operation and parameters (**See Playbook section: REST Operation Naming Convention section).[[TBD]]**

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

### Naming Convention Summary

Table 2: 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 APIs deployed in VA should incorporate environment names always. An API will always be deployed across multiple environments.

As an example, the environment should be the prefix for each of the deployed APIs 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 predominantly 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 the VA and are provided below:

Examples of Line of Business Team Name for the VA, 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

It is recommended to have 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 VA 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 VA 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 VA 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 APIs 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

The full 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

### Property File Naming Convention

#### Use Case

Can create a property file for each target API 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 API application using a global configuration named: <context:property-placeholder>

In practice, property-placeholders work 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 |

#### 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>)

In the example below, each property row is a specifically named connection to the back-end system. In this example, the back-end system is named “cat.csf”

Example:

### Vet360 amq ###

# Connection to System Properties

cat.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

#### Configuration Parameters

Configuration parameters are defined inside environment property files or added to the runtime configuration of the application. In instances where these parameters are defined inside property files, these parameters are commonly entered as key-value pairs.

For example, for VA vet360 API, there is a need to store a configuration parameter for the relational database. The database has its host, port, username, password, and database instance. All of these variables are considered configuration parameters that are managed either in one of the environment property files or as runtime arguments when launching an application (e.g., Linux runtime arguments, Java JVM arguments, etc).

**Important**: Since configuration parameters are managed in Environment Property files, which already are named by specific environment (“dev.configuration”), there is no need to add an environment name into the name of the configuration parameter. In fact, it is the opposite, no environment-specific indicators, keywords, or flags should be part of the name of any configuration parameter. Configuration parameters are reused across environments and, as such, are not tied to a specific environment.

With that said, once a configuration parameter is placed in an environment property file tied to a specific environment, the value of that parameter becomes specific to that environment.

For example, **mule.env**=dev inside the dev.configuration property file. The same **mule.env**=qa is now inside the qa.configuration file.

The naming conventions for the configuration parameters should reflect the following:

|  |  |
| --- | --- |
| API Name | Example |
| Product Name | VA\_AppName\_[*owner-initials]* Calendar API |
| Service Name | calendar. VA\_AppName\_[*owner-initals]*apis.com |
| Package Name | VA\_AppName\_[*owner-initials]*/.calendar.v3 |
| Interface Name | VA\_AppName\_[*owner-initials]*/.calendar.v3.CalendarService |
| Source Directory | //VA\_AppName\_[*owner-initials]*/calendar/v3 |
| API Name | calendar |

* Application name
  + This will be a short name for the application based on the APP code that is associated with this property. For example, “vet360” for the Vet360 application.
* Platform type:
  + “db” for database
  + “jms” for JMS queue
  + “mule” for Mulesoft platform parameters
  + “web” for webserver
  + etc
* Platform name (if applicable):
  + “sqlserver” for SQL Server database
  + “mongo” for MongoDB
  + “oracle” for Oracle database
  + “apache” for Apache web server
  + “tomcat” for Tomcat instance
  + etc
* Parameter type:
  + “env” for environment instance
  + “host” for host name
  + “port” for port number
  + “username” for username
  + “password” for password
  + “instance” for the database instance name
  + etc

An example of a full name of a configuration parameter:

* vetapi.mule.host
* vetapi.mule.audituser
* vet360.db.oracle.host
* vet360.db.oracle.port
* vet360.web.iis.port

#### Optional vs Required Configuration Parameters (i.e. 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

#### Configuration Properties Naming Convention Guidelines for Properties Files

Some general guidelines and rules when choosing naming convention for configuration parameters:

* Shorter names
* Never have environment specific keywords in the name of the parameter (e.g., “dev” or “prod”)
* Names should be descriptive to clearly indicate which platform or system that parameter is associated with, the name of the platform, and the type of the parameter
* All parameters across the VA organization should be consistently named based on a single canonical model. Example model for the parameter names can contain these globally defined parameter types (this is just an example of what the model can contain):
  + host
  + port
  + instance
  + env
  + username
  + password
  + path

### 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.
* An API name should not conflict with any well-established concepts in programming languages and their runtime libraries (for example, File). In the rare case where an interface name would conflict with another name within the API, a suffix (for example Api or Service) should be used to disambiguate.
* Field names should avoid prepositions (e.g. "for", "during", "at"), for example:
* If a date must be represented as a string, it should be in the ISO 8601 date format YYYY-MM-DD, e.g. 2014-07-30.
* If a time of day must be represented as a string, it should be in the ISO 8601 24-hour time format HH:MM:SS[.FFF], e.g. 14:55:01.672.

## 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 1: VA - API URL setting in MuleSoft API Manager

INSERT SCREENSHOT

### MuleSoft Application Naming Convention

#### Use Case

Standardize MuleSoft application names across the VA 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 VA 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 VA 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

### MuleSoft specific API Naming Convention

#### Use Case

Can create a configuration parameter 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 configuration parameter naming convention.

#### Global Properties

Mule 4 has ability to define global properties. As a developer, you can use the <global-property> element to set a placeholder value from within your Mule configuration, such as from within another Mule configuration file. Example of a definition of a global property is shown below:

Figure 2: Global Properties Example



The properties are managed in mule-app.properties file in Mule 3 and Mule 4, but with the new 4.x release Mulesoft has also provide an ability to manage global properties inside a YAML file. The format of a YAML file allows to structure all properties in a nested tree. For example, instead of the above “smtp.host”, we can now store the same property inside a YAML file like this:

smtp:

host: “{host name}”,

subject: “{subject name”

With that said, the naming conventions for the global configuration parameters should still follow the naming conventions defined in the previous, Technology-Agnostic, section.

Mulesoft has many reserved global configuration parameters, which all start with “mule.” in the name of the parameter. For example, “mule.env” is reserved to the name of the Mule environment. In addition, there are other reserved Mule properties that are tied to the application deployed. For example, “app.name” is the name of the application running within the Mule runtime.

In a Mule process flow, a developer can reference these parameters like this: ${mule.env}, by adding a dollar sign “$” and wrapping the parameter name with the curly braces. In Mule Expression Language (MEL) used inside an expression such as inside a DataWeave Transformations control, accessing properties is done with a reserved “#p[]”, where you would place the name of the property inside the square brackets. For example, “#p[‘vet360.db.oracle.host’]”.

### 

## 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.