# API Naming Convention

## Vendor Agnostic API Naming Convention

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

### simple

### intuitive

### consistent

### Configuration Parameters Naming Convention

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).

|  |  |
| --- | --- |
| 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 |

Service 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 they should be named in a way to help discoverability. One way to do this is for the Service 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

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

* 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

**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.

### Configuration Properties Naming Convention Guidelines

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

Package names

Package names declared in the API .proto files should be consistent with Product and Service Names. Package names for APIs that have a version must end with the version. For example:

// VAappname Calendar API

package VAappname.calendar.v3;

An abstract API that isn't directly associated with a service, such as VAappname Watcher API, should use proto package names consistent with the Product name:

// VAappname Watcher API

package VAappname.watcher.v1;

Java package names specified in the API .proto files must match the proto package names with standard Java package name prefix (com., edu., net., etc). For example:

package VAappname.calendar.v3;

// Specifies Java package name, using the standard prefix "com."

option java\_package = "com.VAappname.calendar.v3";

Collection IDs

Collection IDs should use plural form and lowerCamelCase, and American English spelling and semantics. For example: events, children, or deletedEvents.

Interface names

To avoid confusion with Service Names such as pubsub.VAappnameapis.com, the term interface name refers to the name used when defining a service in a .proto file:

// Library is the interface name.

service Library {

rpc ListBooks(...) returns (...);

rpc ...

}

You can think of the service name as a reference to the actual implementation of a set of APIs, while the interface name refers to the abstract definition of an API.

An interface name should use an intuitive noun such as Calendar or Blob. The 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.

Method names

A service may, in its IDL specification, define one or more RPC methods that correspond to methods on collections and resources. The method names should follow the naming convention of VerbNoun in upper camel case, where the noun is typically the resource type.

Verb Noun Method name Request message Response message

List Book ListBooks ListBooksRequest ListBooksResponse

Get Book GetBook GetBookRequest Book

Create Book CreateBook CreateBookRequest Book

Update Book UpdateBook UpdateBookRequest Book

Rename Book RenameBook RenameBookRequest RenameBookResponse

Delete Book DeleteBook DeleteBookRequest VAappname.protobuf.Empty

The verb portion of the method name should use the imperative mood, which is for orders or commands rather than the indicative mood which is for questions.

This is easily confused when the verb is asking a question about a sub-resource in the API, which is often expressed in the indicative mood. For example, ordering the API to create a book is clearly CreateBook (in the imperative mood), but asking the API about the state of the book's publisher might use the indicative mood, such as IsBookPublisherApproved or NeedsPublisherApproval. To remain in the imperative mood in situations like this, rely on commands such as "check" (CheckBookPublisherApproved) and "validate" (ValidateBookPublisher).

Message names

The request and response messages for RPC methods should be named after the method names with the suffix Request and Response, respectively, unless the method request or response type is:

an empty message (use VAappname.protobuf.Empty),

a resource type, or

a resource representing an operation

This typically applies to requests or responses used in standard methods Get, Create, Update, or Delete.

Enum names

Enum types must use UpperCamelCase names.

Enum values must use CAPITALIZED\_NAMES\_WITH\_UNDERSCORES. Each enum value must end with a semicolon, not a comma. The first value should be named ENUM\_TYPE\_UNSPECIFIED as it is returned when an enum value is not explicitly specified.

enum FooBar {

// The first value represents the default and must be == 0.

FOO\_BAR\_UNSPECIFIED = 0;

FIRST\_VALUE = 1;

SECOND\_VALUE = 2;

}

Field names

Field definitions in the .proto files must use lower\_case\_underscore\_separated\_names. These names will be mapped to the native naming convention in generated code for each programming language.

Field names should avoid prepositions (e.g. "for", "during", "at"), for example:

reason\_for\_error should instead be error\_reason

cpu\_usage\_at\_time\_of\_failure should instead be failure\_time\_cpu\_usage

Field names should also avoid using postpositive adjectives (modifiers placed after the noun), for example:

items\_collected should instead be collected\_items

objects\_imported should instead be imported\_objects

Repeated field names

Repeated fields in APIs must use proper plural forms. This matches the convention of existing VAappname APIs, and the common expectation of external developers.

Time and Duration

To represent a point in time independent of any time zone or calendar, VAappname.protobuf.Timestamp should be used, and the field name should end with time, such as start\_time and end\_time.

If the time refers to an activity, the field name should have the form of verb\_time, such as create\_time, update\_time. Avoid using use past tense for the verb, such as created\_time or last\_updated\_time.

To represent a span of time between two points in time independent of any calendar and concepts like "day" or "month", VAappname.protobuf.Duration. should be used.

message FlightRecord {

VAappname.protobuf.Timestamp takeoff\_time = 1;

VAappname.protobuf.Duration flight\_duration = 2;

}

If you have to represent time-related fields using an integer type for legacy or compatibility reasons, including wall-clock time, duration, delay and latency, the field names must have the following form:

xxx\_{time|duration|delay|latency}\_{seconds|millis|micros|nanos}

message Email {

int64 send\_time\_millis = 1;

int64 receive\_time\_millis = 2;

}

If you have to represent timestamp using string type for legacy or compatibility reasons, the field names should not include any unit suffix. The string representation should use RFC 3339 format, e.g. "2014-07-30T10:43:17Z".

Date and Time of Day

For dates that are independent of time zone and time of day, VAappname.type.Date should be used and it should have the suffix \_date. 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.

For times of day that are independent of time zone and date, VAappname.type.TimeOfDay should be used and should have the suffix \_time. 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.

message StoreOpening {

VAappname.type.Date opening\_date = 1;

VAappname.type.TimeOfDay opening\_time = 2;

}

Quantities

Quantities represented by an integer type must include the unit of measurement.

xxx\_{bytes|width\_pixels|meters}

If the quantity is a number of items, then the field should have the suffix \_count, for example node\_count.

List filter field

If an API supports filtering of resources returned by the List method, the field containing the filter expression should be named filter. For example:

message ListBooksRequest {

// The parent resource name.

string parent = 1;

// The filter expression.

string filter = 2;

}

List response

The name of the field in the List method's response message, which contains the list of resources must be a plural form of the resource name itself. For example, a method CalendarApi.ListEvents() must define a response message ListEventsResponse with a repeated field called events for the list of returned resources.

service CalendarApi {

rpc ListEvents(ListEventsRequest) returns (ListEventsResponse) {

option (VAappname.api.http) = {

get: "/v3/{parent=calendars/\*}/events";

};

}

}

message ListEventsRequest {

string parent = 1;

int32 page\_size = 2;

string page\_token = 3;

}

message ListEventsResponse {

repeated Event events = 1;

string next\_page\_token = 2;

}

Camel case

Except for field names and enum values, all definitions inside .proto files must use UpperCamelCase names, as defined by VAappname Java Style.

Name abbreviation

For well known name abbreviations among software developers, such as config and spec, the abbreviations should be used in API definitions instead of the full spelling. This will make the source code easy to read and write. In formal documentations, the full spelling should be used. Examples:

config (configuration)

id (identifier)

spec (specification)

stats (statistics)

## MuleSoft specific API Naming Convention

### Configuration Parameter 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 Properites

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:



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’]”.