[API Naming Convention 2](#_Toc520754091)

[Vendor Agnostic API Naming Convention 2](#_Toc520754092)

[Objects Naming Convention 2](#_Toc520754093)

[MuleSoft-specific API Naming Convention 3](#_Toc520754094)

[Objects and Methods Naming Conventin 3](#_Toc520754095)

# API Naming Convention

## Vendor Agnostic API Naming Convention

### Objects Naming Convention

Objects and methods follow specific naming convention based on programming language of choice and best practices for naming objects and methods across platforms. Different programming languages have different naming convention for defining objects and object methods and attributes. For example, in .NET it is common to capitalize object names. In addition, Microsoft encourages to also prefix all objects with a three-letter prefix indicating the type of the object. .NET classes are usually capitalized, without any prefixes.

For example, to declare a panel object for “Group” in C#, one would call it pnlGroup. To declare a Submit button, one would declare an object named btnSubmit. A custom class of Customer will be defined as Customer, without any prefix. So only object instances of a particular default class like Button or a custom class like Customer will be prefixed with a three-letter type prefix.

With Java programming language, objects and classes typically follow a different naming convention:

* For custom Java classes, it is always the best practice to capitalize a class name. All default Java classes are capitalized (e.g., String, Integer, etc), so all custom classes should follow the same naming convention and be capitalized
* For Java objects that are instances of a given class, it is the best practice to lower-case the object name. For example, customer object of a Customer class.

In Java and .NET, all objects, classes, and methods should follow Hungarian naming convention when concatenating multiple words together.

In addition, it is not recommended to separate the words in the name with dashes or underscores. So instead of my\_customer object name, it should always be myCustomer.

Now in other programming languages that are typically used to write microservice APIs such as NodeJS, the naming conventions are completely different from OOO languages like .NET and Java. Node follows JavaScript naming conventions for objects, classes and methods. For object and class names, it is recommended to follow the JavaScript naming convention of having them all lower-cased, with a Hungarian convention when concatenating multiple words together in the names.

For example, myCustomer.js for an object name or customer for a class name.

Method Naming Convention

Method names are pretty consistently named across languages such as .NET, Java, and NodeJS. All methods should follow be lower-cased, following Hungarian notation. All methods should start with the operation type, followed by the name of the resource or object. For example, getCustomerName or deleteCustomerRecord.

Method input parameters should follow the same convention as names of the objects in each language. So in .NET, the method input parameter should be prefixed with a three-letter type, followed by a camel-cased capitalized name. For example, tblTable. In Java, it is a Hungarian notation lower-cased name. For example, customerName. In Node, it is similar to Java.

## MuleSoft-specific API Naming Convention

### Objects and Methods Naming Convention

Mulesoft follows the Java standards for naming custom classes and objects since Mulesoft is built on a Java Spring-based framework. So all custom classes should following Java naming convention of capitalizing the name of a class, and following camelCase to concatenate the words together. For example, a custom Java class to define a customer’s primary address in Mule application should be called PrimaryAddress.

Object instances in Mule applications should follow Java-based object naming convention. All object names should always start with lower case, and follow Hungarian naming convention. For example, customerAddress.

All custom methods in a Mule application should follow Java method naming convention. For example, getCustomerName or deleteCustomerRecord.

Finally, it is worth mentioning that Mule also supports scripting languages to script routines and operations inside custom policies or Dataweave transformations. There are choices of a scripting language of choice, with Groovy being the mostly commonly used scripting language in Mule. As Groovy follows Java naming convention, all Java object and method naming conventions should be closely following inside Groovy scripts as well. Python and Ruby are other scripting languages that can be used in Mule, but they are not as widely used as Groovy.