

[Cloud APIs](https://cloud.google.com/apis/) (<https://cloud.google.com/apis/>)

Naming Conventions

In order 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

This includes names of interfaces, resources, collections, methods, and messages.

Since many developers are not native English speakers, one goal of these naming conventions is to ensure that the majority of developers can easily understand an API. It does this by encouraging the use of a simple, consistent, and small vocabulary when naming methods and resources.

- Names used in APIs **should** be in correct American English. For example, license (instead of licence), color (instead of colour).
- Commonly accepted short forms or abbreviations of long words **may** be used for brevity. For example, API is preferred over Application Programming Interface.
- Use intuitive, familiar terminology where possible. For example, when describing removing (and destroying) a resource, delete is preferred over erase.
- Use the same name or term for the same concept, including for concepts shared across APIs.
- Avoid name overloading. Use different names for different concepts.
- Avoid overly general names that are ambiguous within the context of the API and the larger ecosystem of Google APIs. They can lead to misunderstanding of API concepts. Rather, choose specific names that accurately describe the API concept. This is particularly important for names that define first-order API elements, such as resources. There is no definitive list of names to avoid, as every name must be evaluated in the context of other names. Instance, info, and service are examples of

names that have been problematic in the past. Names chosen should describe the API concept clearly (for example: instance of what?) and distinguish it from other relevant concepts (for example: does "alert" mean the rule, the signal, or the notification?).

- Carefully consider use of names that may conflict with keywords in common programming languages. Such names **may** be used but will likely trigger additional scrutiny during API review. Use them judiciously and sparingly.

Product names

Product names refer to the product marketing names of APIs, such as *Google Calendar API*. Product names **must** be consistently used by APIs, UIs, documentation, Terms of Service, billing statements, commercial contracts, etc. Google APIs **must** use product names approved the product and marketing teams.

The table below shows examples of all related API names and their consistency. See further below on this page for more details on the respective names and their conventions.

API Name	Example
Product Name	Google Calendar API
Service Name	<code>calendar.googleapis.com</code>
Package Name	<code>google.calendar.v3</code>
Interface Name	<code>google.calendar.v3.CalendarService</code>
Source Directory	<code>//google/calendar/v3</code>
API Name	<code>calendar</code>

Service names

Service names **should** be syntactically valid DNS names (as per [RFC 1035](http://www.ietf.org/rfc/rfc1035.txt) (<http://www.ietf.org/rfc/rfc1035.txt>)) which can be resolved to one or more network addresses. The service names of public Google APIs follow the pattern: `xxx.googleapis.com`. For example, the service name of the Google Calendar is `calendar.googleapis.com`.

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.googleapis.com` and `buildresults.googleapis.com` are both services that are part of the Google Build API.

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:

```
// Google Calendar API
package google.calendar.v3;
```



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

```
// Google Watcher API
package google.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 google.calendar.v3;
```



```
// Specifies Java package name, using the standard prefix "com."
option java_package = "com.google.calendar.v3";
```

Collection IDs

Collection IDs (https://cloud.google.com/apis/design/resource_names#collection_id) **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 (`#service_names`) such as `pubsub.googleapis.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	google.protobuf.Empty

The verb portion of the method name **should** use the imperative mood (https://en.wikipedia.org/wiki/Imperative_mood#English), 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 `google.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 Google APIs, and the common expectation of external developers.

Time and Duration

To represent a point in time independent of any time zone or calendar, `google.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", `google.protobuf.Duration` **should** be used.

```
message FlightRecord {  
  google.protobuf.Timestamp takeoff_time = 1;  
  google.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, `google.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, `google.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 {
  google.type.Date opening_date = 1;
  google.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 (google.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 Google Java Style (<https://google.github.io/styleguide/javaguide.html#s5.3-camel-case>).

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)

[Previous](#)[Next](#)[Errors](/apis/design/errors) (/apis/design/errors)[Design Patterns](/apis/design/design_patterns)

(/apis/design/design_patterns)

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](http://creativecommons.org/licenses/by/3.0/) (<http://creativecommons.org/licenses/by/3.0/>), and code samples are licensed under the [Apache 2.0 License](http://www.apache.org/licenses/LICENSE-2.0) (<http://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated February 1, 2018.