이름 지정 규칙

다수의 API에서 오랜 시간이 지나더라도 일관적인 개발자 환경을 제공하려면 API에서 사용하는 이름이 모두 다음과 **같아야 합니다**.

- 단순해야 합니다.
- 직관적이어야 합니다.
- 일관적이어야 합니다.

여기에는 인터페이스, 리소스, 컬렉션, 메서드 및 메시지의 이름도 포함됩니다.

대부분 개발자들은 영어를 모국어로 사용하는 사람이 아니기 때문에 이러한 이름 지정 규칙은 대다수 개발자들이 API를 쉽게 이해할 수 있도록 하는 데 한 가지 목적이 있습니다. 이를 위해서는 메서드와 리소스의 이름을 지정할 때 단순하고, 일관적이고, 적은 어휘를 사용해야 합니다.

- API에서 사용하는 이름은 정확한 미국 영어**여야 합니다**. 예를 들어 licence가 아닌 license를, 그리고 colour가 아닌 color를 사용하세요.
- 긴 단어의 경우 일반적으로 인정되는 줄임말 또는 축약어를 **사용할 수 있습니다**. 예를 들어 Application Programming Interface보다는 API가 권장됩니다.
- 가능하다면 직관적이고 친숙한 용어를 사용하세요. 예를 들어 리소스를 삭제한다고 할 때는 erase보다는 delete가 권장됩니다.
- 여러 API에서 공유하는 개념을 포함해 동일한 개념일 때는 이름 또는 용어도 동일하게 사용하세요.
- 이름 중복을 피하세요. 개념에 따라 이름도 다르게 사용해야 합니다.
- API와 더욱 광범위한 Google API 생태계 내에서 지나치게 일반적인 모호한 이름을 사용하지 마세요. 이로 인해 API 개념을 혼동할 수 있습니다. 오히려 API 개념을 정확하게 설명하는 구체적인 이름을 선택하는 것이 좋습니다. 특히 리소스 같이 1차 API 요소를 정의하는 이름일 때는 더욱 중요합니다. 모든 이름은 다른 이름의 컨텍스트에서 평가해야 하기 때문에 피해야 할 이름을 명확하게 정해놓은 목록은 없습니다. 예를 들어 instance, info, service 등이 과거에 문제를 일으켰던 이름이기는 합니다. 선택한 이름은 API 개념을 명확하게 설명하는(예: 어떤 인스턴스입니까?) 동시에 다른 관련 개념을 구별해야 합니다(예: '알림'이 규칙입니까? 신호입니까? 아니면 말 그대로 알림입니까?).
- 일반적인 프로그래밍 언어의 키워드와 충돌을 일으킬 수 있는 이름을 사용할 때는 주의해야 합니다. 이러한 이름은 사용이 **가능하기는 하지만** API를 검토할 때 더욱 철저한 검사가 필요할 수 있습니다. 따라서 사용할 때는 신중하게 선택해야 하지만 최대한 사용하지 않는 것이 좋습니다.

제품 이름

제품 이름이란 *Google Calendar API* 같이 API의 제품 마케팅 이름을 말합니다. 제품 이름은 API, UI, 문서, 서비스 약관, 청구 명세서, 상업 계약서 등에서 일관적으로 **사용되어야 합니다**. 또한 Google API에서는 제품 팀과 마케팅 팀에서 승인한 제품 이름만 **사용해야 합니다**.

아래 표는 관련된 API 이름과 이름의 일관성을 예로 나타낸 것입니다. 각 이름과 이름 규칙에 대한 자세한 내용은 이 페이지 아래 내용을 참조하세요.

API 이름	예시
제품 이름	`Google Calendar API`
서비스 이름	`calendar.googleapis.com`
패키지 이름	`google.calendar.v3`
인터페이스 이름	`google.calendar.v3.CalendarService`
소스 디렉터리	`//google/calendar/v3`
API 이름	`calendar`

서비스 이름

서비스 이름은 **RFC 1035** 기준으로 구문이 유효하며 네트워크 주소 1개 이상으로 연결될 수 있는 DNS 이름이어야 합니다 (http://www.ietf.org/rfc/rfc1035.txt). 공개 Google API의 서비스 이름은 `xxx.googleapis.com` 패턴을 따릅니다. 예를 들어 Google 캘린더의 서비스 이름은 `calendar.googleapis.com`입니다.

API 1개가 다수의 서비스로 구성되어 있으면 검색이 가능하도록 서비스 이름을 **지정해야 합니다**. 이를 위한 한 가지 방법은 서비스 이름마다 공통 프리픽스를 추가하는 것입니다. 예를 들어 `build.googleapis.com` 및 `buildresults.googleapis.com` 서비스는 모두 Google Build API의 일부인 서비스입니다.

패키지 이름

API .proto 파일에서 선언하는 패키지 이름은 제품 이름 및 서비스 이름과 **일치해야 합니다**. 패키지 이름에는 단수형 구성요소 이름을 **사용해야 하며** 단수형 및 복수형 구성요소 이름의 혼합 사용을 피해야 합니다. 패

키지 이름에 밑줄을 사용해서는 **안 됩니다**. 버전이 지정된 API의 패키지 이름은 해당 버전으로 **끝나야 합니다**. 예를 들면 다음과 같습니다.

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

Google Watcher API 같이 서비스와 직접 연결되지 않아 추상적인 API는 다음과 같이 제품 이름과 일치하는 proto 패키지 이름을 **사용해야 합니다**.

```
// Google Watcher API
package google.watcher.v1;
```

API .proto 파일에서 지정된 자바 패키지 이름은 표준 자바 패키지 이름 프리픽스(`com.`, `edu.`, `net.` 등)가 포함된 proto 패키지 이름과 **일치해야 합니다**. 예를 들면 다음과 같습니다.

```
package google.calendar.v3;

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

컬렉션 ID

컬렉션 ID (/apis/design/resource_names#collection_id)는 복수형, **`lowerCamelCase`**, 미국 영어의 철자와 시맨틱스를 **사용해야 합니다**. 예를 들면 **`events`**, **`children`**, **`deletedEvents`**입니다.

인터페이스 이름

*인터페이스 이름*은 **`pubsub.googleapis.com`**과 같은 서비스 이름 (#service_names)과 혼동을 방지하기 위해 .proto 파일에서 **`service`**를 정의할 때 사용되는 이름입니다.

```
// Library is the interface name.
service Library {
```

```
  rpc ListBooks(...) returns (...);
  rpc ...
}
```

*서비스 이름*이 API 집합의 실제 구현체를 나타내는 것이라면 *인터페이스 이름*은 API의 추상적 정의를 나타낸다고 볼 수 있습니다.

인터페이스 이름은 Calendar 또는 Blob 같이 직관적인 명사를 **사용해야 합니다**. 또한 프로그래밍 언어 및 런타임 라이브러리에서 이미 확립된 개념(예: File)과 **충돌해서는 안 됩니다**.

드물지만 *인터페이스 이름*이 API에 속한 다른 이름과 충돌할 경우에는 서픽스(예: `Api` 또는 `Service`)를 사용하여 **구분해야 합니다**.

메서드 이름

서비스는 IDL 사양에 따라 컬렉션 및 리소스의 메서드와 일치하는 RPC 메서드를 1개 이상 **정의할 수 있습니다**. 메서드 이름을 지정할 때는 단어마다 첫 글자를 대문자로 쓰는 카멜 표기법에 따라 `VerbNoun` 방식의 이름 지정 규칙을 **따라야 합니다**. 여기서 일반적으로 명사는 리소스 유형을 말합니다.

동사	명사	메서드 이름	요청 메시지	응답 메시지
`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`

메서드 이름의 동사 부분에서는 질문을 위한 직설법보다는 지시 또는 명령을 위한 명령법 (https://en.wikipedia.org/wiki/Imperative_mood#English)을 **사용해야 합니다**.

표준 메서드의 경우 메서드 이름에서 명사 부분은 `List`를 제외한 모든 메서드는 **단수형이어야 하며** `List`는 **복수형이어야 합니다**. 커스텀 메서드의 경우 명사는 필요에 따라 단수형 또는 복수형이 **될 수 있습니다**. 일괄 메서드는 복수형 명사를 **사용해야 합니다**.

의 사례는 프로토콜 버퍼의 RPC 이름을 나타냅니다. HTTP/JSON URI 서픽스는 **`:lowerCamelCase`**를 사용합니다.

동사가 API의 하위 리소스에 대해 물을 때는 직설법으로 표현되는 경우가 많기 때문에 쉽게 헷갈릴 수 있습니다. 예를 들어 API에 책을 만들라고 지시할 때는 분명히 명령법으로 **`CreateBook`**입니다. 하지만 책의 발행인 상태에 대해 API에게 물을 때는 **`IsBookPublisherApproved`** 또는 **`NeedsPublisherApproval`**과 같은 직설법을 사용할 수 있습니다. 이러한 상황에서 명령법을 유지하려면 'check'(**`CheckBookPublisherApproved`**)와 'validate'(**`ValidateBookPublisher`**)와 같은 명령어를 사용합니다.

메서드 이름에는 'For', 'With', 'At', 'To' 등의 전치사가 포함되지 **않아야 합니다**. 일반적으로 메서드 이름에 전치사가 있다면, 새 메서드를 사용하는 대신 기존 메서드에 필드를 추가하거나 메서드에 별도의 동사를 사용해야 하는 경우입니다.

예를 들어 **`CreateBook`** 메시지가 이미 있는데 **`CreateBookFromDictation`**을 추가하려는 경우 **`TranscribeBook`** 메서드를 대신 사용하는 것이 좋습니다.

메시지 이름

메시지 이름은 짧고 간결해야 **합니다**. 불필요하거나 중복되는 단어를 사용하지 마세요. 형용사가 없는 경우에 해당하는 메시지가 없다면 해당 형용사는 대개 생략 가능합니다. 예를 들어 *비공유* 프록시 설정이 없으면 **`SharedProxySettings`**의 **`Shared`**는 불필요합니다.

메시지 이름은 'With', 'For' 등의 전치사를 포함하지 **않아야 합니다**. 전치사가 있는 메시지 이름은 메시지의 선택적 필드로 더 잘 표현되는 것이 일반적입니다.

요청 및 응답 메시지

RPC 메서드에 사용되는 요청 및 응답 메시지의 이름을 메서드 이름 뒤에 각각 서픽스 **`Request`** 및 **`Response`**를 추가하여 **지정해야 합니다**. 단, 메서드 요청 또는 응답 유형이 다음과 같은 경우는 예외입니다.

- 빈 메시지(**`google.protobuf.Empty`** 사용)
- 리소스 유형
- 작업을 의미하는 리소스

이러한 규칙은 일반적으로 표준 메서드인 **`Get`**, **`Create`**, **`Update`** 또는 **`Delete`**에 사용되는 요청 또는 응답에 적용됩니다.

열거형 이름

열거형은 UpperCamelCase 이름을 **사용해야 합니다**.

열거형 값은 CAPITALIZED_NAMES_WITH_UNDERSCORES를 **사용해야 합니다**. 각 열거형 값은 쉼표가 아닌 세미콜론으로 **끝나야 합니다**. 첫 번째 값은 열거형 값을 명시적으로 지정하지 않을 때 반환되므로 이름을 ENUM_TYPE_UNSPECIFIED로 **지정해야 합니다**.

```
enum FooBar {
  // The first value represents the default and must be == 0.
  FOO_BAR_UNSPECIFIED = 0;
  FIRST_VALUE = 1;
  SECOND_VALUE = 2;
}
```

래퍼

`0` 값이 `UNSPECIFIED`가 아닌 다른 의미를 갖는 proto2 열거형을 캡슐화하는 메시지는 서픽스 `Value`를 추가하여 이름을 **지정해야 하며** `value`라는 필드 하나만 가져야 합니다.

```
enum OldEnum {
  VALID = 0;
  OTHER_VALID = 1;
}
message OldEnumValue {
  OldEnum value = 1;
}
```

필드 이름

.proto 파일에서 필드를 정의할 때는 lower_case_underscore_separated_names를 **사용해야 합니다**. 이러한 이름은 각 프로그래밍 언어마다 생성되는 코드에서 기본적인 이름 지정 규칙으로 매핑됩니다.

필드 이름은 'for', 'during', 'at' 등의 전치사를 포함하지 **않아야 합니다**. 예를 들면 다음과 같습니다.

- `error_reason`가 아닌 `reason_for_error`여야 합니다.

- **`failure_time_cpu_usage`**가 아닌 **`cpu_usage_at_time_of_failure`**여야 합니다.

필드 이름에 후치 형용사(명사 뒤에 위치하는 수식어)를 사용해서는 **안 됩니다**. 예를 들면 다음과 같습니다.

- **`collected_items`**가 아닌 **`items_collected`**여야 합니다.
- **`imported_objects`**가 아닌 **`objects_imported`**여야 합니다.

반복되는 필드 이름

API에서 반복되는 필드는 적절한 복수 형태를 **사용해야 합니다**. 이는 기존 Google API의 규칙과 일치할 뿐만 아니라 외부 개발자들도 공통적으로 기대하는 것입니다.

시간과 지속 시간

시간대 또는 캘린더와 상관없이 특정 시점을 나타내려면 **`google.protobuf.Timestamp`**를 **사용해야 하고** 필드 이름은 **`time`**(예: **`start_time`** 및 **`end_time`**)으로 **끝나야 합니다**.

시간이 활동을 나타낼 때는 필드 이름이 **`verb_time`** 형태(예: **`create_time`**, **`update_time`**)여야 **합니다**. 동사의 과거 시제(예: **`created_time`** 또는 **`last_updated_time`**)를 사용하지 마세요.

'일' 또는 '월'과 같은 캘린더와 개념에 관계없이 두 시점 사이의 지속 시간을 나타내려면 **`google.protobuf.Duration`**을 **사용해야 합니다**.

```
message FlightRecord {
  google.protobuf.Timestamp takeoff_time = 1;
  google.protobuf.Duration flight_duration = 2;
}
```

레거시 또는 호환성 때문에 벽시계 시간, 지속 시간, 지연 시간 같은 시간 관련 필드를 정수 형식으로 나타내야 할 경우에는 필드 이름이 다음과 같은 형태가 **되어야 합니다**.

```
xxx_{time|duration|delay|latency}_{seconds|millis|micros|nanos}
```

```
message Email {
  int64 send_time_millis = 1;
  int64 receive_time_millis = 2;
```

```
}
```

레거시 또는 호환성 때문에 문자열 형식을 사용해 타임스탬프를 나타내야 할 경우에는 필드 이름에 단위 서픽스가 포함되어서는 **안 됩니다**. 문자열을 표현할 때는 RFC 3339 형식(예: '2014-07-30T10:43:17Z')을 **사용해야 합니다**.

날짜와 시간

시간대 및 시간에 관계없는 날짜의 경우 `google.type.Date`를 **사용해야 하며** 서픽스 `_date`가 있어야 합니다. 날짜를 문자열로 표현해야 할 경우에는 ISO 8601 날짜 형식인 YYYY-MM-DD를 따라야 합니다(예: 2014-07-30).

시간대와 날짜와 관계없는 시간의 경우 `google.type.TimeOfDay`를 **사용해야 하고** 서픽스 `_time`이 있어야 합니다. 시간을 문자열로 표현해야 할 경우에는 ISO 8601 24시간 형식인 HH:MM:SS[.FFF]를 따라야 합니다(예: 14:55:01.672).

```
message StoreOpening {
  google.type.Date opening_date = 1;
  google.type.TimeOfDay opening_time = 2;
}
```

수량

정수 형식의 수량에는 측정 단위가 **포함되어야 합니다**.

```
xxx_{bytes|width_pixels|meters}
```

수량이 항목 개수인 경우에는 필드에 서픽스 `_count`를 **추가해야 합니다**(예: `node_count`).

필터 필드 나열

API가 `List` 메서드에서 반환되는 리소스의 필터링을 지원할 경우 필터 표현식을 포함하는 필드 이름은 `filter`**여야 합니다**. 예를 들면 다음과 같습니다.

```
message ListBooksRequest {
  // The parent resource name.
  string parent = 1;

  // The filter expression.
  string filter = 2;
}
```

List 응답

`List` 메서드의 응답으로 리소스 목록이 포함된 메시지의 필드 이름은 리소스 이름의 **복수형이어야 합니다**. 예를 들어 **`CalendarApi.ListEvents()`** 메서드는 반환된 리소스 목록에 **`events`**라는 반복 필드가 있는 **`ListEventsResponse`** 응답 메시지를 **정의해야 합니다**.

```
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;
}
```

카멜 표기법

`.proto` 파일의 모든 정의는 필드 이름과 열거형 값을 제외하고 Google 자바 스타일 (https://google.github.io/styleguide/javaguide.html#s5.3-camel-case)의 정의대로 UpperCamelCase 이름을

사용해야 합니다.

이름 약어

`config` 및 `spec`과 같은 소프트웨어 개발자들이 잘 알고 있는 이름 약어의 경우에는 API 정의에서 전체 철자 대신에 약어를 **사용해야 합니다**. 이렇게 하면 소스 코드를 쉽게 읽고 쓸 수 있기 때문입니다. 하지만 공식 문서에서는 전체 철자를 **사용해야 합니다**. 예를 들면 다음과 같습니다.

- config(configuration)
- id(identifier)
- spec(specification)
- stats(statistics)

이전
← 오류 (/apis/design/errors)

다음
설계 패턴 (/apis/design/design_patterns) →