Cloud Vision API Product Search

Package google.rpc

Index

- `Code` (https://cloud.google.com/vision/product-search/docs/reference/rpc/google.rpc#google.rpc.Code) (enum)
- `Status` (https://cloud.google.com/vision/product-search/docs/reference/rpc/google.rpc#google.rpc.Status) (message)

Code

The canonical error codes for gRPC APIs.

Sometimes multiple error codes may apply. Services should return the most specific error code that applies. For example, prefer `OUT_OF_RANGE` over `FAILED_PRECONDITION` if both codes apply. Similarly prefer `NOT_FOUND` or `ALREADY_EXISTS` over `FAILED_PRECONDITION`.

Enums	
OK	Not an error; returned on success HTTP Mapping: 200 OK
CANCELLED	The operation was cancelled, typically by the caller. HTTP Mapping: 499 Client Closed Request
UNKNOWN	Unknown error. For example, this error may be returned when a **Status** value received from another address space belongs to an error space that is not known in this address space. Also errors raised by APIs that do not return enough error information may be converted to this error. HTTP Mapping: 500 Internal Server Error

Enums	
INVALID_ARGUMENT	The client specified an invalid argument. Note that this differs from **FAILED_PRECONDITION**. **INVALID_ARGUMENT** indicates arguments that are problematic regardless of the state of the system (e.g., a malformed file name). HTTP Mapping: 400 Bad Request
DEADLINE_EXCEEDED	The deadline expired before the operation could complete. For operations that change the state of the system, this error may be returned even if the operation has completed successfully. For example, a successful response from a server could have been delayed long enough for the deadline to expire. HTTP Mapping: 504 Gateway Timeout
NOT_FOUND	Some requested entity (e.g., file or directory) was not found. Note to server developers: if a request is denied for an entire class of users, such as gradual feature rollout or undocumented whitelist, **NOT_FOUND** may be used. If a request is denied for some users within a class of users, such as user-based access control, **PERMISSION_DENIED** must be used. HTTP Mapping: 404 Not Found
ALREADY_EXISTS	The entity that a client attempted to create (e.g., file or directory) already exists. HTTP Mapping: 409 Conflict
PERMISSION_DENIED	The caller does not have permission to execute the specified operation. **PERMISSION_DENIED** must not be used for rejections caused by exhausting some resource (use **RESOURCE_EXHAUSTED** instead for those errors). **PERMISSION_DENIED** must not be used if the caller can not be identified (use **UNAUTHENTICATED** instead for those errors). This error code does not imply the request is valid or the requested entity exists or satisfies other pre-conditions. HTTP Mapping: 403 Forbidden
UNAUTHENTICATED	The request does not have valid authentication credentials for the operation. HTTP Mapping: 401 Unauthorized

Enums	
RESOURCE_EXHAUSTED	Some resource has been exhausted, perhaps a per-user quota, or perhaps the entire file system is out of space. HTTP Mapping: 429 Too Many Requests
FAILED_PRECONDITION	The operation was rejected because the system is not in a state required for the operation's execution. For example, the directory to be deleted is non-empty, an rmdir operation is applied to a non-directory, etc. Service implementors can use the following guidelines to decide between **FAILED_PRECONDITION**, **ABORTED**, and **UNAVAILABLE**: (a) Use **UNAVAILABLE** if the client can retry just the failing call. (b) Use **ABORTED** if the client should retry at a higher level (e.g., when a client-specified test-and-set fails, indicating the client should restart a read-modify-write sequence). (c) Use **FAILED_PRECONDITION** if the client should not retry until the system state has been explicitly fixed. E.g., if an "rmdir" fails because the directory is non-empty, **FAILED_PRECONDITION** should be returned since the client should not retry unless the files are deleted from the directory. HTTP Mapping: 400 Bad Request
ABORTED	The operation was aborted, typically due to a concurrency issue such as a sequencer check failure or transaction abort. See the guidelines above for deciding between **FAILED_PRECONDITION**, **ABORTED**, and **UNAVAILABLE**. HTTP Mapping: 409 Conflict

Enums	
OUT_OF_RANGE	The operation was attempted past the valid range. E.g., seeking or reading past end-of-file. Unlike **INVALID_ARGUMENT**, this error indicates a problem that may be fixed if the system state changes. For example, a 32-bit file system will generate **INVALID_ARGUMENT** if asked to read at an offset that is not in the range [0,2^32-1], but it will generate **OUT_OF_RANGE** if asked to read from an offset past the current file size. There is a fair bit of overlap between **FAILED_PRECONDITION** and **OUT_OF_RANGE**. We recommend using **OUT_OF_RANGE** (the more specific error) when it applies so that callers who are iterating through a space can easily look for an **OUT_OF_RANGE** error to detect when they are done. HTTP Mapping: 400 Bad Request
UNIMPLEMENTED	The operation is not implemented or is not supported/enabled in this service. HTTP Mapping: 501 Not Implemented
INTERNAL	Internal errors. This means that some invariants expected by the underlying system have been broken. This error code is reserved for serious errors. HTTP Mapping: 500 Internal Server Error
UNAVAILABLE	The service is currently unavailable. This is most likely a transient condition, which can be corrected by retrying with a backoff. Note that it is not always safe to retry non-idempotent operations. See the guidelines above for deciding between **FAILED_PRECONDITION**, **ABORTED**, and **UNAVAILABLE**. HTTP Mapping: 503 Service Unavailable
DATA_LOSS	Unrecoverable data loss or corruption. HTTP Mapping: 500 Internal Server Error

Status

The `Status` type defines a logical error model that is suitable for different programming environments, including REST APIs and RPC APIs. It is used by gRPC (https://github.com/grpc). Each `Status` message contains three pieces of data: error code, error message, and error details.

You can find out more about this error model and how to work with it in the API Design Guide (https://cloud.google.com/apis/design/errors).

Fields	
`code`	`int32` The status code, which should be an enum value of **`google.rpc.Code`** (https://cloud.google.com/vision/product-search/docs/reference/rpc/google.rpc#google.rpc.Code) .
`message`	`string` A developer-facing error message, which should be in English. Any user-facing error message should be localized and sent in the **`google.rpc.Status.details`** (https://cloud.google.com/vision/product-search/docs/reference/rpc/google.rpc#google.rpc.Status.FIELDS.repeated.google.protobuf.Any.google.rpc.Status.details) field, or localized by the client.
`details[]`	Any (https://developers.google.com/protocol-buffers/docs/reference/google.protobuf#google.protobuf.Any) A list of messages that carry the error details. There is a common set of message types for APIs to use.