Error Handling

This page describes how gRPC deals with errors, including gRPC's built-in error codes. Example code in different languages can be found here.

Standard error model

As you'll have seen in our concepts document and examples, when a gRPC call completes successfully the server returns an OK status to the client (depending on the language the OK status may or may not be directly used in your code). But what happens if the call isn't successful?

If an error occurs, gRPC returns one of its error status codes instead, with an optional string error message that provides further details about what happened. Error information is available to gRPC clients in all supported languages.

Richer error model

The error model described above is the official gRPC error model, is supported by all gRPC client/server libraries, and is independent of the gRPC data format (whether protocol buffers or something else). You may have noticed that it's quite limited and doesn't include the ability to communicate error details.

If you're using protocol buffers as your data format, however, you may wish to consider using the the richer error model developed and used by Google as described here. This model enables servers to return and clients to consume additional error details expressed as one or more protobuf messages. It further specifies a standard set of error message types to cover the most common needs (such as invalid parameters, quota violations, and stack traces). The protobuf binary encoding of this extra error information is provided as trailing metadata in the response.

This richer error model is already supported in the C++, Go, Java, Python, and Ruby libraries, and at least the grpc-web and Node.js libraries have open issues requesting it. Other language libraries may add support in the future if there's demand, so check their github repos if interested. Note however that the grpc-core library written in C will not likely ever support it since it is purposely data format agnostic.

You could use a similar approach (put error details in trailing response metadata) if you're not using protocol buffers, but you'd likely need to find or develop library support for accessing this data in order to make practical use of it in your APIs.

There are important considerations to be aware of when deciding whether to use such an extended error model, however, including:

- library implementations of the extended error model may not be consistent across languages in terms of requirements for and expectations of the error details payload

- existing proxies, loggers, and other standard HTTP request processors don't have visibility into the error details and thus wouldn't be able to leverage them for monitoring or other purposes
- additional error detail in the trailers interferes with head-of-line blocking, and will decrease HTTP/2 header compression efficiency due to more frequent cache misses
- larger error detail payloads may run into protocol limits (like max headers size), effectively losing the original error

Error status codes

Errors are raised by gRPC under various circumstances, from network failures to unauthenticated connections, each of which is associated with a particular status code. The following error status codes are supported in all gRPC languages.

General errors

Case	Status code
Client application cancelled the request	GRPC_STATUS_CANCELLED
Deadline expired before server returned status	GRPC_STATUS_DEADLINE_EXCEEDED
Method not found on server	GRPC_STATUS_UNIMPLEMENTED
Server shutting down	GRPC_STATUS_UNAVAILABLE
Server threw an exception (or did something other than returning a status code to terminate the RPC)	GRPC_STATUS_UNKNOWN

Network failures

Case	Status code
No data transmitted before deadline expires. Also applies to cases where some data is transmitted and no other failures are detected before the deadline expires	GRPC_STATUS_DEADLINE_EXCEEDED
Some data transmitted (for example, the request metadata has been written to the TCP connection) before the connection breaks	GRPC_STATUS_UNAVAILABLE

Protocol errors

Case	Status code
Could not decompress but compression algorithm supported	GRPC_STATUS_INTERNAL
Compression mechanism used by client not supported by the server	GRPC_STATUS_UNIMPLEMENTED
Flow-control resource limits reached	GRPC_STATUS_RESOURCE_EXHAUSTED
Flow-control protocol violation	GRPC_STATUS_INTERNAL
Error parsing returned status	GRPC_STATUS_UNKNOWN
Unauthenticated: credentials failed to get metadata	GRPC_STATUS_UNAUTHENTICATED
Invalid host set in authority metadata	GRPC_STATUS_UNAUTHENTICATED
Error parsing response protocol buffer	GRPC_STATUS_INTERNAL
Error parsing request protocol buffer	GRPC_STATUS_INTERNAL