Protocol Buffers Schema compiler

START HERE

`protoc --version`
Check the compiler

`protoc -I. --python_out=. <file>.proto`
Generate Python code

`protoc -I. --go_out=. <file>.proto`
Generate Go code

`protoc -I. --cpp_out=. <file>.proto`
Generate C++ code

`protoc -I. --descriptor_set_out=<f>.pb <file>.proto`
Write a descriptor set

`protoc -I. --encode=<Msg> <file>.proto`
Text in, binary out

`protoc -I. --decode=<Msg> <file>.proto`
Binary in, text out

`protoc --decode_raw`
Dump binary with no schema

.proto file schema → `protoc` → **Generated code** classes, stubs → `your app` → **Binary wire data** compact, tagged by field number

CONCEPTS

protoc only reads .proto files and hands them to a generator; each --<lang>_out flag picks one generator.

The field number, not the name, identifies a field on the wire. Never change or reuse one.

Go, gRPC and many others are plugins: protoc runs protoc-gen-<name> from PATH for --<name>_out.

INVOKE PROTOC

`protoc --version`
Print the compiler version

`protoc -I <dir> <file>.proto`
Add an import search path

`protoc --proto_path=<dir> <file>.proto`
Long form of -I

`protoc --python_out=<dir> <file>.proto`
Generate Python modules

`protoc --cpp_out=<dir> <file>.proto`
Generate C++ sources

`protoc --java_out=<dir> <file>.proto`
Generate Java classes

INVOKE PROTOC (cont.)

`protoc --go_out=<dir> --go_opt=paths=source_relative <file>.proto`
Generate Go (needs protoc-gen-go)

`protoc --pyi_out=<dir> <file>.proto`
Generate Python type stubs

`protoc --<lang>_out=<opts>:<dir> <file>.proto`
Pass options to a generator
Built-in generators take a comma-separated option list before the colon.

`protoc --plugin=protoc-gen-<name>=<path> --<name>_out=<dir> <file>.proto`
Use a plugin not on PATH

`protoc @<argsfile>`
Read arguments from a file

`protoc --fatal_warnings <file>.proto`
Fail on warnings

`protoc --help`
List all flags

DESCRIPTORS

`protoc --descriptor_set_out=<f>.pb <file>.proto`
Write a FileDescriptorSet

DESCRIPTORS (cont.)

`protoc --descriptor_set_out=<f>.pb --include_imports <file>.proto`
Bundle imported files too
Tools such as grpcurl need imports included to load the set.

`protoc --descriptor_set_in=<f>.pb <Msg>`
Load schemas from a descriptor set

ENCODE AND DECODE

`protoc --decode_raw < <f>.bin`
Print wire data without a schema

`protoc -I. --decode=<pkg.Msg> <file>.proto < <f>.bin`
Decode binary to text

`protoc -I. --encode=<pkg.Msg> <file>.proto < <f>.txt > <f>.bin`
Encode text format to binary

`protoc --decode=<pkg.Msg> --descriptor_set_in=<f>.pb < <f>.bin`
Decode using a descriptor set

MESSAGES AND FIELDS

`syntax = "proto3";`
First line of a proto3 file

`package <name>;`
Namespace for the file's types

`import "<other>.proto";`
Use types from another file

`message <Name> { <type> <field> = <n>; }`
Declare a message

`optional <type> <field> = <n>;`
Field with presence tracking

`repeated <type> <field> = <n>;`
A list of values

`map<<key>, <value>> <field> = <n>;`
Key-value map field

`oneof <name> { <type> <a> = 1; <type> <b> = 2; }`
At most one of these fields set

`enum <Name> { <NAME>_UNSPECIFIED = 0; }`
Declare an enum; first value must be 0

`reserved 2, 15 to 20;`
Block retired field numbers

`reserved "<old_name>";`
Block retired field names

MESSAGES AND FIELDS (cont.)

`option go_package = "<path>";`
Set the Go import path

SCALAR TYPES

Common scalars

Type	Notes
`int32 / int64`	varint; poor for negatives
`sint32 / sint64`	zigzag; negatives are cheap
`uint32 / uint64`	unsigned varint
`fixed32 / fixed64`	fixed width, large values
`float / double`	IEEE 754
`bool`	true or false
`string`	UTF-8 or ASCII text
`bytes`	arbitrary byte data

Field numbers 1-15 take one byte on the wire; keep them for frequent fields. 19000-19999 are reserved.

Unset fields read as their zero value: empty string, 0, false, first enum value.

SERVICES

`service <Name> { rpc <Method>(<Req>) returns (<Resp>); }`
Declare a unary RPC

SERVICES (cont.)

`rpc <Method>(stream <Req>) returns (<Resp>);`
Client streaming RPC

`rpc <Method>(<Req>) returns (stream <Resp>);`
Server streaming RPC

`rpc <Method>(stream <Req>) returns (stream <Resp>);`
Bidirectional streaming RPC

protoc alone generates no RPC stubs; add a gRPC plugin (for example --grpc_out) for those.

COMPATIBILITY

Safe or not

Change	Safe?
Add a new field	yes
Remove a field and reserve it	yes
Rename a field	wire yes, JSON no
Reuse a field number	never
Change a field number	never
int32 / int64 / uint / bool	interchange may truncate
string and bytes	if valid UTF-8
Singular to repeated	strings, bytes, messages only
Move a field into an existing oneof	no