Protocol Buffers Schema compiler

.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

INVOKE PROTOC (cont.)

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

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

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

`protoc --help`
List all flags

DESCRIPTORS

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

`protoc --descriptor_set_out=<f>.pb --include_imports <file>.proto`
Bundle imported files too

ENCODE AND DECODE

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

ENCODE AND DECODE (cont.)

`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

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

MESSAGES AND FIELDS (cont.)

`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

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

SCALAR TYPES (cont.)

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

SERVICES

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

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	interchangea may truncate
string and bytes	if valid UTF-8
Singular to repeated	strings, bytes, messages only
Move a field into an existing oneof	no