

# Forum:[Hack Forums](#)

Thread: [GRPC Overview](#)

Sigma - May 10, 2021, 12:38 pm

Package

<https://github.com/lixiangyun/grpc-c.git>

Description

Given a protobuf definition (proto 3) provides routines to client routines (hello.grpc-c.[ch])  
service/server async/sync interface routines (hello.grpc-c.service.[ch])

Limitationsno protobuf defined validation layer

no protobuf defined HTTP REST

Build-time requirements (libgrpc-c)libgpr

libprotobuf

libgrpc

libprotobuf-c

libssl

libcrypto

libabsl\_\* (abseil-cpp)

libupb

libstdc++ (g++)

libcares

libm

See grpc-c/third-party and grpc/third-party

For MacOS:Brew install protobuf-c

Brew install gperftools  
Homebrew installs protobuf-c OpenSSL as keg only, because it does not want to conflict with the already non-Homebrew-installed OpenSSL, due to possible bad juju ensuing. Excerpt from "brew install protobuf-c" output:==> Caveats  
openssl@1.1 is keg-only, which means it was not symlinked into /usr/local, because this is an alternate version of another formula.

If you need to have openssl@1.1 first in your PATH run:  
echo 'export PATH="/usr/local/opt/openssl@1.1/bin:\$PATH"' >> ~/.bash\_profile

For compilers to find openssl@1.1 you may need to set:  
export LDFLAGS="-L/usr/local/opt/openssl@1.1/lib"

For pkg-config to find openssl@1.1 you may need to set:  
export PKG\_CONFIG\_PATH="/usr/local/opt/openssl@1.1/lib/pkgconfig"

Export LDFLAGS="-L/usr/local/opt/openssl@1.1/lib"

Export PKG\_CONFIG\_PATH="/usr/local/opt/openssl@1.1/lib/pkgconfig"

Run the build steps at <https://github.com/lixiangyun/grpc-c.git>

One other strange thing I had to do (if deps do not build) because la files with the wrong path were:  
sudo ln -s /usr/local/lib/ /lib

Generated code runtime library dependencies libgrp (allocation ie gpr\_alloc)

Service encode/decode, uses libgrpc (grpc\_slice, grpc\_slice\_unref, grpc\_byte\_buffer\_reader\_init, grpc\_byte\_buffer\_reader\_readall)

Service async/sync code, use libgrpc-c (grpc\_c\_client\_request\_sync, grpc\_c\_client\_request\_async, grpc\_c\_client\_request\_stream)

Service async callback, the caller must define appropriately named callback for async handling, use grpc-c routines for I/O.

Protobuf compilation

Code protoc -I/usr/local/include \

-I. \

-I\${GOPATH}/src \

-I\${GOPATH}/pkg/mod/github.com/sear!king/golang/tools/cmd/protoc-gen-go-tag\@v0.0.0-20201130132251-ce9944dd493c \

-I\${GOPATH}/pkg/mod/github.com/grpc-ecosystem/grpc-gateway/v2\@v2.0.1/third\_party/googleapi

```
s \
-I${GOPATH}/pkg/mod/github.com/envoyproxy/protoc-gen-validate/@v0.4.1 \
--grpc-gateway_opt=paths=source_relative --grpc-gateway_out=logtostderr=true:. \
--go-tag_out=paths=source_relative:. \
--validate_out="lang=go:." --validate_opt=paths=source_relative \
--openapi2_out=logtostderr=true:. \
--go-grpc_out=. --go-grpc_opt=paths=source_relative \
--grpc-c_out=. --plugin=protoc-gen-grpc-c=/usr/local/bin/protoc-gen-grpc-c \
pb/*.proto
Patch to grpc-c to enable compilation against newer grpc library
```

libgrpc-c was developed for a much older version of libgrpc ie 1.17.

To compile against 1.35 (latest), using the following patch will suffice:  
Code diff --git a/lib/include/grpc-c/grpc-c.h b/lib/include/grpc-c/grpc-c.h  
old mode 100644  
new mode 100755  
index 6045b71c..cfd7b0a  
--- a/lib/include/grpc-c/grpc-c.h  
+++ b/lib/include/grpc-c/grpc-c.h  
@@ -3,6 +3,7 @@  
#define \_\_GRPC\_C\_H\_\_

```
#include <string.h>
+#include <pthread.h>
#include <grpc/grpc.h>
#include <grpc/grpc_security.h>
#include <grpc/support/alloc.h>
diff --git a/lib/init.c b/lib/init.c
old mode 100644
new mode 100755
index 068117ff..5252ac22
--- a/lib/init.c
+++ b/lib/init.c
@@ -19,9 +19,11 @@ extern "C"{
*/
int grpc_c_init ()
{
#ifdef GRPC_LEGACY
gpr_allocation_functions func = {grpc_malloc,NULL,grpc_realloc,grpc_free};

gpr_set_allocation_functions(func);
#endif
```

Docker image (alpine) with grpc-c compiler available

Image has been pushed to aaronk-docker-registry.eng.sig:5000 (repository: alpine-grpc-c-compiler)

Deploy container (specifying bash), su - dev. The source code that was used to build the compiler etc is found in this directory, for reference.

All required build/runtime support binaries/libraries are pre-installed (/usr/local/lib or standard location).

## Creating a C client API for a new service

When writing a new service it is necessary to:

Create a service .proto file and include required imports (ie validation/tagging), generate C-code using grpc-c compiler

Create new package/library libplm-XXXclient, boilerplate(connection/shutdown) + generated code. Library needs to have run-time dependency on libgrpc-c.

Link GRPC client application with libplm-XXXclient to provide access to API routines.

## Required components

We want to create a dependency chain that matches the following.

client-app -> libplm-XXXclient -> libgrpc-c -> libgrpc

awarrenhttp (client-app)

GRPC client application sends requests and receives responses. In this case, all operations are read-only and constitute the web policy. Client applications need only worry about populating the protobuf request structures and reading the responses. No encoding/packing/unpacking required.

### libplm-webclient

Client API (C library) to the web config service as defined by the service protobuf definition + boilerplate code to perform initialization/destruction of the libgrpc-c context. As of this writing, the bulk of the code in this package is generated in a container (alpine-grpc-c-compiler) and the resultant files are manually copied into this package when needed the compiler will be integrated into the OMC build process. Technically this library isn't required but this approach ensures the code is reusable and simplifies the build process for the client application.

### libgrpc-c

Wrapper library provides access to libgrpc's I/O routines (asynchronous + synchronous) interfaces to send/receive GRPC messages using the protobuf format. Creates threads to handle I/O (async only?) via libgrpc. Library also supports server-side but that is not currently required.

### libgrpc

Core library provides I/O routines (ie using TLS) and performs (de)serialization of protobuf messages.