Compiling OpenSSL for Ubuntu and for the OmniFlash

Table of Contents

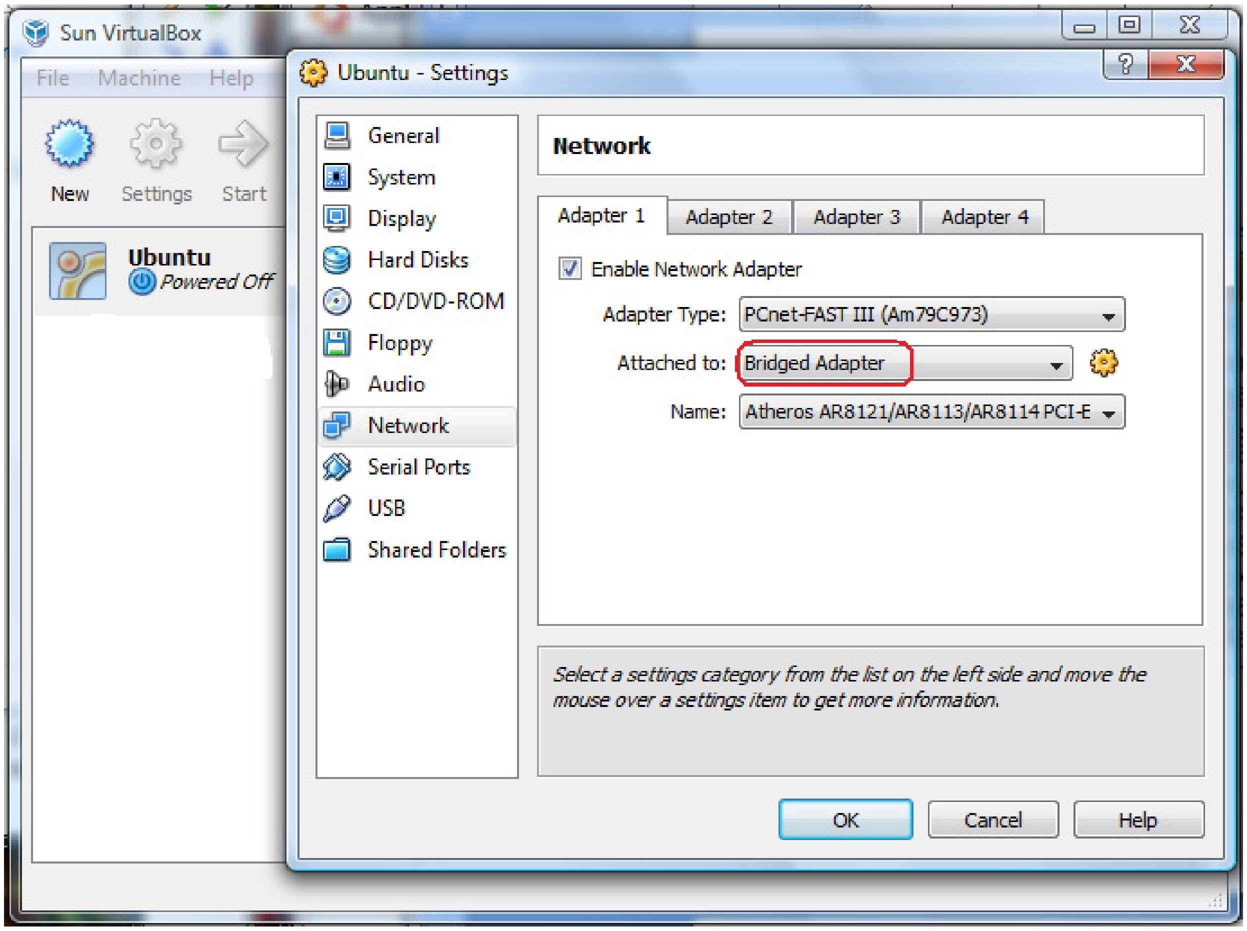
Introduction

This document describes how to Compile and install the OpenSSL libraries on Ubuntu Linux. It also describes how to cross compile it for the OmniFlash ARM processor.

Getting Started

Make sure you have a working Ubuntu Environment. Please refer to the document "Installing and configuring Ubuntu Linux.docx". Make sure you have the cross compiler installed for the ARM processor. Please see the document "Configuring Ubuntu to Code for the OmniFlash or OmniEP.docx".

Before we begin, if you are running Ubuntu in a Virtual machine, in order to communicate with the OmniFlash over the network, we must be on the same physical network.



Network

Enable Network Adapter [✓]

Adapter Type: PCnet-FAST III (Am79C973)

Attached to: Bridged Adapter

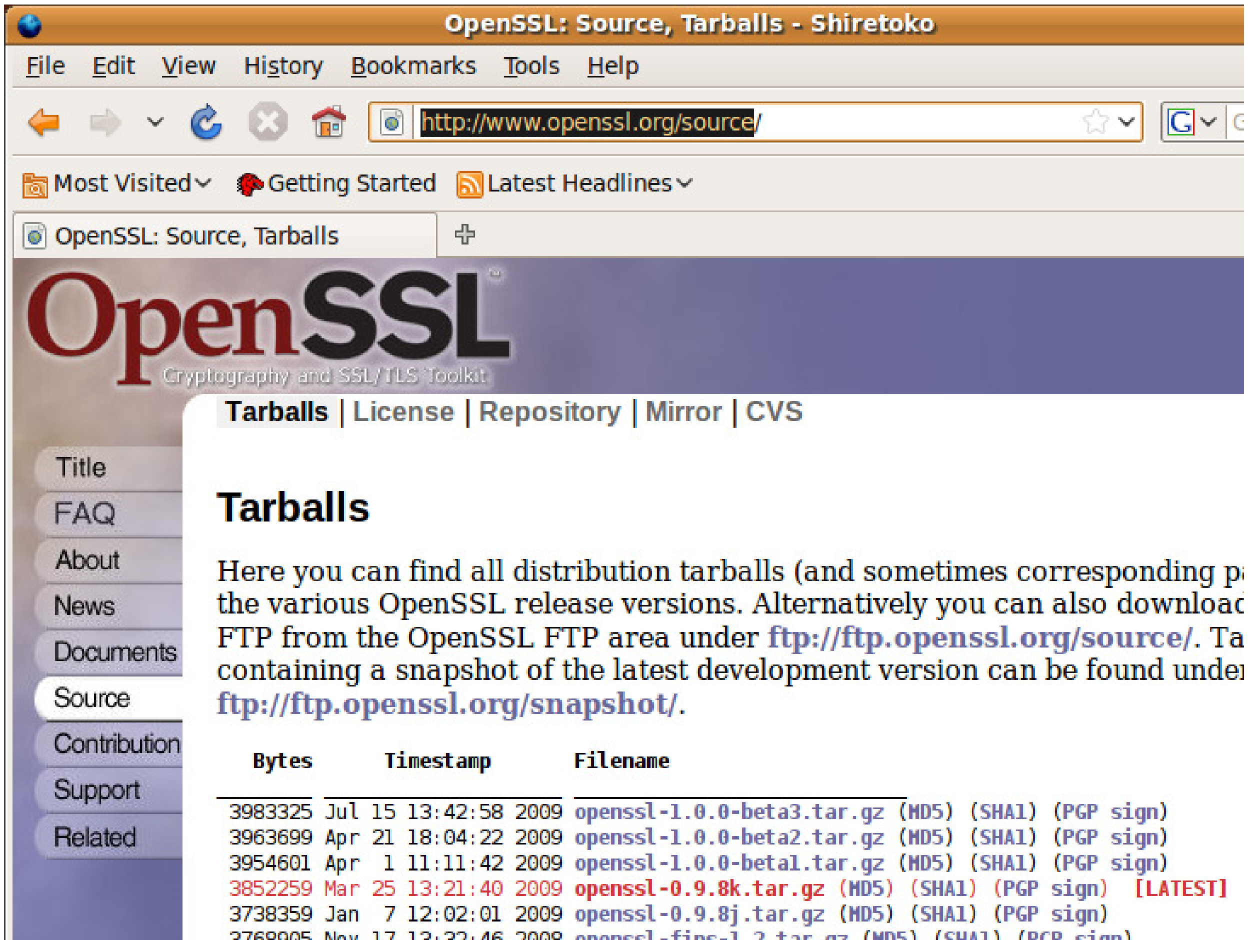
Name: Atheros AR8121/AR8113/AR8114 PCI-E

Select a settings category from the list on the left side and move the mouse over a settings item to get more information.

Make sure your Virtual Machine's network type is Bridged Adapter or we won't be able to communicate.

Getting the OpenSSL Source Code

You can download the source code from: http://www.openssl.org/source and clicking on the Latest.



OpenSSL: Source, Tarballs - Shiretoko
File Edit View History Bookmarks Tools Help
http://www.openssl.org/source/
Most Visited
Getting Started
Latest Headlines
OpenSSL: Source, Tarballs
OpenSSL
Cryptography and SSL/TLS Toolkit
Tarballs | License | Repository | Mirror | CVS
Title
FAQ
About
News
Documents
Source
Contribution
Support
Related

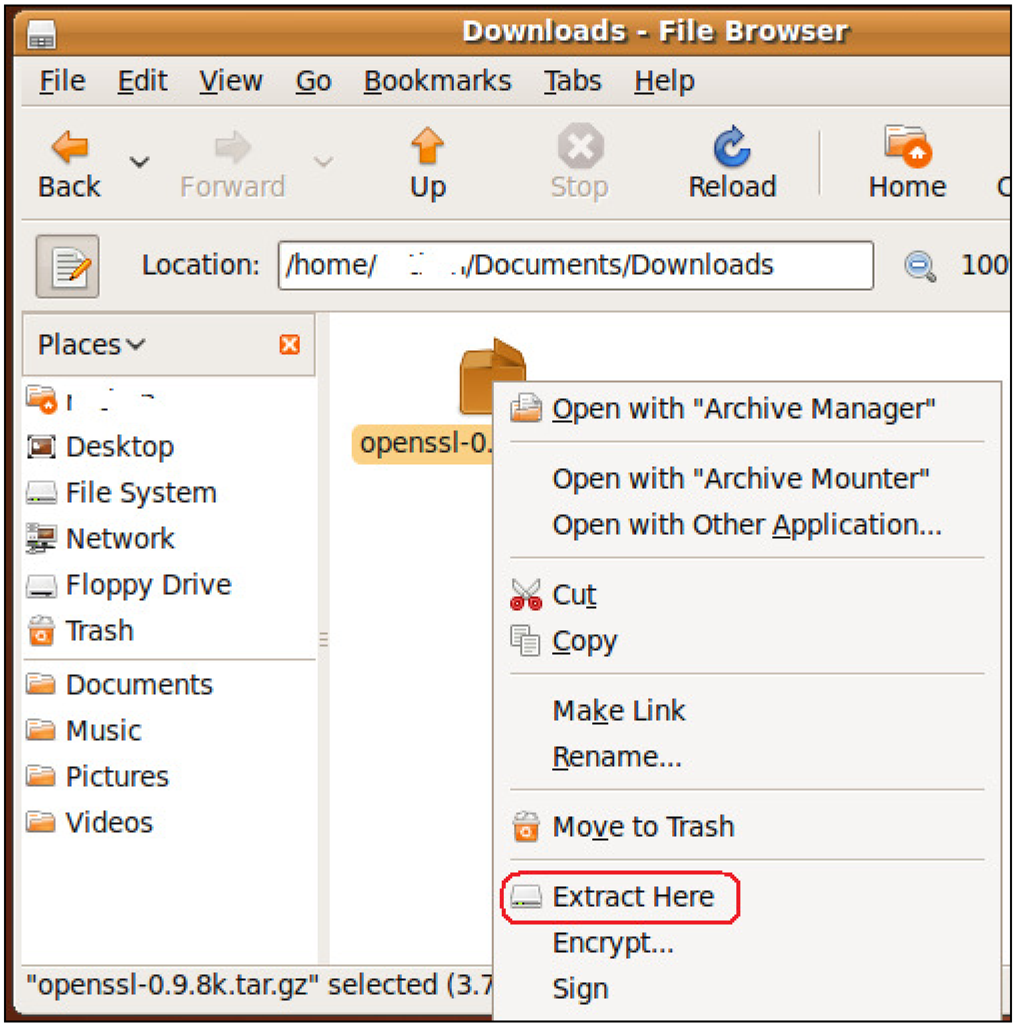
Tarballs

Here you can find all distribution tarballs (and sometimes corresponding p
the various OpenSSL release versions. Alternatively you can also downloac
FTP from the OpenSSL FTP area under ftp://ftp.openssl.org/source/. Ta
containing a snapshot of the latest development version can be found unde
ftp://ftp.openssl.org/snapshot/.

```
  Bytes      Timestamp         Filename
3983325 Jul 15 13:42:58 2009 openssl-1.0.0-beta3.tar.gz (MD5) (SHA1) (PGP sign)
3963699 Apr 21 18:04:22 2009 openssl-1.0.0-beta2.tar.gz (MD5) (SHA1) (PGP sign)
3954601 Apr  1 11:11:42 2009 openssl-1.0.0-beta1.tar.gz (MD5) (SHA1) (PGP sign)
3852259 Mar 25 13:21:40 2009 openssl-0.9.8k.tar.gz (MD5) (SHA1) (PGP sign)  [LATEST]
3738359 Jan  7 12:02:01 2009 openssl-0.9.8j.tar.gz (MD5) (SHA1) (PGP sign)
```

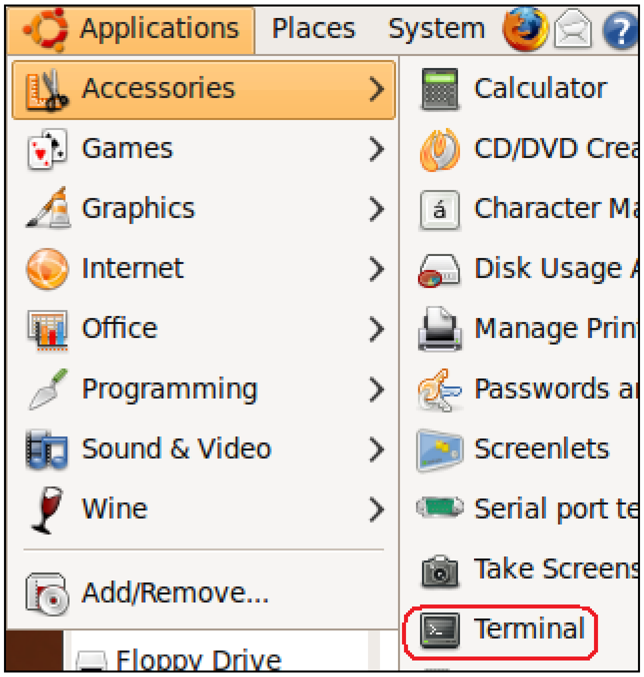
Save the file to your system somewhere.

Extracting the Source

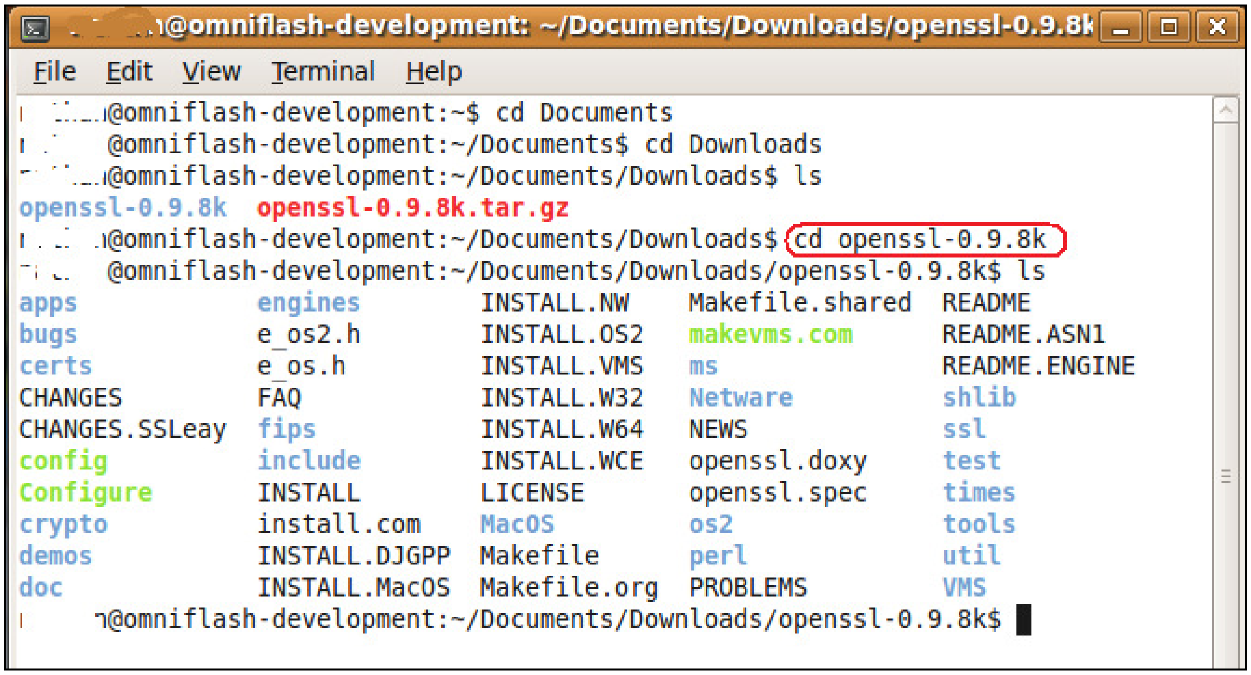


Now right-click on the file and select Extract here.

Compiling the Source

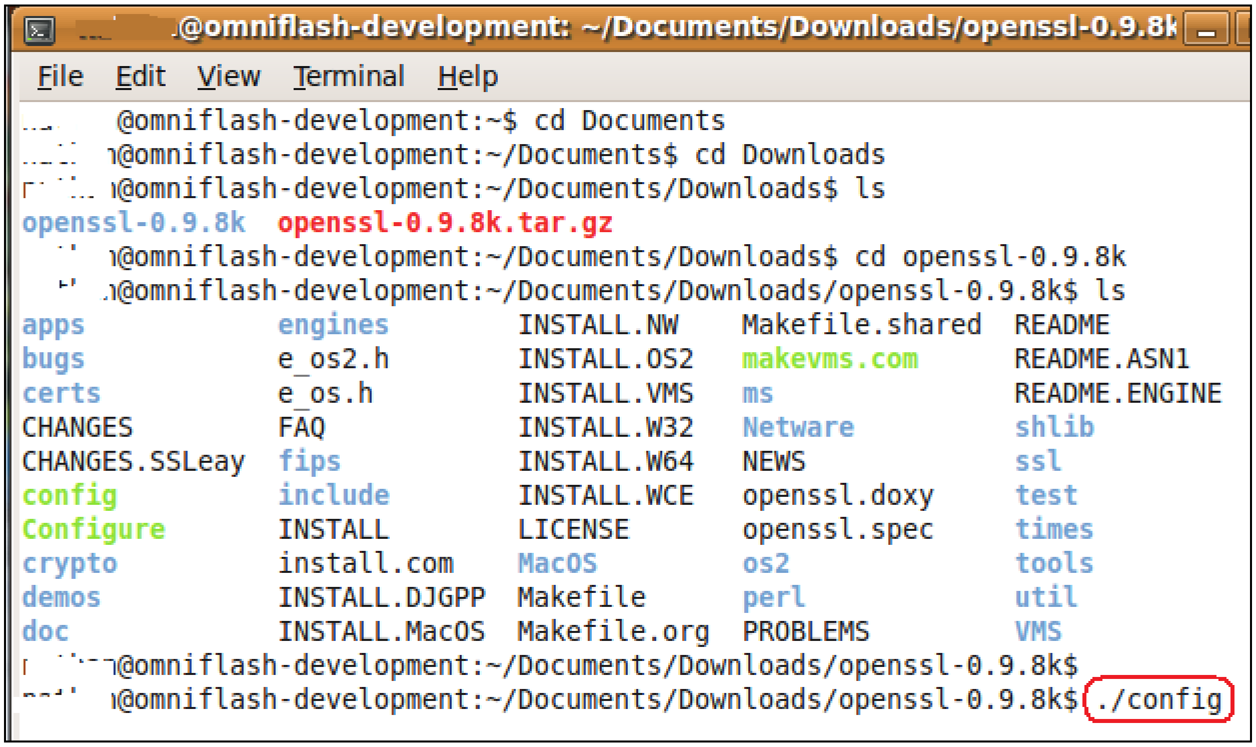


Open a terminal so we can build the source code.


n@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File
Edit
View
Terminal
Help

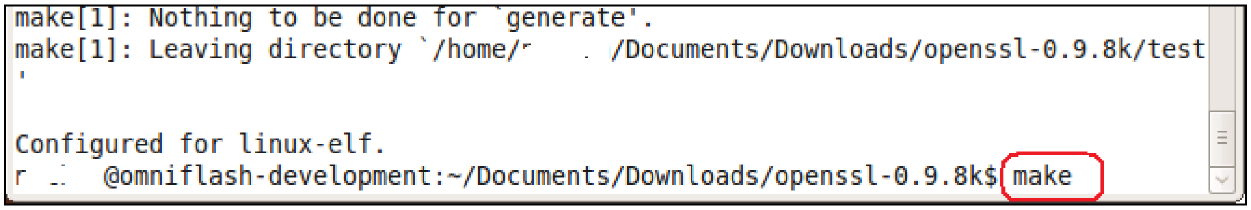
```
@omniflash-development:~$ cd Documents
@omniflash-development:~/Documents$ cd Downloads
@omniflash-development:~/Documents/Downloads$ ls
openssl-0.9.8k  openssl-0.9.8k.tar.gz
@omniflash-development:~/Documents/Downloads$ cd openssl-0.9.8k
@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ ls
apps            engines        INSTALL.NW    Makefile.shared  README
bugs            e_os2.h        INSTALL.OS2   makevms.com      README.ASN1
certs           e_os.h         INSTALL.VMS   ms               README.ENGINE
CHANGES         FAQ            INSTALL.W32   Netware          shlib
CHANGES.SSLeay  fips           INSTALL.W64   NEWS             ssl
config          include        INSTALL.WCE   openssl.doxy     test
Configure       INSTALL        LICENSE       openssl.spec     times
crypto          install.com    MacOS         os2              tools
demos           INSTALL.DJGPP  Makefile      perl             util
doc             INSTALL.MacOS  Makefile.org  PROBLEMS         VMS
n@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
```

cd into the folder we just extracted.


[redacted]@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File Edit View Terminal Help

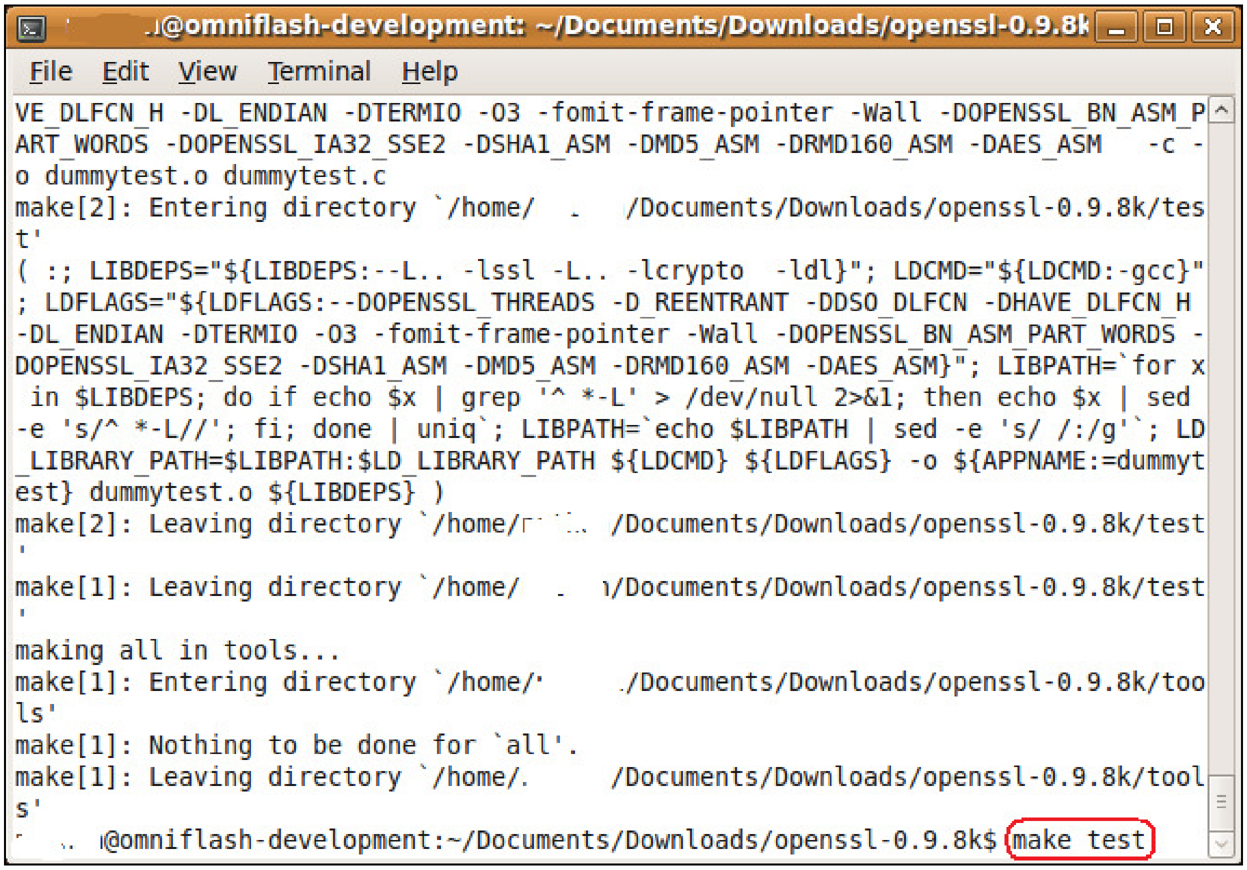
```
[redacted]@omniflash-development:~$ cd Documents
[redacted]@omniflash-development:~/Documents$ cd Downloads
[redacted]@omniflash-development:~/Documents/Downloads$ ls
openssl-0.9.8k  openssl-0.9.8k.tar.gz
[redacted]@omniflash-development:~/Documents/Downloads$ cd openssl-0.9.8k
[redacted]@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ ls
apps            engines         INSTALL.NW     Makefile.shared  README
bugs            e_os2.h         INSTALL.OS2    makevms.com      README.ASN1
certs           e_os.h          INSTALL.VMS    ms               README.ENGINE
CHANGES         FAQ             INSTALL.W32    Netware          shlib
CHANGES.SSLeay  fips            INSTALL.W64    NEWS             ssl
config          include         INSTALL.WCE    openssl.doxy     test
Configure       INSTALL         LICENSE        openssl.spec     times
crypto          install.com     MacOS          os2              tools
demos           INSTALL.DJGPP   Makefile       perl             util
doc             INSTALL.MacOS   Makefile.org   PROBLEMS         VMS
[redacted]@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
[redacted]@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ ./config
```

Type **./config** and press enter to configure the makefile.



```
make[1]: Nothing to be done for `generate'.
make[1]: Leaving directory `/home/[redacted]/Documents/Downloads/openssl-0.9.8k/test
'

Configured for linux-elf.
[redacted]@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ make
```

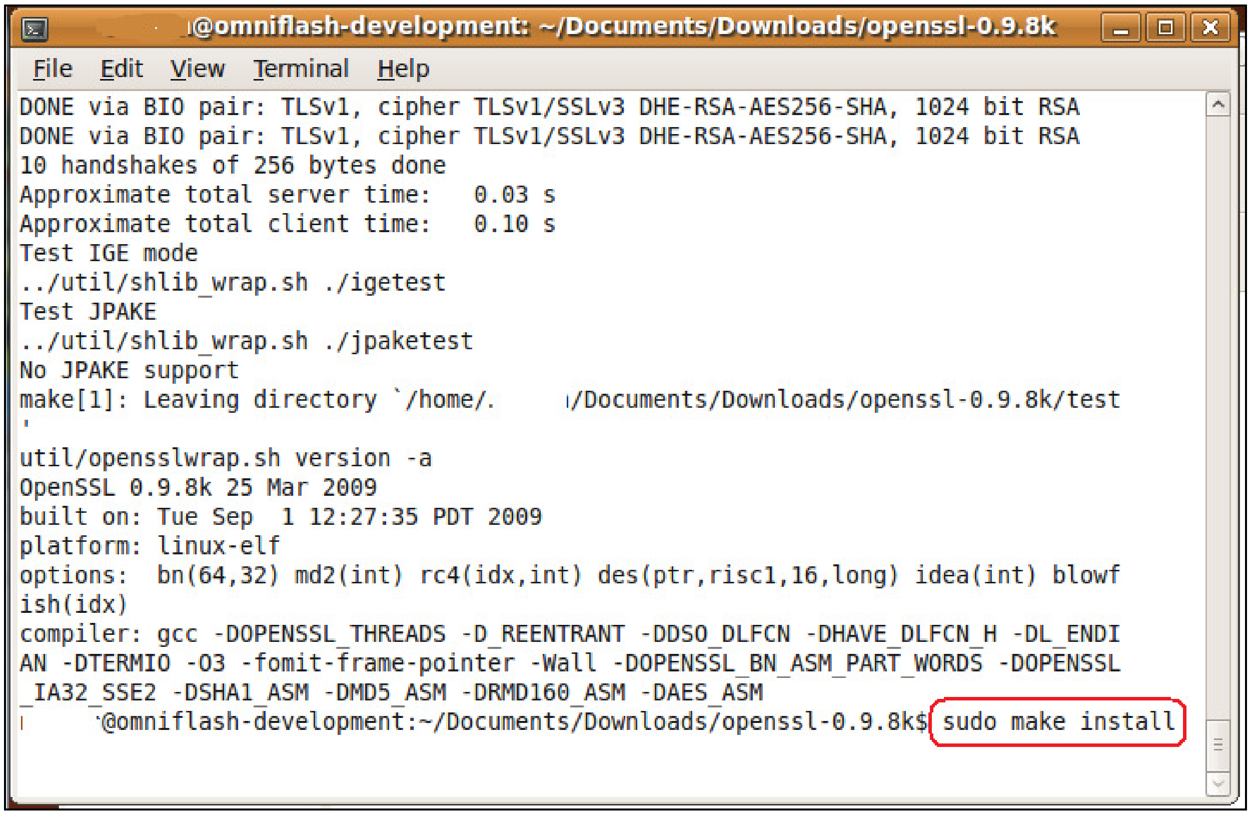
Now type **make** to build the libraries.


@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File Edit View Terminal Help

```
VE_DLFCN_H -DL_ENDIAN -DTERMIO -O3 -fomit-frame-pointer -Wall -DOPENSSL_BN_ASM_P
ART_WORDS -DOPENSSL_IA32_SSE2 -DSHA1_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM   -c -
o dummytest.o dummytest.c
make[2]: Entering directory `/home/       /Documents/Downloads/openssl-0.9.8k/tes
t'
( :; LIBDEPS="${LIBDEPS:--L.. -lssl -L.. -lcrypto  -ldl}"; LDCMD="${LDCMD:-gcc}"
; LDFLAGS="${LDFLAGS:--DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H
-DL_ENDIAN -DTERMIO -O3 -fomit-frame-pointer -Wall -DOPENSSL_BN_ASM_PART_WORDS -
DOPENSSL_IA32_SSE2 -DSHA1_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM}"; LIBPATH=`for x
 in $LIBDEPS; do if echo $x | grep '^ *-L' > /dev/null 2>&1; then echo $x | sed
-e 's/^ *-L//'; fi; done | uniq`; LIBPATH=`echo $LIBPATH | sed -e 's/ /:/g'`; LD
_LIBRARY_PATH=$LIBPATH:$LD_LIBRARY_PATH ${LDCMD} ${LDFLAGS} -o ${APPNAME:=dummyt
est} dummytest.o ${LIBDEPS} )
make[2]: Leaving directory `/home/       /Documents/Downloads/openssl-0.9.8k/test
'
make[1]: Leaving directory `/home/       /Documents/Downloads/openssl-0.9.8k/test
'
making all in tools...
make[1]: Entering directory `/home/       /Documents/Downloads/openssl-0.9.8k/too
ls'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/       /Documents/Downloads/openssl-0.9.8k/tool
s'
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ make test
```

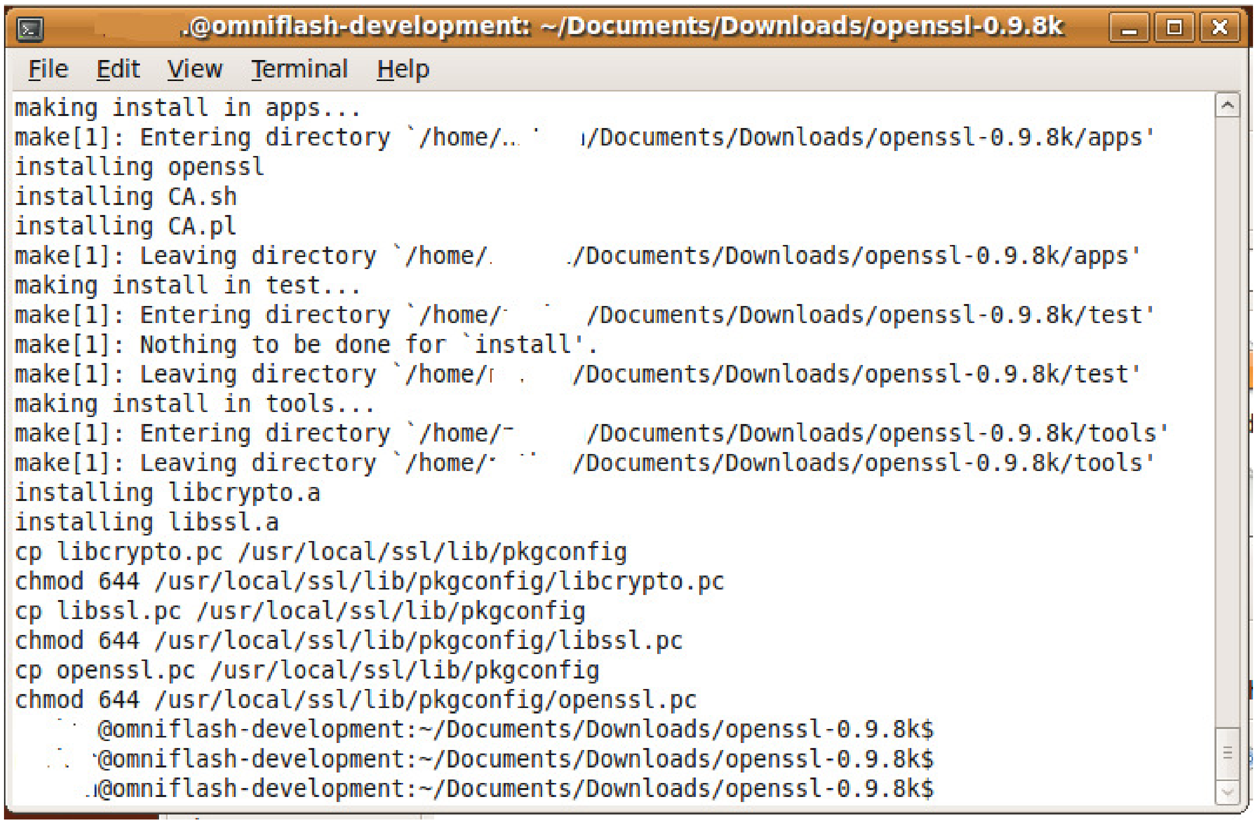
Verify that there were no errors. Now type **make test**.

Installing the libraries


@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File Edit View Terminal Help

```
DONE via BIO pair: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-AES256-SHA, 1024 bit RSA
DONE via BIO pair: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-AES256-SHA, 1024 bit RSA
10 handshakes of 256 bytes done
Approximate total server time:   0.03 s
Approximate total client time:   0.10 s
Test IGE mode
../util/shlib_wrap.sh ./igetest
Test JPAKE
../util/shlib_wrap.sh ./jpaketest
No JPAKE support
make[1]: Leaving directory `/home/.     /Documents/Downloads/openssl-0.9.8k/test
'
util/opensslwrap.sh version -a
OpenSSL 0.9.8k 25 Mar 2009
built on: Tue Sep  1 12:27:35 PDT 2009
platform: linux-elf
options:  bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) idea(int) blowf
ish(idx)
compiler: gcc -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDI
AN -DTERMIO -O3 -fomit-frame-pointer -Wall -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL
_IA32_SSE2 -DSHA1_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ sudo make install
```

And finally, we need to install the libraries where we can get to them. We do this by typing **sudo make install** . Enter your password when prompted.

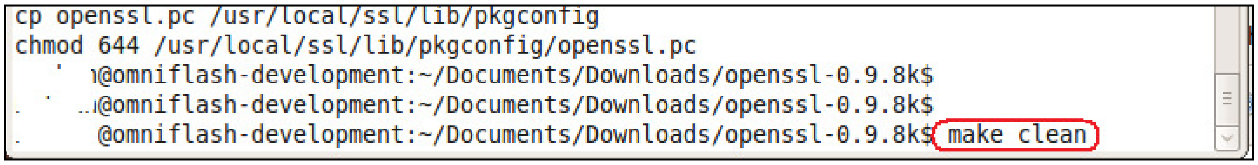

@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File Edit View Terminal Help

```
making install in apps...
make[1]: Entering directory `/home/      /Documents/Downloads/openssl-0.9.8k/apps'
installing openssl
installing CA.sh
installing CA.pl
make[1]: Leaving directory `/home/      /Documents/Downloads/openssl-0.9.8k/apps'
making install in test...
make[1]: Entering directory `/home/      /Documents/Downloads/openssl-0.9.8k/test'
make[1]: Nothing to be done for `install'.
make[1]: Leaving directory `/home/      /Documents/Downloads/openssl-0.9.8k/test'
making install in tools...
make[1]: Entering directory `/home/      /Documents/Downloads/openssl-0.9.8k/tools'
make[1]: Leaving directory `/home/      /Documents/Downloads/openssl-0.9.8k/tools'
installing libcrypto.a
installing libssl.a
cp libcrypto.pc /usr/local/ssl/lib/pkgconfig
chmod 644 /usr/local/ssl/lib/pkgconfig/libcrypto.pc
cp libssl.pc /usr/local/ssl/lib/pkgconfig
chmod 644 /usr/local/ssl/lib/pkgconfig/libssl.pc
cp openssl.pc /usr/local/ssl/lib/pkgconfig
chmod 644 /usr/local/ssl/lib/pkgconfig/openssl.pc
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
```

You now have OpenSSL installed for Ubuntu Linux.

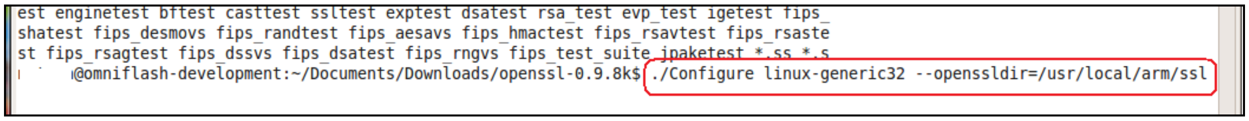
Compiling OpenSSL For the OmniFlash ARM

After we are done building the source for Ubuntu Linux, we need to recompile it for the ARM processor. Open up a terminal window (if you closed the above window) to the area where the source code is extracted to.



```
cp openssl.pc /usr/local/ssl/lib/pkgconfig
chmod 644 /usr/local/ssl/lib/pkgconfig/openssl.pc
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ make clean
```

Now type **make clean** to clean up our area.

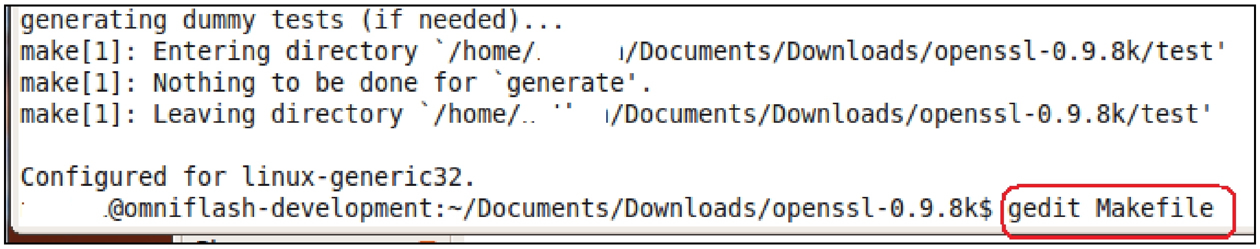


```
est enginetest bftest casttest ssltest exptest dsatest rsa_test evp_test igetest fips_
shatest fips_desmovs fips_randtest fips_aesavs fips_hmactest fips_rsavtest fips_rsaste
st fips_rsagtest fips_dssvs fips_dsatest fips_rngvs fips_test_suite jpaketest *.ss *.s
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ ./Configure linux-generic32 --openssldir=/usr/local/arm/ssl
```

Next configure the makefile for generic linux and specify where you want your output to end up at.

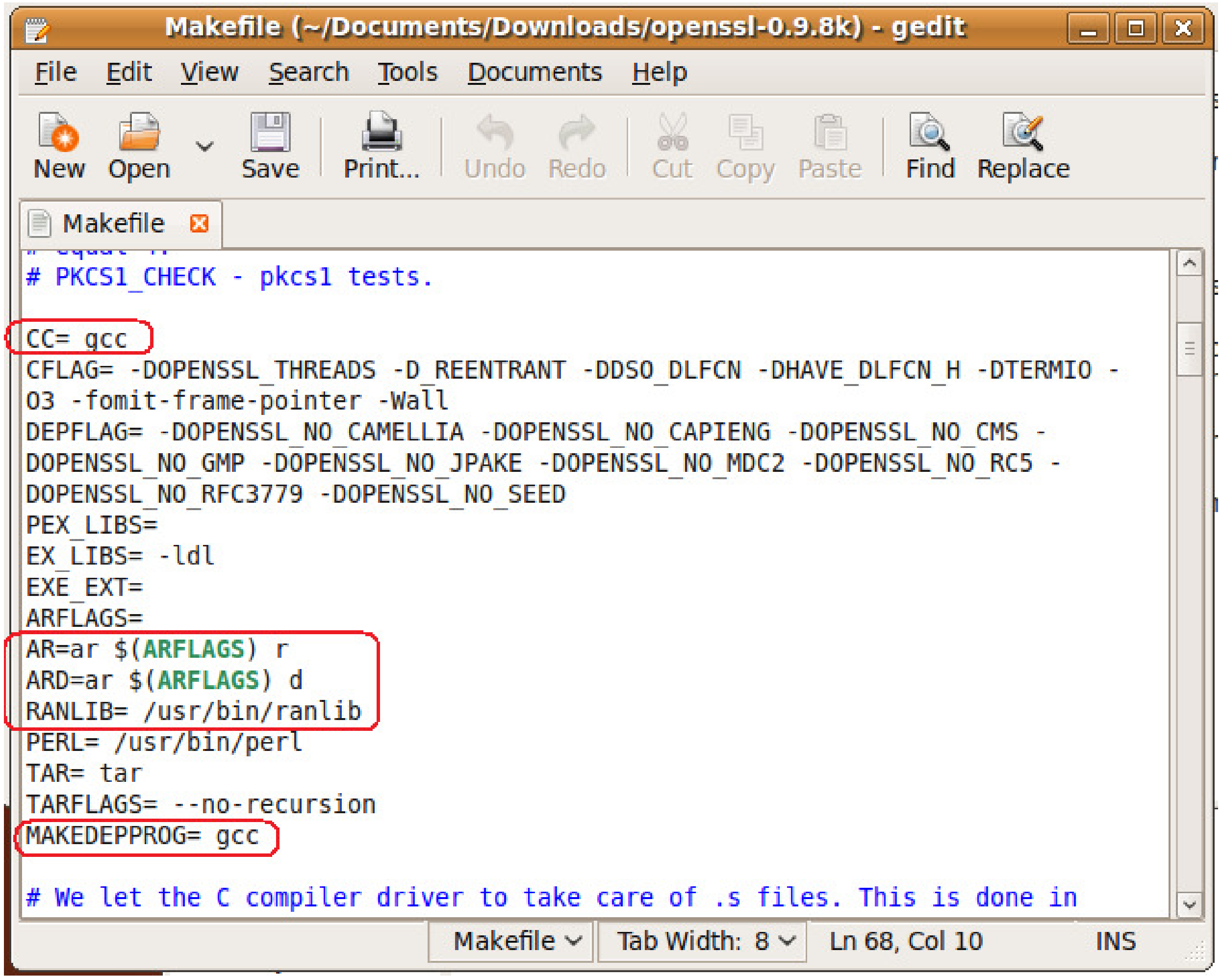
Type **./Configure linux-generic32 --openssldir=/usr/local/arm/ssl**

Editing the Makefile



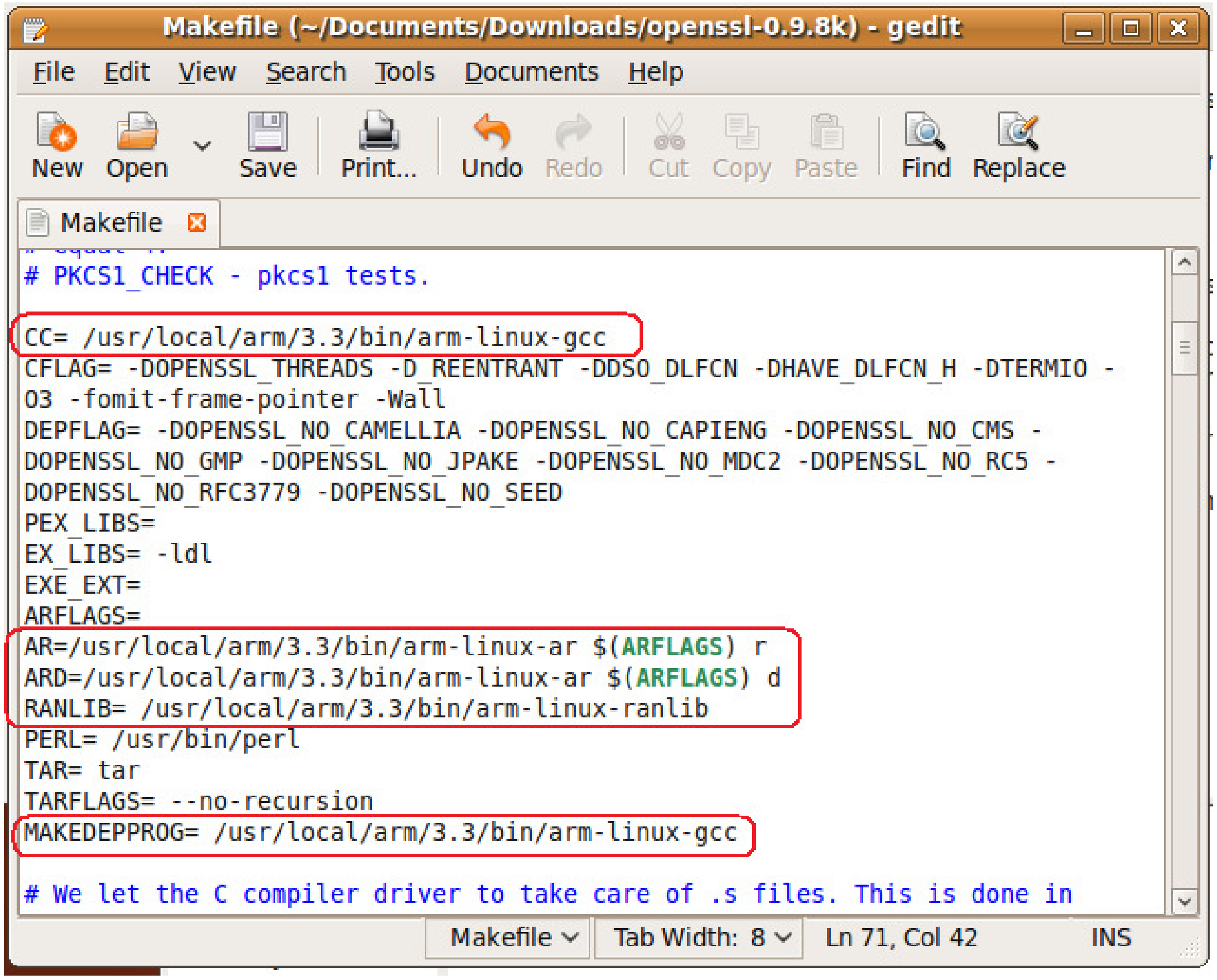
```
generating dummy tests (if needed)...
make[1]: Entering directory `/home/.      ı/Documents/Downloads/openssl-0.9.8k/test'
make[1]: Nothing to be done for `generate'.
make[1]: Leaving directory `/home/.. ''  ı/Documents/Downloads/openssl-0.9.8k/test'

Configured for linux-generic32.
      .@omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ gedit Makefile
```

Now we need to edit the Makefile and adjust the compiler settings to use the ARM compiler. Type **gedit Makefile** and press enter.


Makefile (~/Documents/Downloads/openssl-0.9.8k) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
Makefile
Makefile Tab Width: 8 Ln 68, Col 10 INS

```
# PKCS1_CHECK - pkcs1 tests.

CC= gcc
CFLAG= -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIO -
O3 -fomit-frame-pointer -Wall
DEPFLAG= -DOPENSSL_NO_CAMELLIA -DOPENSSL_NO_CAPIENG -DOPENSSL_NO_CMS -
DOPENSSL_NO_GMP -DOPENSSL_NO_JPAKE -DOPENSSL_NO_MDC2 -DOPENSSL_NO_RC5 -
DOPENSSL_NO_RFC3779 -DOPENSSL_NO_SEED
PEX_LIBS=
EX_LIBS= -ldl
EXE_EXT=
ARFLAGS=
AR=ar $(ARFLAGS) r
ARD=ar $(ARFLAGS) d
RANLIB= /usr/bin/ranlib
PERL= /usr/bin/perl
TAR= tar
TARFLAGS= --no-recursion
MAKEDEPPROG= gcc

# We let the C compiler driver to take care of .s files. This is done in
```

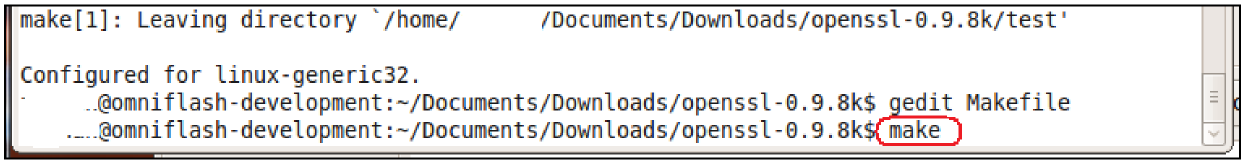
Scroll down to the lines shown above. We need to alter these to point to the ARM version.


Makefile (~/Documents/Downloads/openssl-0.9.8k) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
Makefile
Makefile Tab Width: 8 Ln 71, Col 42 INS

```
# PKCS1_CHECK - pkcs1 tests.

CC= /usr/local/arm/3.3/bin/arm-linux-gcc
CFLAG= -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIO -
O3 -fomit-frame-pointer -Wall
DEPFLAG= -DOPENSSL_NO_CAMELLIA -DOPENSSL_NO_CAPIENG -DOPENSSL_NO_CMS -
DOPENSSL_NO_GMP -DOPENSSL_NO_JPAKE -DOPENSSL_NO_MDC2 -DOPENSSL_NO_RC5 -
DOPENSSL_NO_RFC3779 -DOPENSSL_NO_SEED
PEX_LIBS=
EX_LIBS= -ldl
EXE_EXT=
ARFLAGS=
AR=/usr/local/arm/3.3/bin/arm-linux-ar $(ARFLAGS) r
ARD=/usr/local/arm/3.3/bin/arm-linux-ar $(ARFLAGS) d
RANLIB= /usr/local/arm/3.3/bin/arm-linux-ranlib
PERL= /usr/bin/perl
TAR= tar
TARFLAGS= --no-recursion
MAKEDEPPROG= /usr/local/arm/3.3/bin/arm-linux-gcc

# We let the C compiler driver to take care of .s files. This is done in
```

Change the path and compiler names to match those above. The prefix is where we installed the compiler to earlier. If you are using another ARM compiler, change the paths and executable names to match the one you are using.
Save the file and exit.

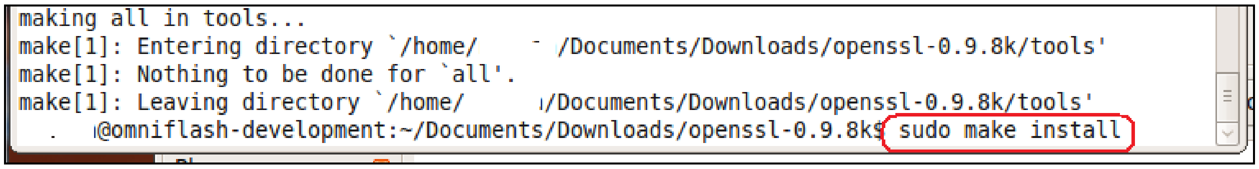
Building the source



```
make[1]: Leaving directory `/home/      /Documents/Downloads/openssl-0.9.8k/test'

Configured for linux-generic32.
    @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ gedit Makefile
    @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ make
```

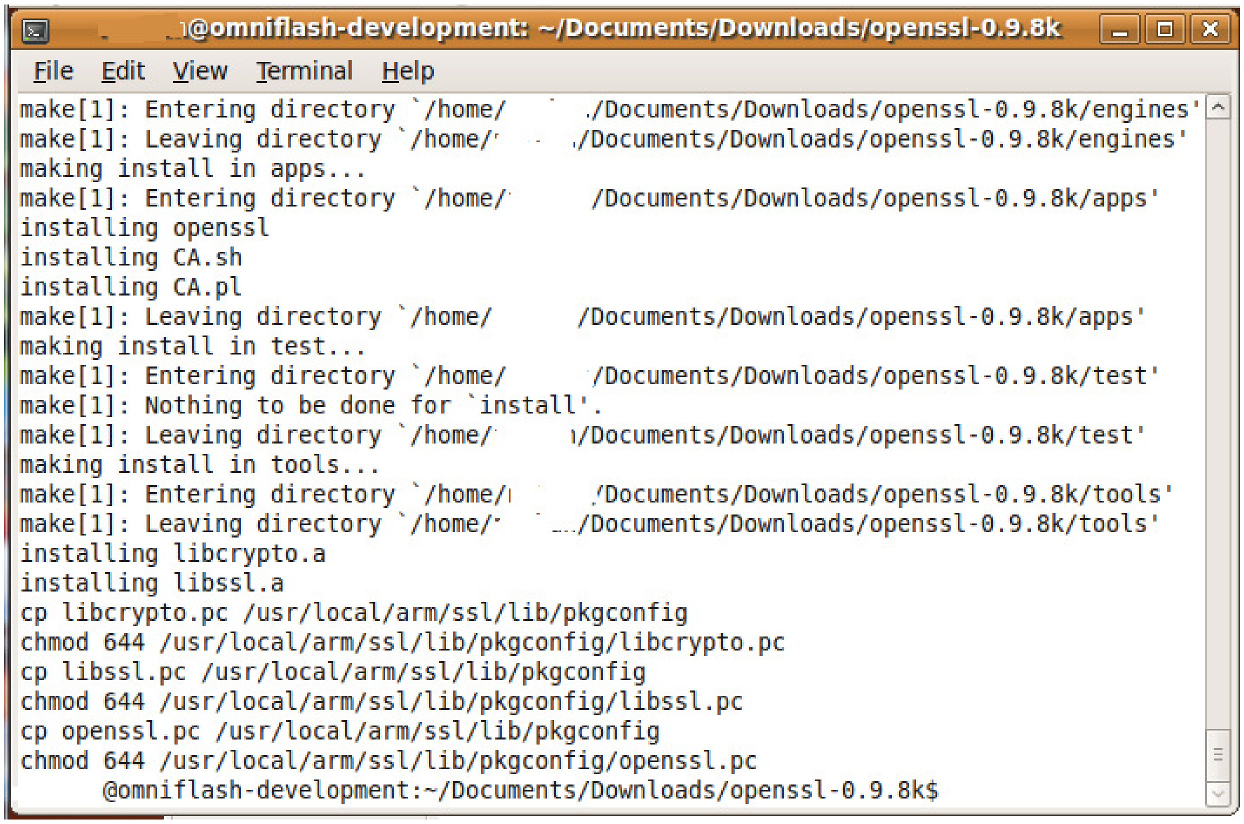
Now type **make** to start the build process.

Installing the ARM version



```
making all in tools...
make[1]: Entering directory `/home/    /Documents/Downloads/openssl-0.9.8k/tools'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/    /Documents/Downloads/openssl-0.9.8k/tools'
    @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$ sudo make install
```

Type **sudo make install** to install it to the directory we set in the configuration above. Type your password if asked.



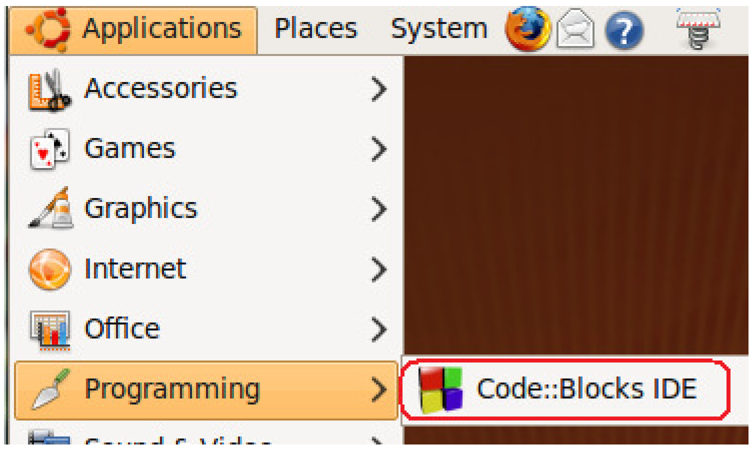
```
make[1]: Entering directory `/home/    /Documents/Downloads/openssl-0.9.8k/engines'
make[1]: Leaving directory `/home/    /Documents/Downloads/openssl-0.9.8k/engines'
making install in apps...
make[1]: Entering directory `/home/    /Documents/Downloads/openssl-0.9.8k/apps'
installing openssl
installing CA.sh
installing CA.pl
make[1]: Leaving directory `/home/    /Documents/Downloads/openssl-0.9.8k/apps'
making install in test...
make[1]: Entering directory `/home/    /Documents/Downloads/openssl-0.9.8k/test'
make[1]: Nothing to be done for `install'.
make[1]: Leaving directory `/home/    /Documents/Downloads/openssl-0.9.8k/test'
making install in tools...
make[1]: Entering directory `/home/    /Documents/Downloads/openssl-0.9.8k/tools'
make[1]: Leaving directory `/home/    /Documents/Downloads/openssl-0.9.8k/tools'
installing libcrypto.a
installing libssl.a
cp libcrypto.pc /usr/local/arm/ssl/lib/pkgconfig
chmod 644 /usr/local/arm/ssl/lib/pkgconfig/libcrypto.pc
cp libssl.pc /usr/local/arm/ssl/lib/pkgconfig
chmod 644 /usr/local/arm/ssl/lib/pkgconfig/libssl.pc
cp openssl.pc /usr/local/arm/ssl/lib/pkgconfig
chmod 644 /usr/local/arm/ssl/lib/pkgconfig/openssl.pc
      @omniflash-development:~/Documents/Downloads/openssl-0.9.8k$
```

@omniflash-development: ~/Documents/Downloads/openssl-0.9.8k
File Edit View Terminal Help

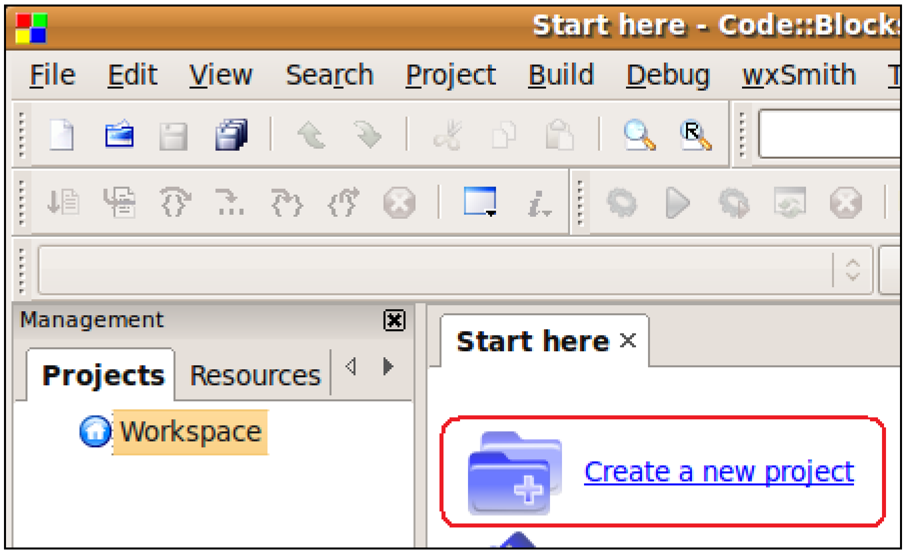
We should end up with no errors and OpenSSL configured and installed for us.

Creating a Test Program for Linux (OpenSSL Client)

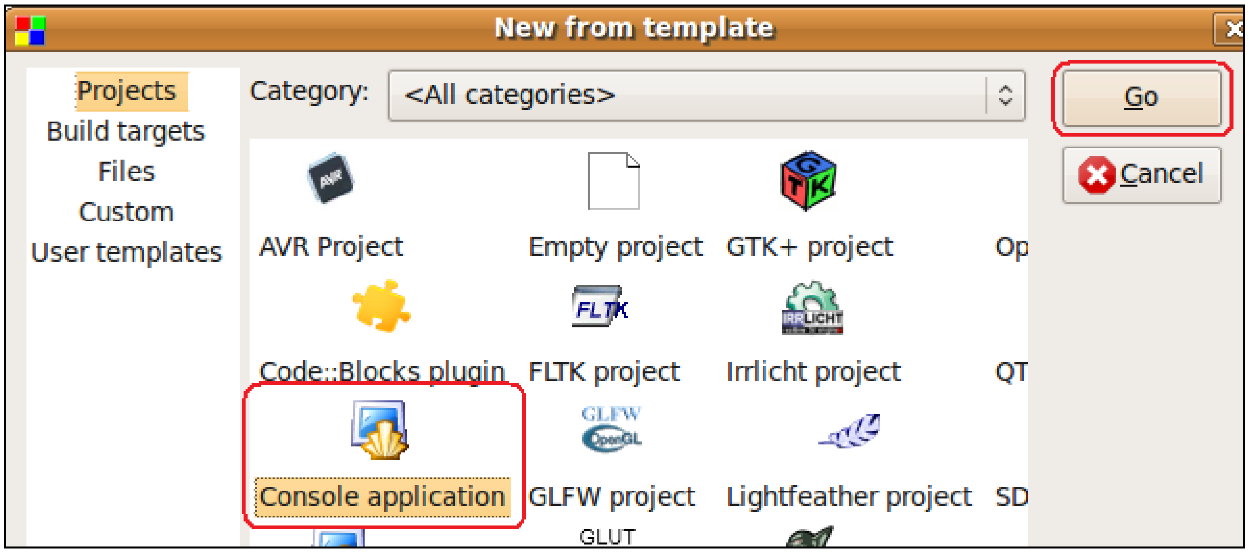
Now let's take one of the sample programs compile it.



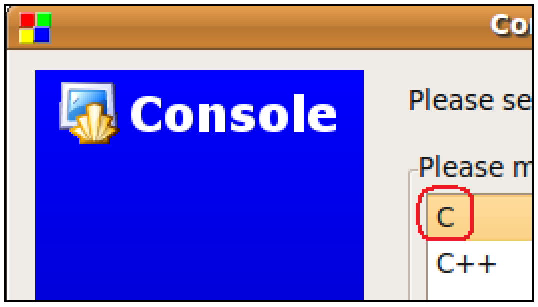
Launch CodeBlocks



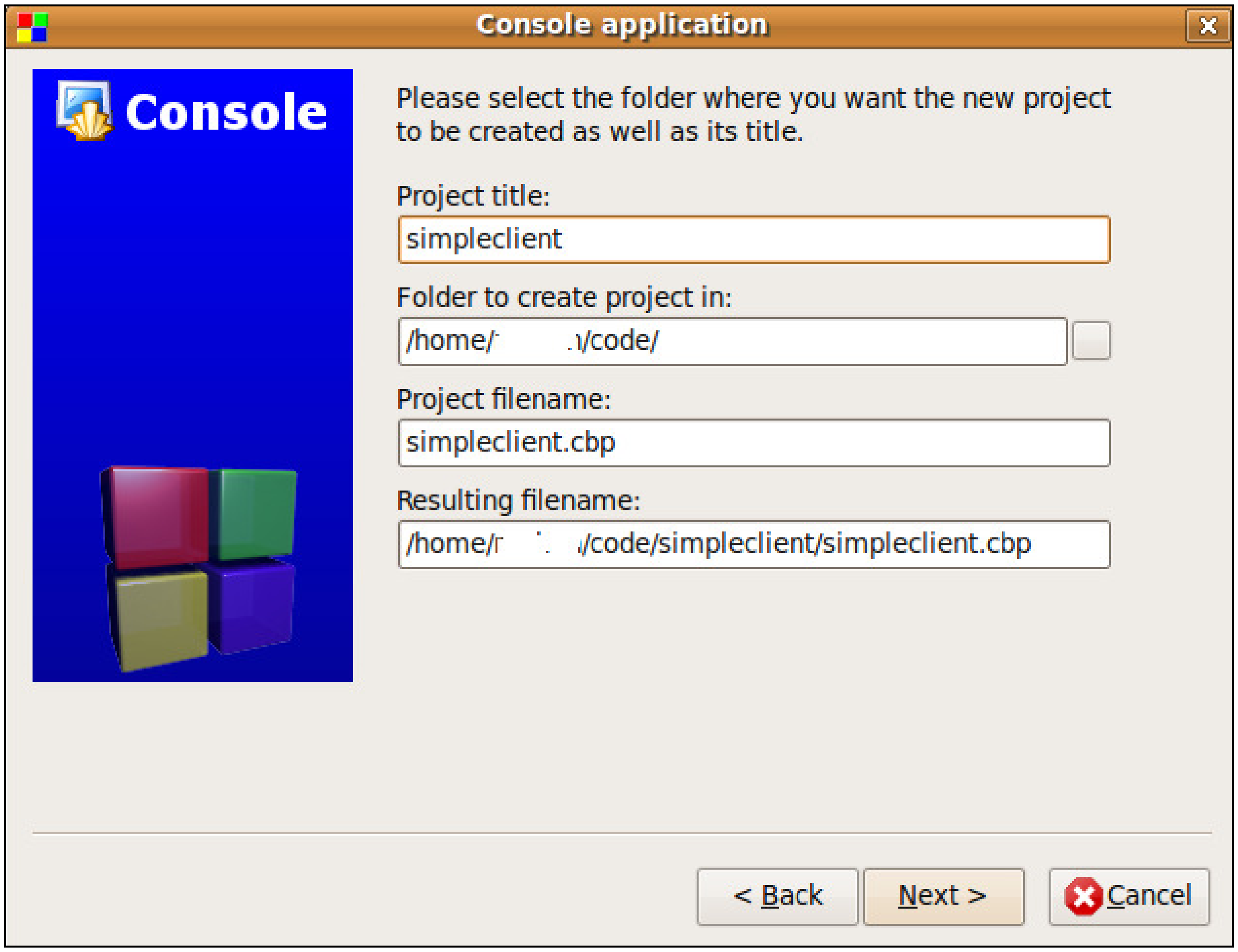
Click Create a new project.



Click Console application and Go.



Click C style and click Next.



Please select the folder where you want the new project to be created as well as its title.

Project title: simpleclient

Folder to create project in: /home/[redacted]/code/

Project filename: simpleclient.cbp

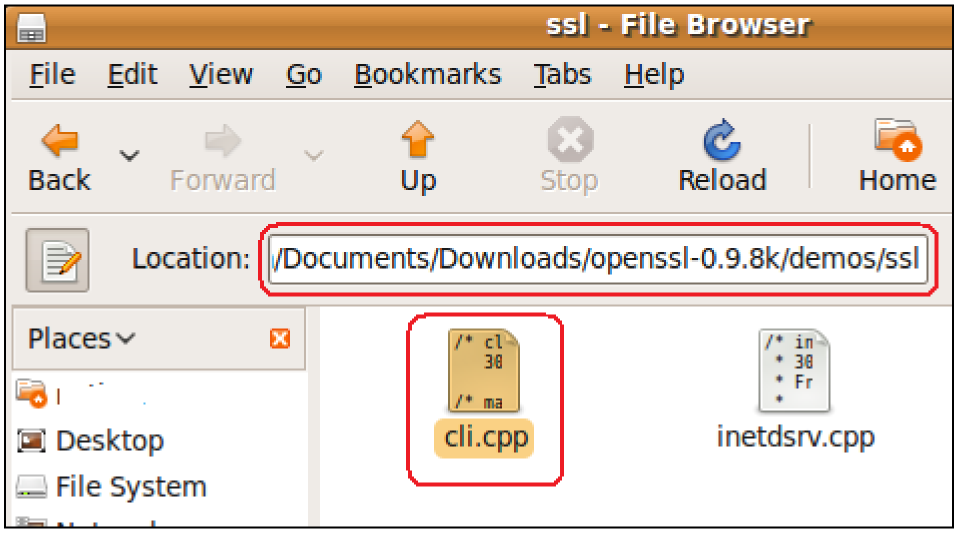
Resulting filename: /home/[redacted]/code/simpleclient/simpleclient.cbp

Type in a project a project name and choose a folder to create the project in.

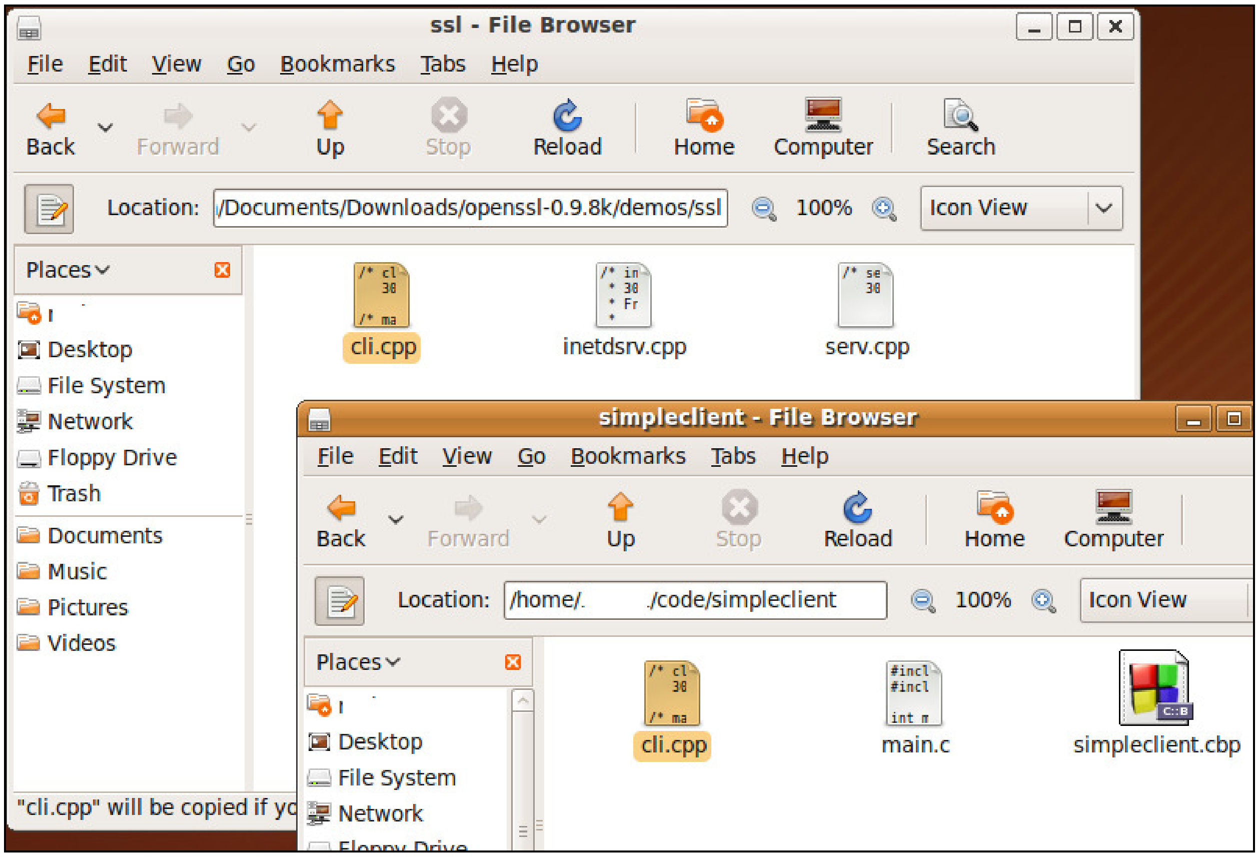
Click Next until you get to the Finish button and click Finish.

Getting a sample client source code file

Now we need to get a copy of the client sample SSL program. Minimize CodeBlocks and open up a file explorer.


ssl - File Browser
File Edit View Go Bookmarks Tabs Help
Back Forward Up Stop Reload Home
Location: /Documents/Downloads/openssl-0.9.8k/demos/ssl
Places
Desktop
File System
cli.cpp
inetdsrv.cpp

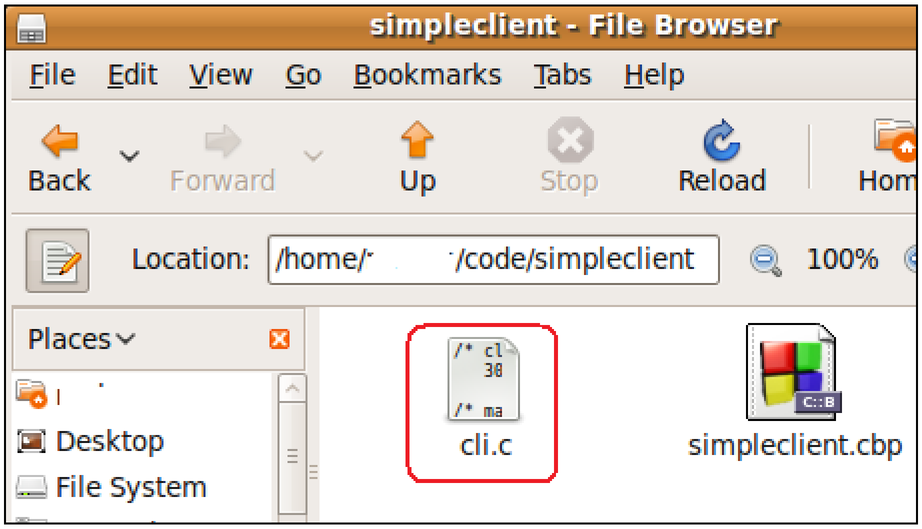
Navigate to the folder where you extracted the source code to. The client test program will be under the demos/ssl folder of the source folder. Copy the file cli.cpp to your project's folder.


ssl - File Browser
File Edit View Go Bookmarks Tabs Help
Back Forward Up Stop Reload Home Computer Search
Location: /Documents/Downloads/openssl-0.9.8k/demos/ssl
100%
Icon View
Places
Desktop
File System
Network
Floppy Drive
Trash
Documents
Music
Pictures
Videos
cli.cpp
inetdsrv.cpp
serv.cpp
"cli.cpp" will be copied if yo
simpleclient - File Browser
Location: /home/. ./code/simpleclient
cli.cpp
main.c
simpleclient.cbp

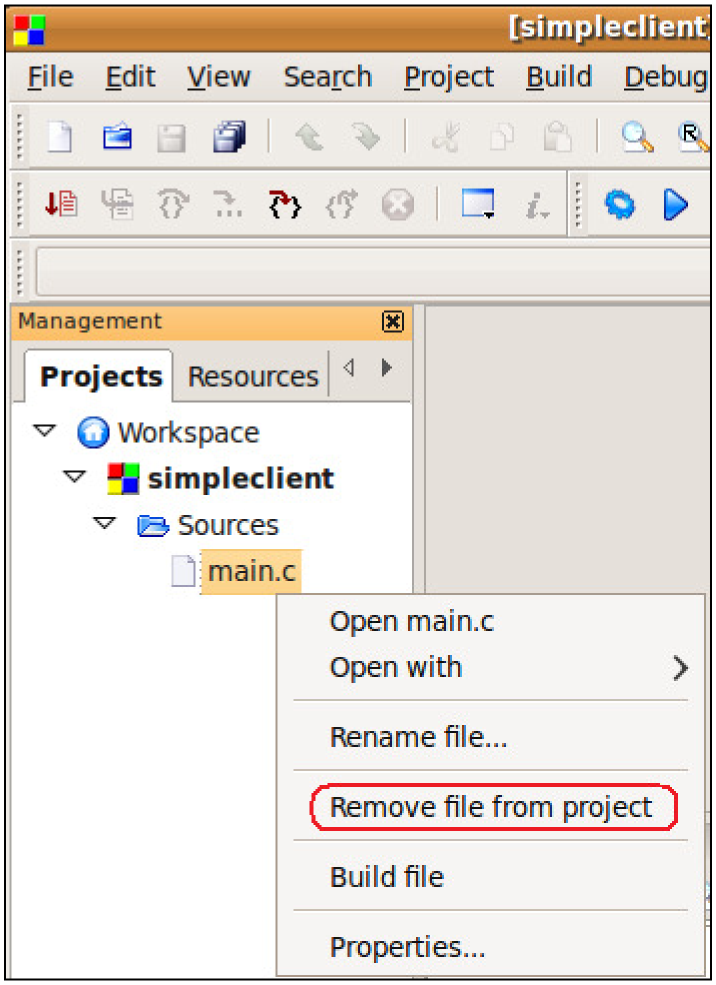
This picture shows that I copied the file cli.cpp to my project's folder. While we are here, we need to do some cleanup.

1.) Delete main.c . We won't be using it.

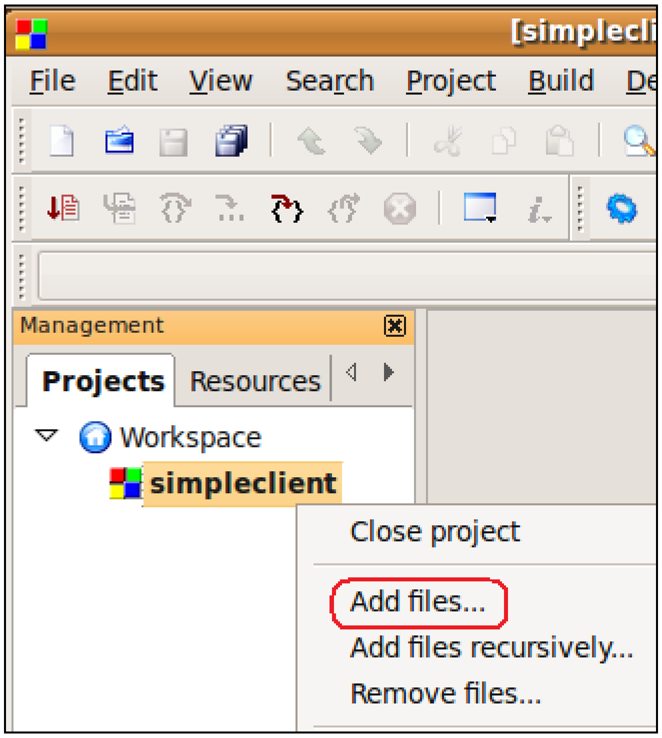
2.) Rename the extension on **cli.cpp** to **cli.c** . CPP programs complicate the process and it is beyond the scope of this document.


simpleclient - File Browser
File Edit View Go Bookmarks Tabs Help
Back
Forward
Up
Stop
Reload
Hom
Location: /home/ /code/simpleclient
100%
Places
Desktop
File System
cli.c
simpleclient.cbp

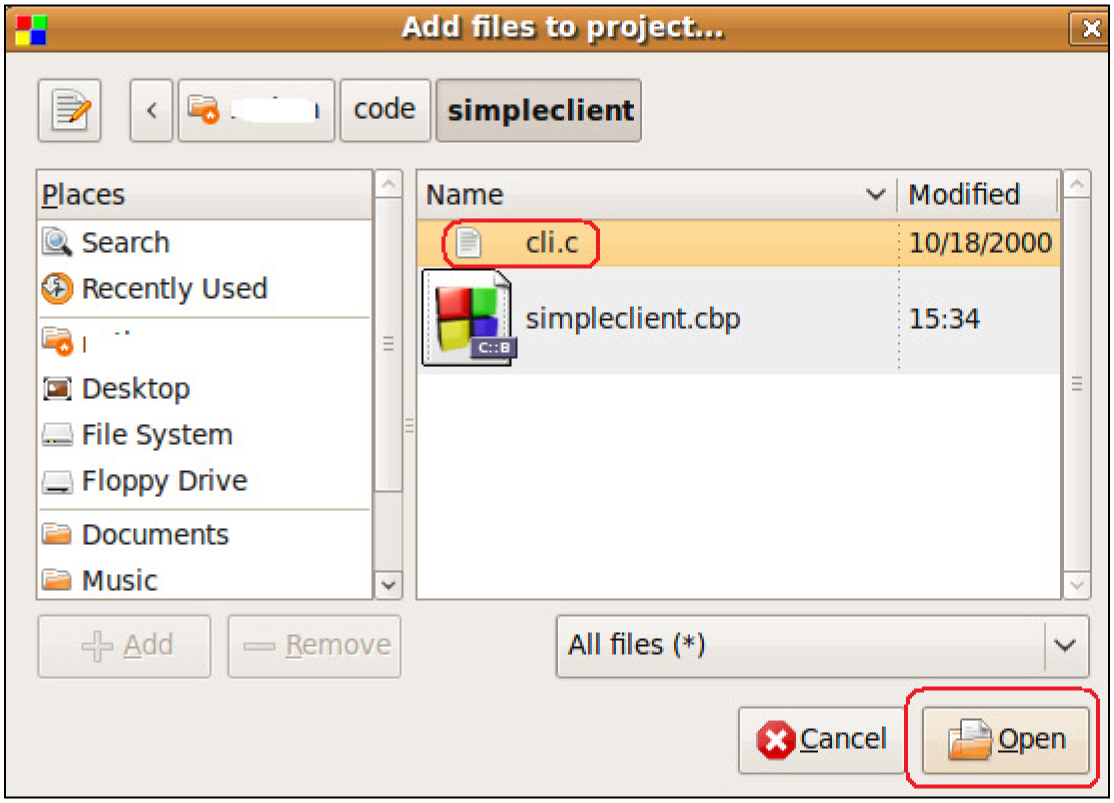
This screen shot shows the renamed file and main.c deleted.



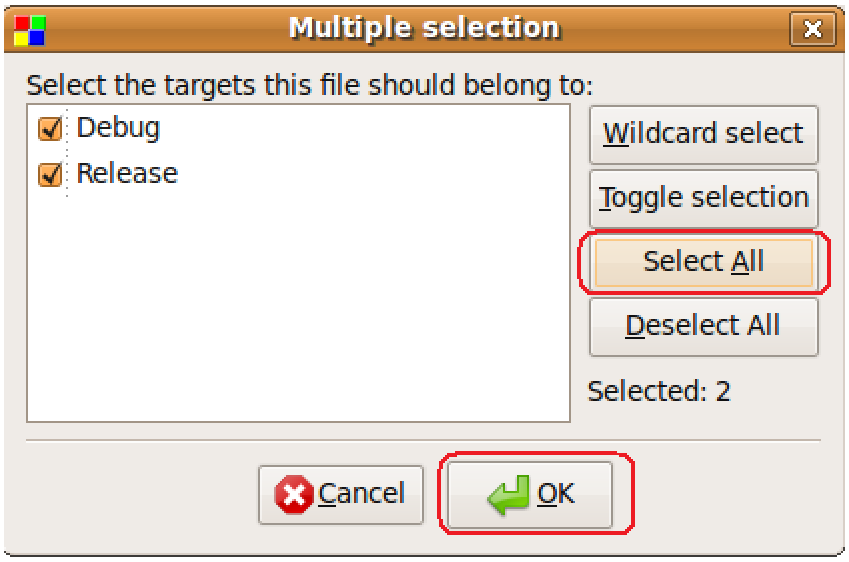
Now open up CodeBlocks again. Find main.c and **right-click** on it and select **Remove file** from project.



Next we need to add in the client source code. **Right-click** on the **Project** name and click **Add files...**



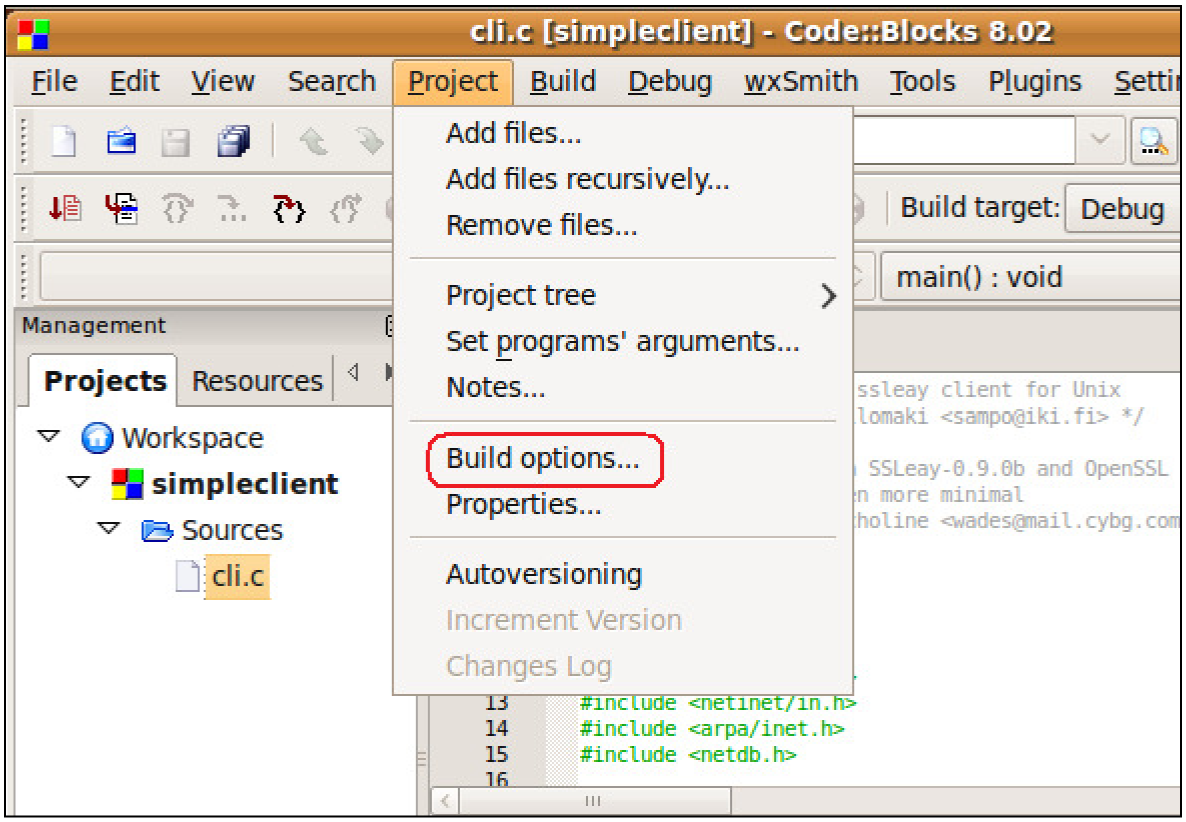
Click on the file and click Open.



Click the **Select All** button and then **OK**

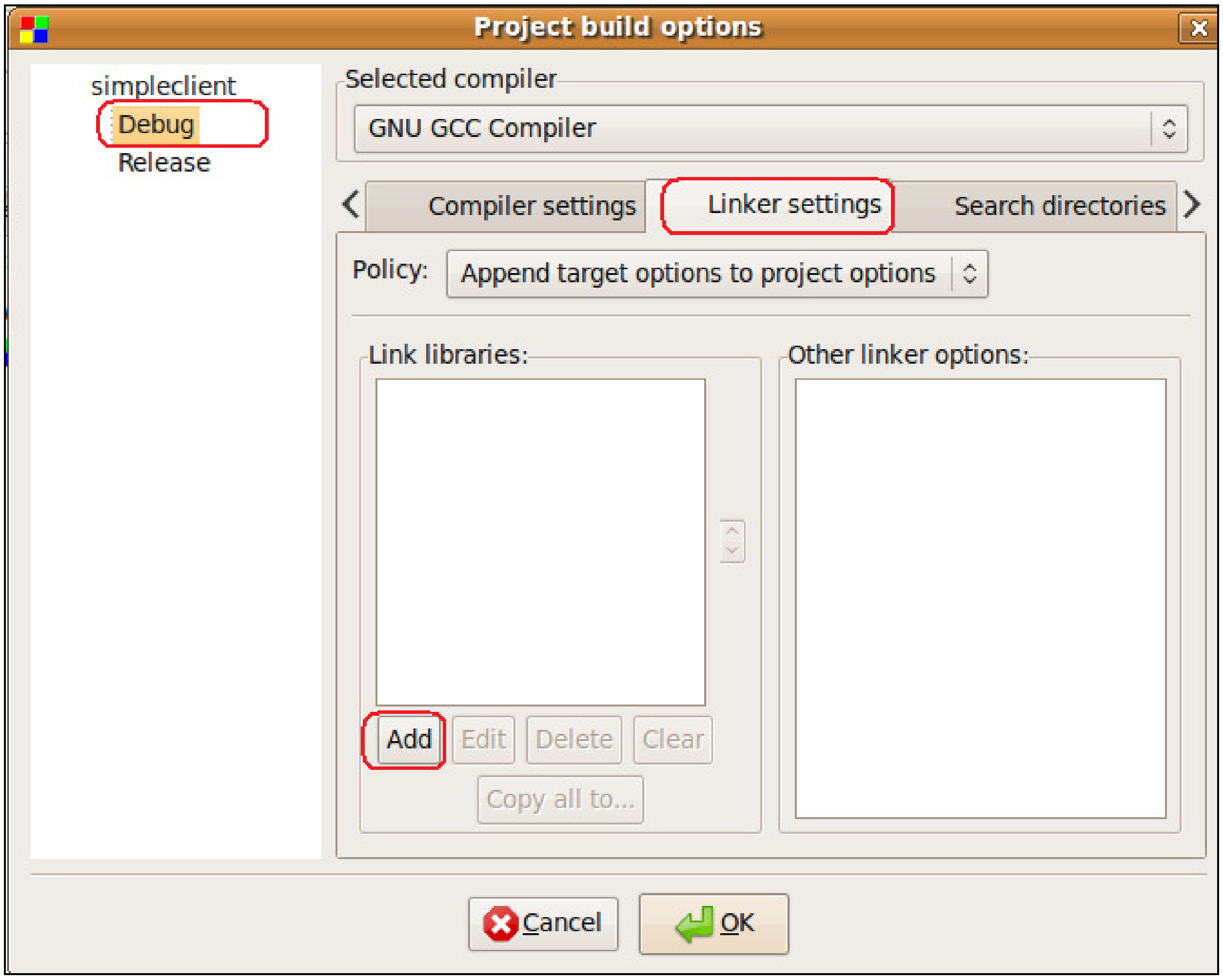
Compiling the Client program

Next we need to add in the OpenSSL libraries so we can build and run the program.

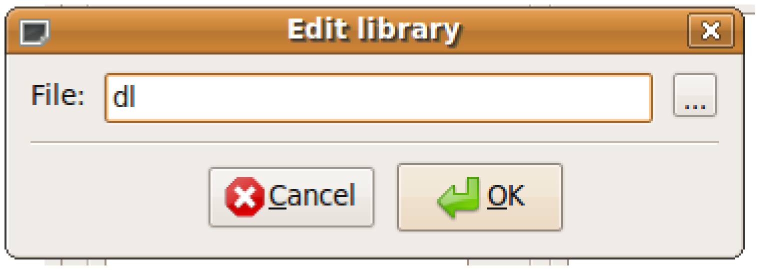


```
ssleay client for Unix
lomaki <sampo@iki.fi> */
SSLeay-0.9.0b and OpenSSL
n more minimal
holine <wades@mail.cybg.com
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

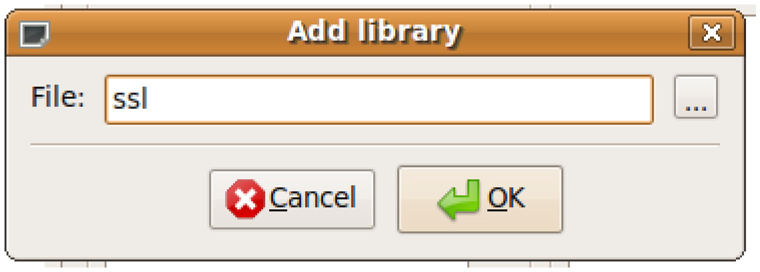
Click on Project -> Build options...



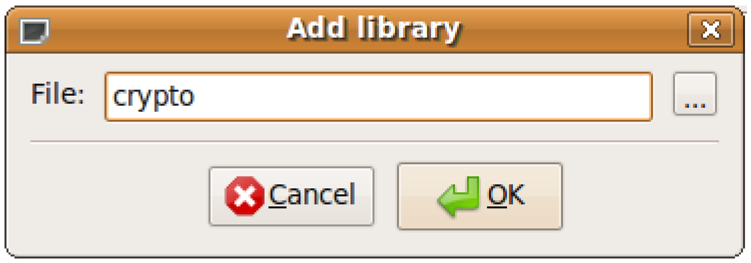
Select the Debug target from the left. Click on the Linker settings tab. Click the Add button to add in some libraries.



Type **dl** and press OK.

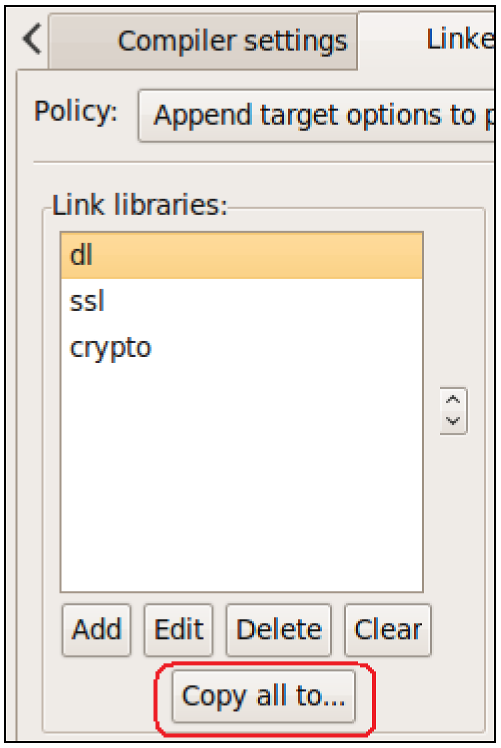


Do the same step and add in **ssl**

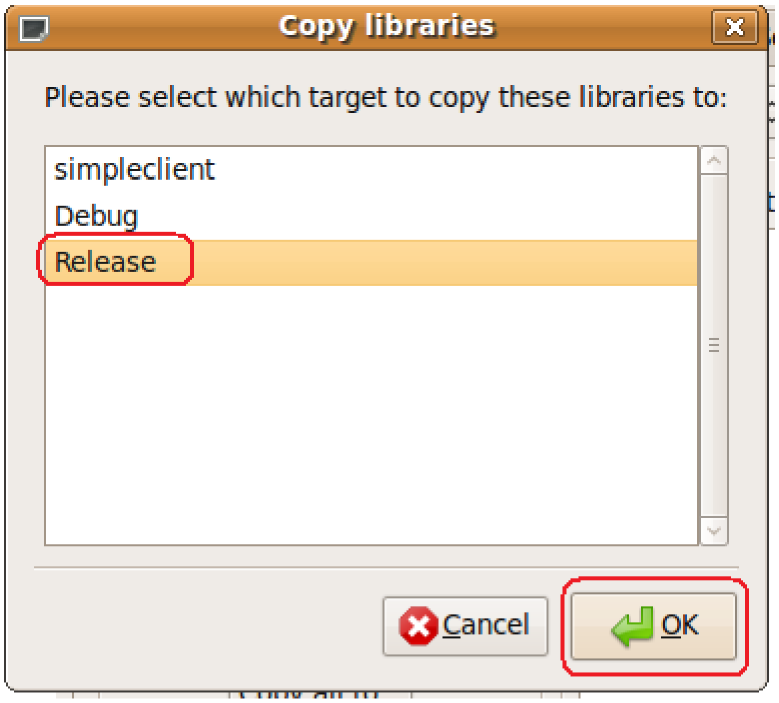


And add **crypto**

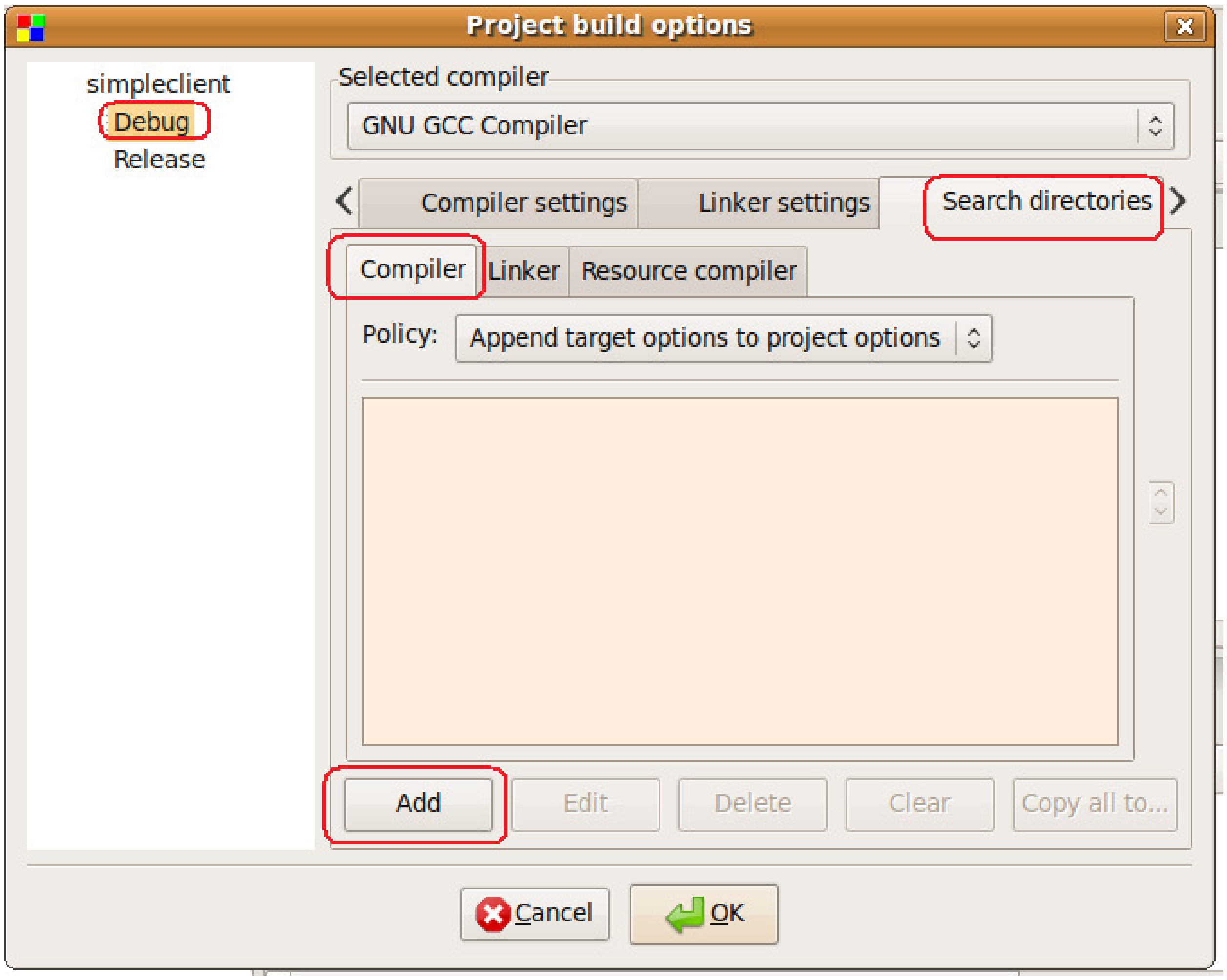
NOTE: ssl must come before crypto or you will get linker errors. If you get the order wrong, there are arrows just to the right side of the window where you can move libraries up and down.



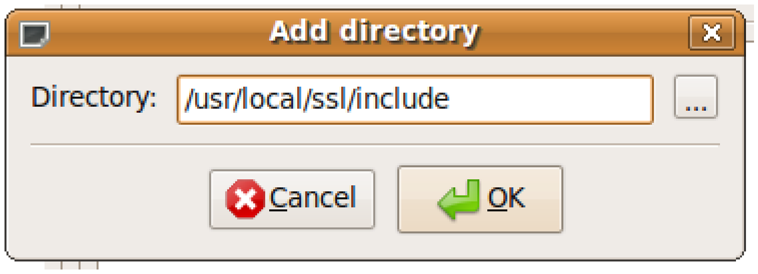
You should now have these three libraries listed. Now click the **Copy all to...** button so we can copy them to the Release build.



Click **Release** and click **OK**.

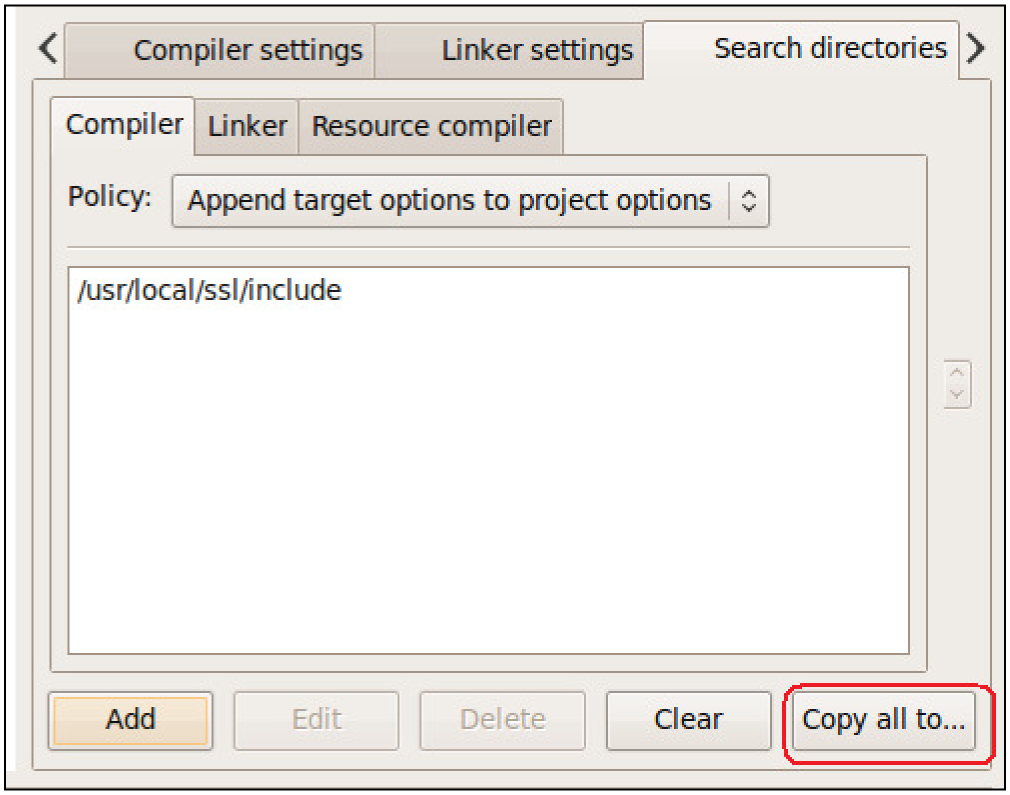


Click the **Search directories** tab and the **compiler** tab and click the **Add** button.



Directory: /usr/local/ssl/include

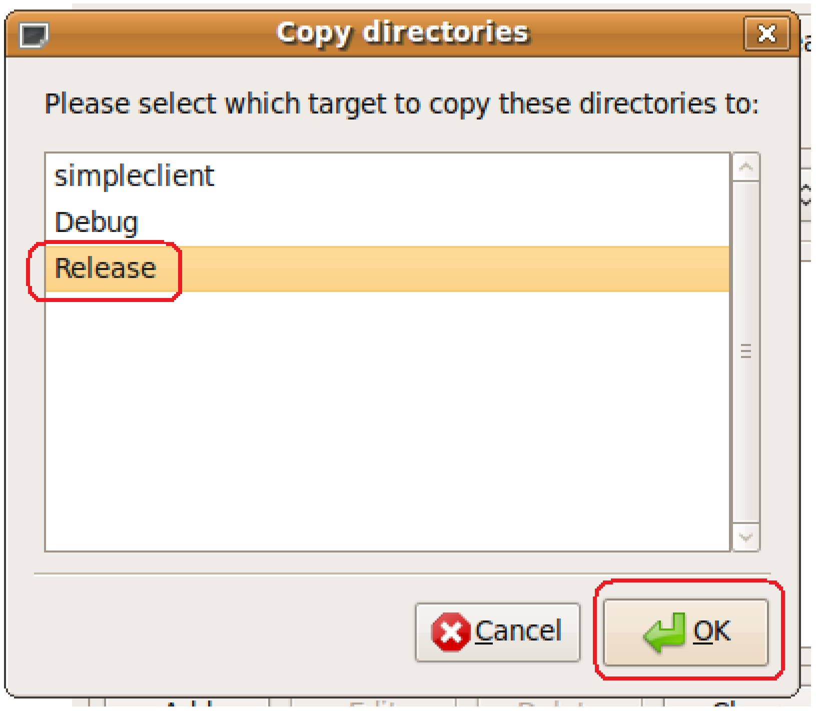
Type **/usr/local/ssl/include** and press **OK**.



Policy: Append target options to project options

/usr/local/ssl/include

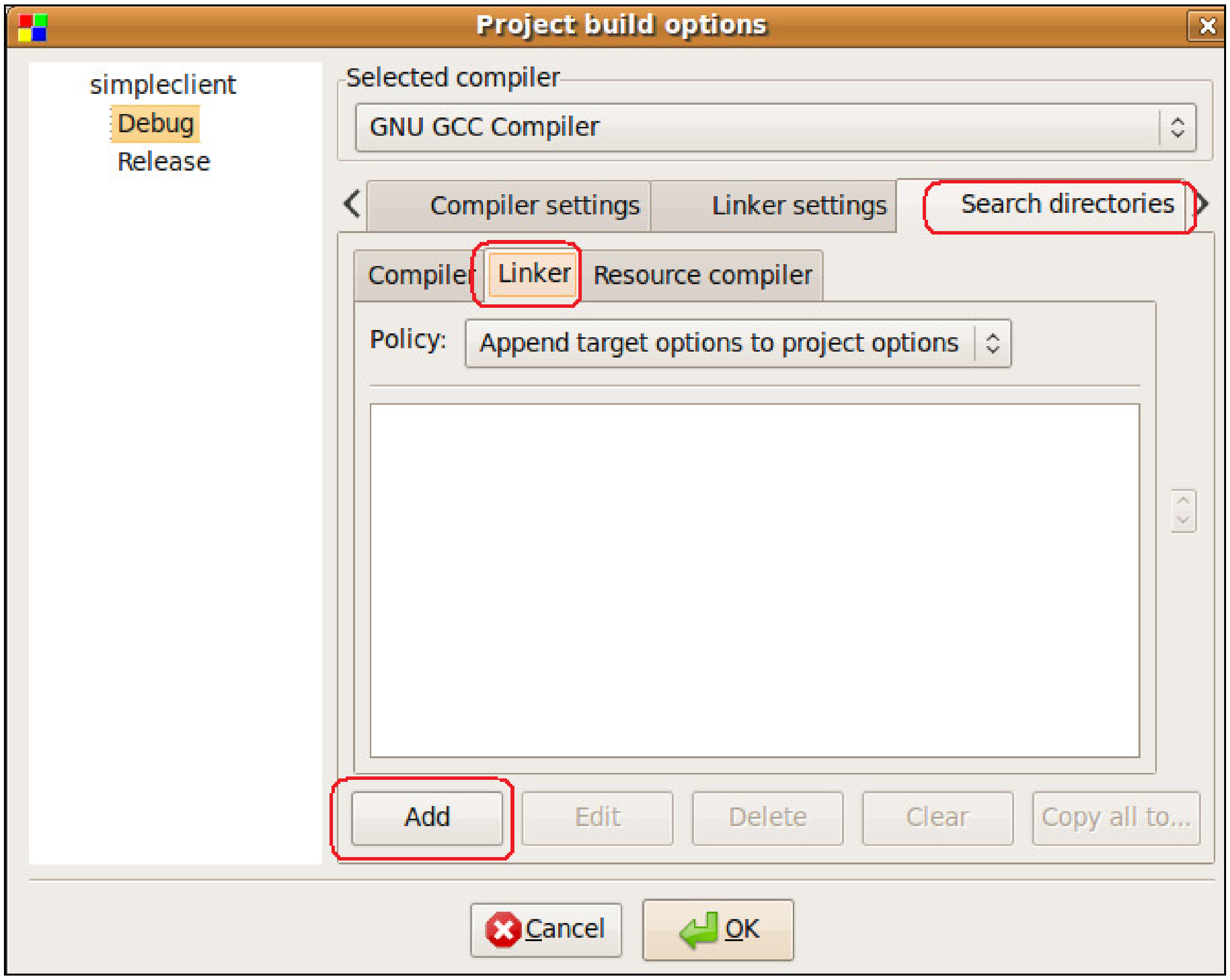
Click the **Copy all to...** button



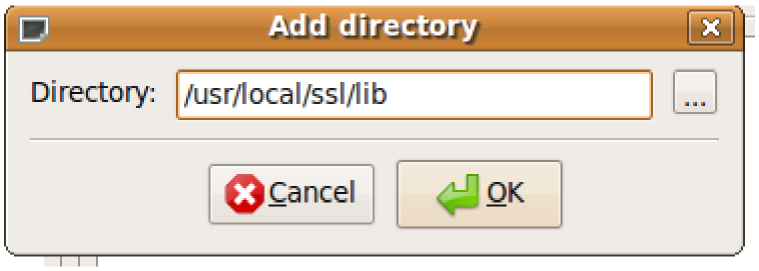
Please select which target to copy these directories to:

simpleclient
Debug
Release

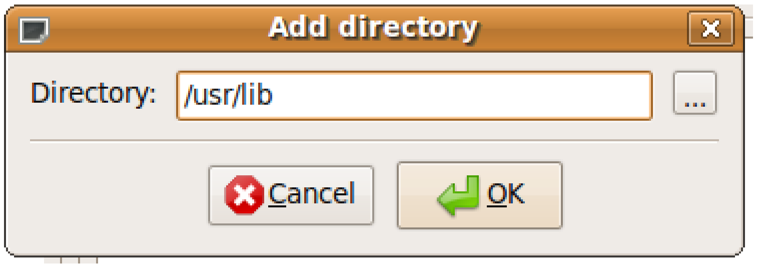
Click the **Release** target and click **OK**.



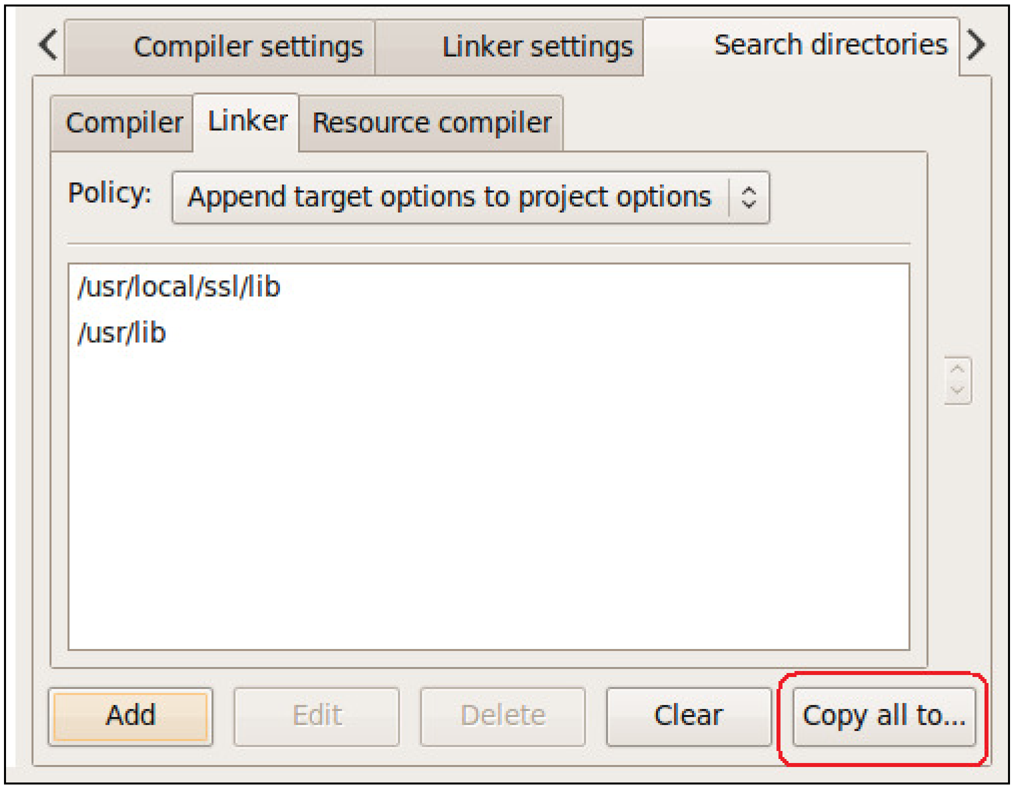
Click the **Search directories** and then click the **Linker tab**. Click **Add**.



Change the directory to **/usr/local/ssl/lib** and click OK.



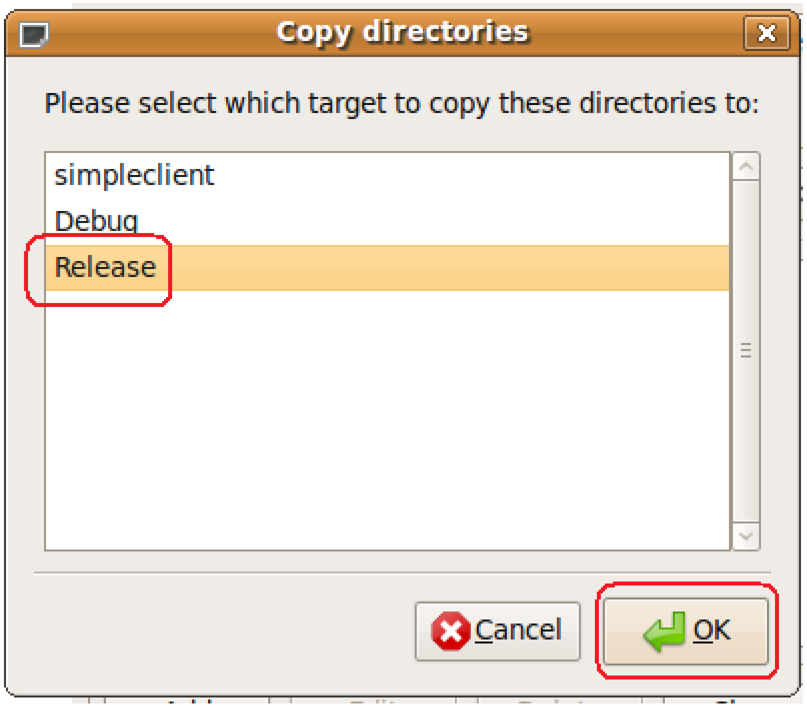
Add another one and type in the name **/usr/lib** and click **OK**.



Policy: Append target options to project options

/usr/local/ssl/lib
/usr/lib

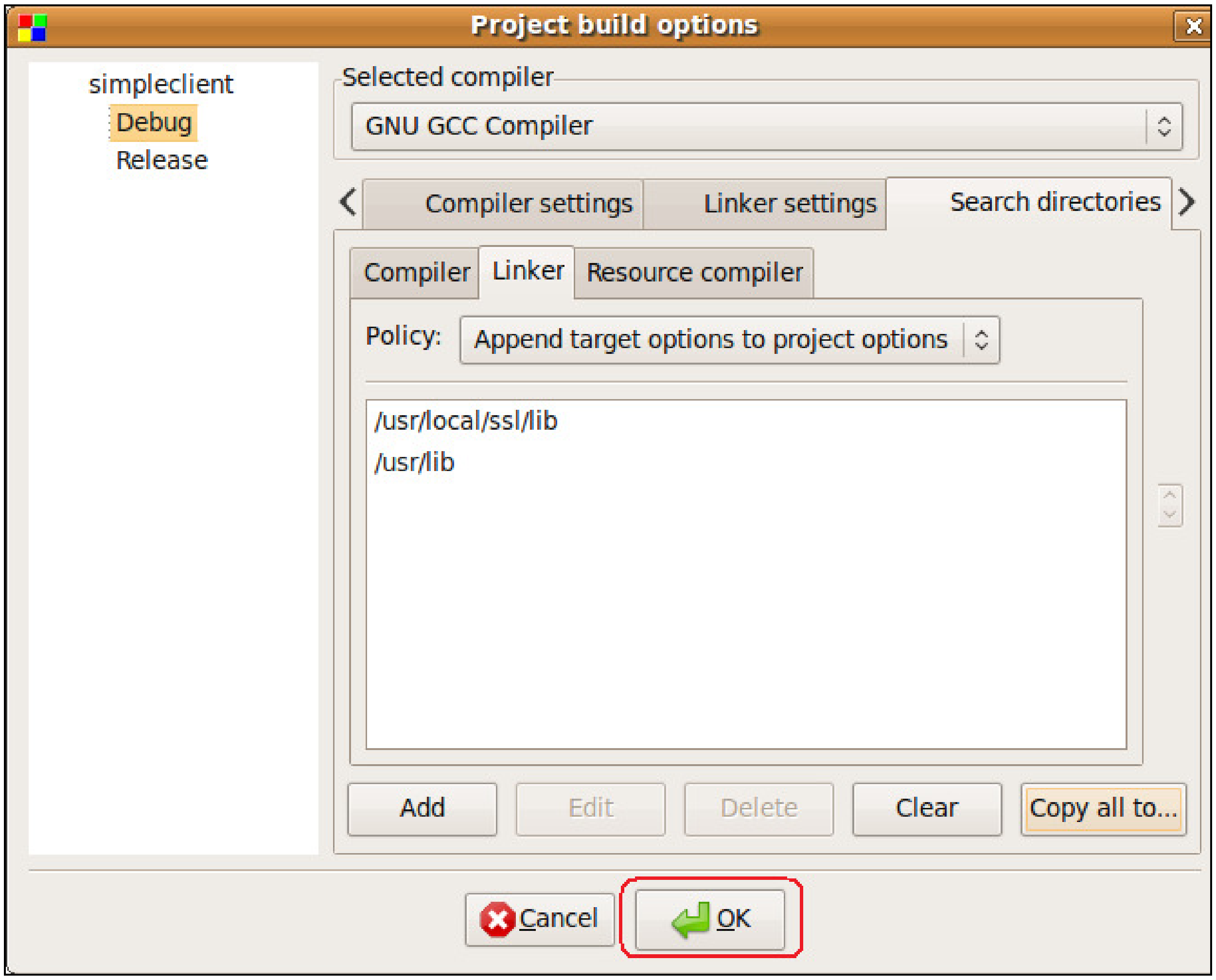
Click the **Copy all to...** button.



Please select which target to copy these directories to:

simpleclient
Debug
Release

Click the Release target and click OK.

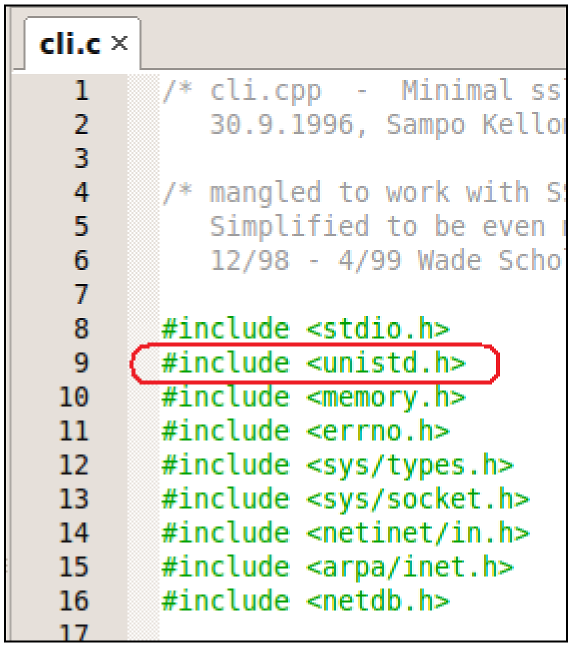


Policy: Append target options to project options

```
/usr/local/ssl/lib
/usr/lib
```

Now click **OK** to save all the settings.

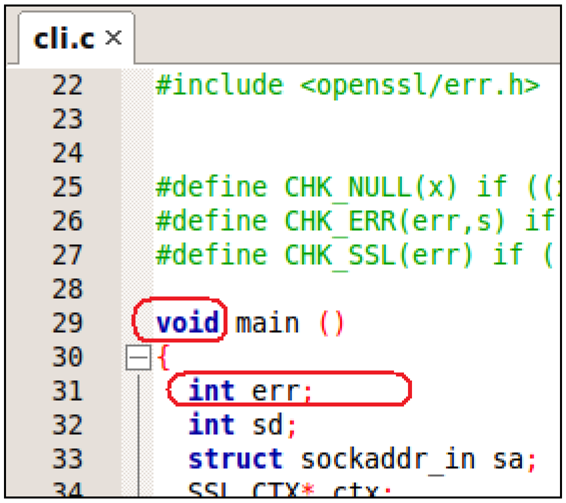
Next, there are a couple warnings we need to get rid of in the code to get a clean compile.



```
/* cli.cpp  -  Minimal ss
   30.9.1996, Sampo Kello

/* mangled to work with S
   Simplified to be even 
   12/98 - 4/99 Wade Scho

#include <stdio.h>
#include <unistd.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

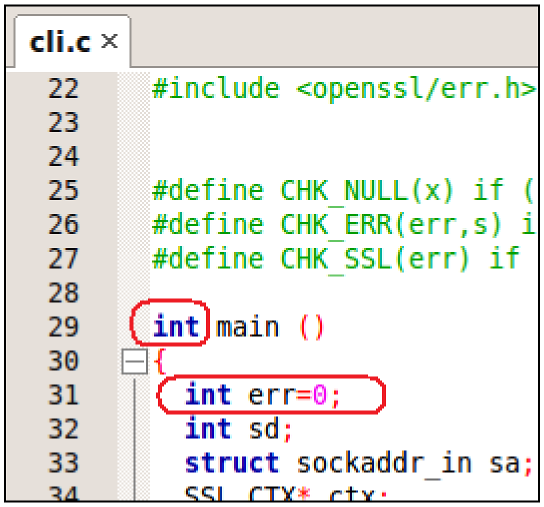
We need to all a line to **#include <unistd.h>**. This gets rid of the warning that a call to close() was implicit.

Next we need to make some code changes.



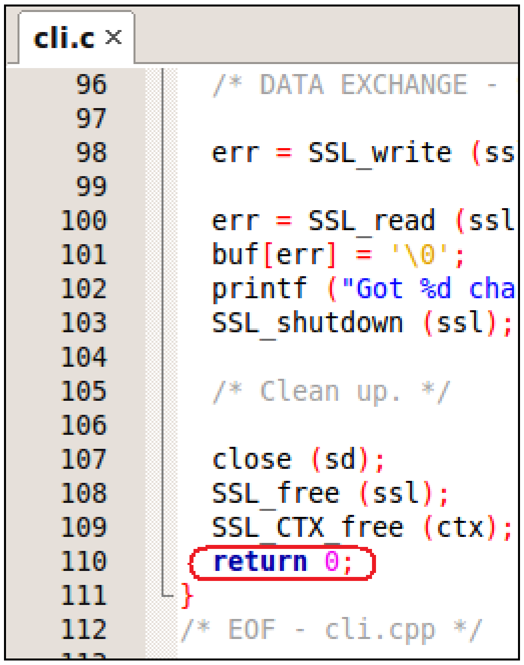
```
#include <openssl/err.h>


#define CHK_NULL(x) if ((
#define CHK_ERR(err,s) if
#define CHK_SSL(err) if (

void main ()
{
  int err;
  int sd;
  struct sockaddr_in sa;
  SSL_CTX* ctx;
```

TO



```
#include <openssl/err.h>


#define CHK_NULL(x) if (
#define CHK_ERR(err,s) i
#define CHK_SSL(err) if

int main ()
{
  int err=0;
  int sd;
  struct sockaddr_in sa;
  SSL_CTX* ctx;
```

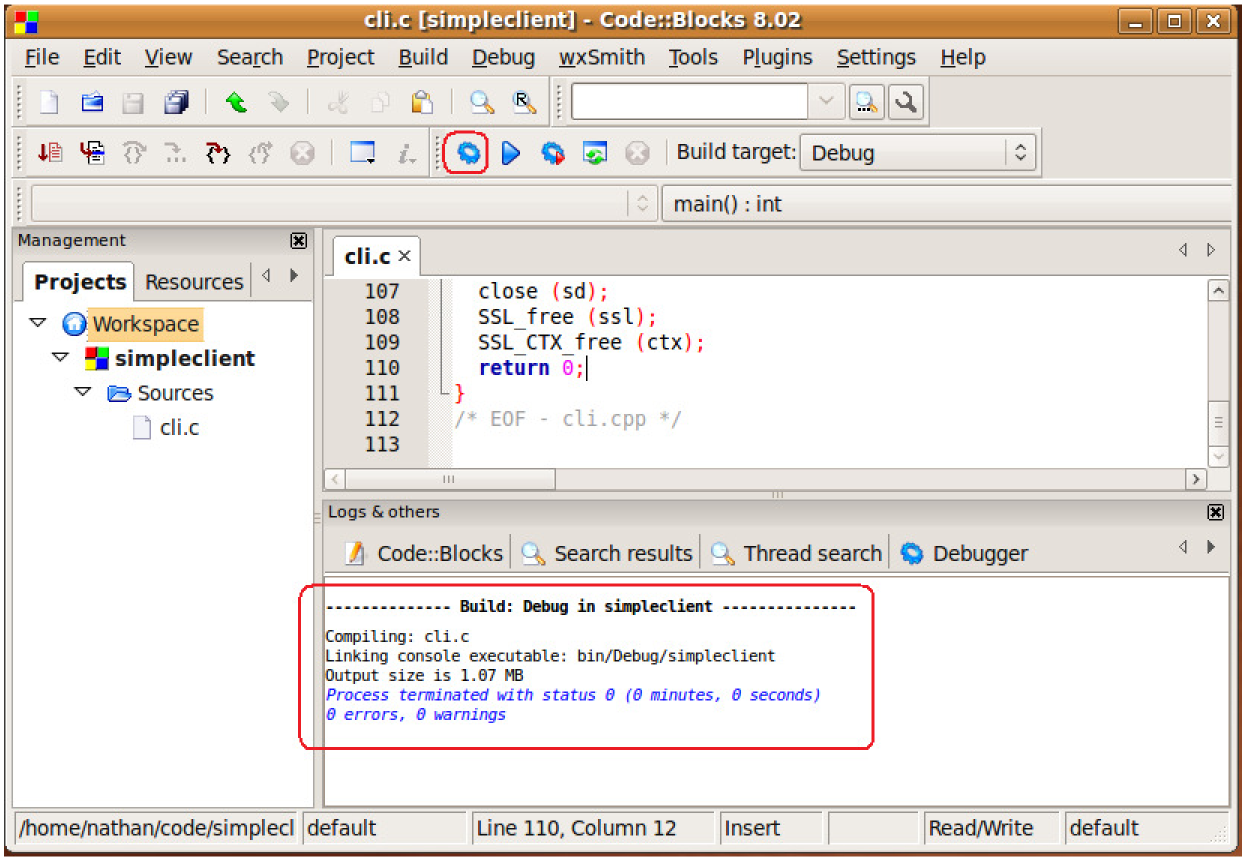
Change the return type of main from **void** to **int**.
Initialize the variable **err** to **0**.



```
    /* DATA EXCHANGE -

    err = SSL_write (ss

    err = SSL_read (ssl
    buf[err] = '\0';
    printf ("Got %d cha
    SSL_shutdown (ssl);

    /* Clean up. */

    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);
    return 0;
}
/* EOF - cli.cpp */
```

At the end of function **main**, add a **return** of **0**;

Now compile and verify that it works!


cli.c [simpleclient] - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
main() : int
Management
Projects Resources
Workspace
simpleclient
Sources
cli.c
Logs & others
Code::Blocks Search results Thread search Debugger
/home/nathan/code/simplecl default Line 110, Column 12 Insert Read/Write default

```
    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);
    return 0;
}
/* EOF - cli.cpp */
```

```
-------------- Build: Debug in simpleclient ---------------
Compiling: cli.c
Linking console executable: bin/Debug/simpleclient
Output size is 1.07 MB
Process terminated with status 0 (0 minutes, 0 seconds)
0 errors, 0 warnings
```

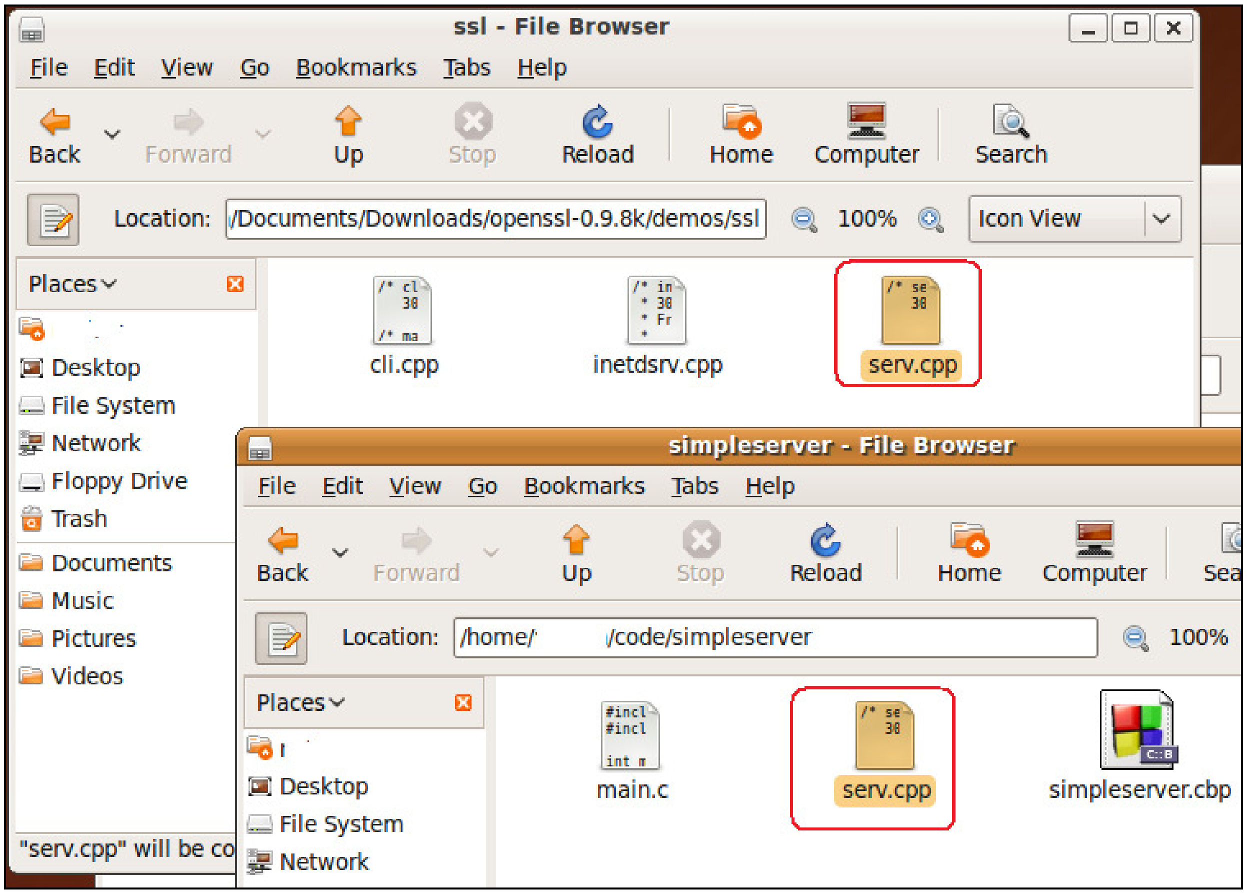
You should get a successful compile.

Creating a Test Program for the OmniFlash (OpenSSL Server)

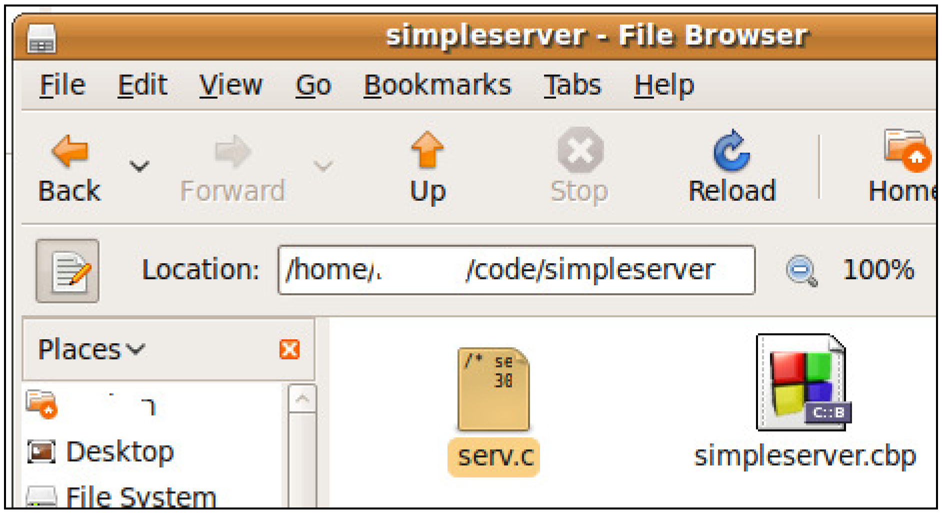
We need to grab the server sample program and do the same steps above. We need to set it up to compile for both Linux and ARM.
I won't show every screen shot this time as the steps are almost identical.

1.) Launch CodeBlocks
2.) Click Create a new project.
3.) Click Console application and Go.
4.) Click C style and click Next.
5.) Name the project. I chose the name **simpleserver**.
6.) Click Finish.

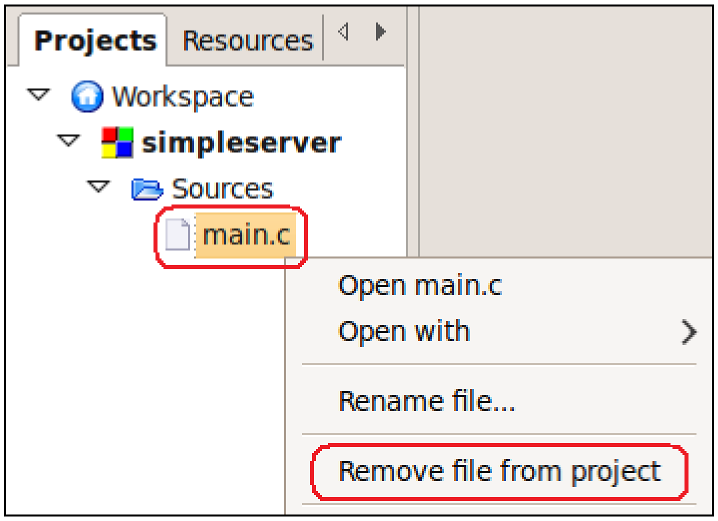
Getting a sample server source code file



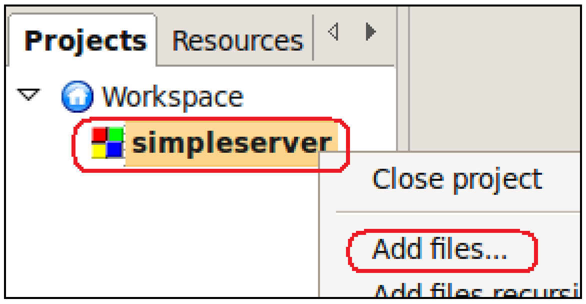
Copy the file serv.cpp from the **demos/ssl** folder under the OpenSSL code we extracted to your project folder.


simpleserver - File Browser
File Edit View Go Bookmarks Tabs Help
Back Forward Up Stop Reload Home
Location: /home/. /code/simpleserver 100%
Places
Desktop
File System
serv.c
simpleserver.cbp

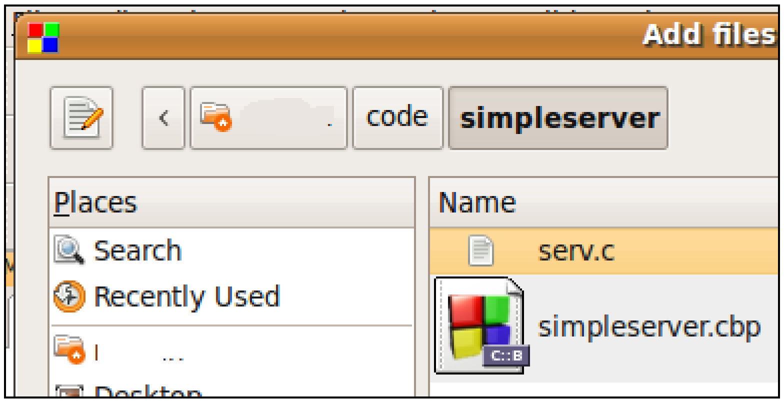
Delete the main.c file and rename serv.cpp to serv.c



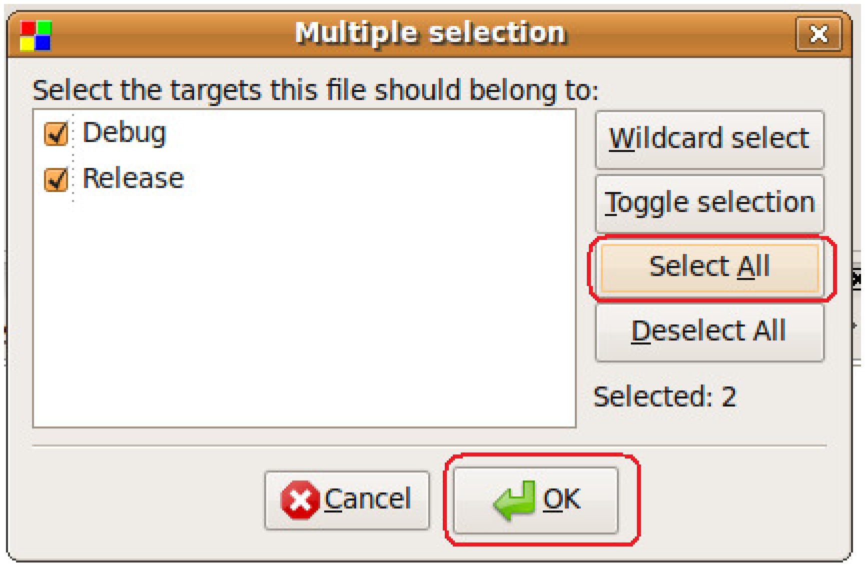
Right-click on main.c and remove it from the project.



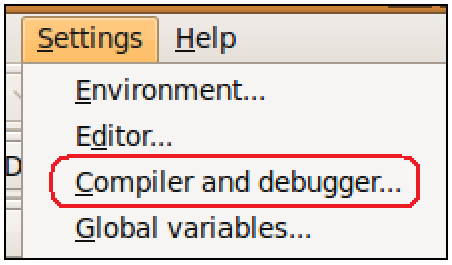
Right-click on the project and select Add files...



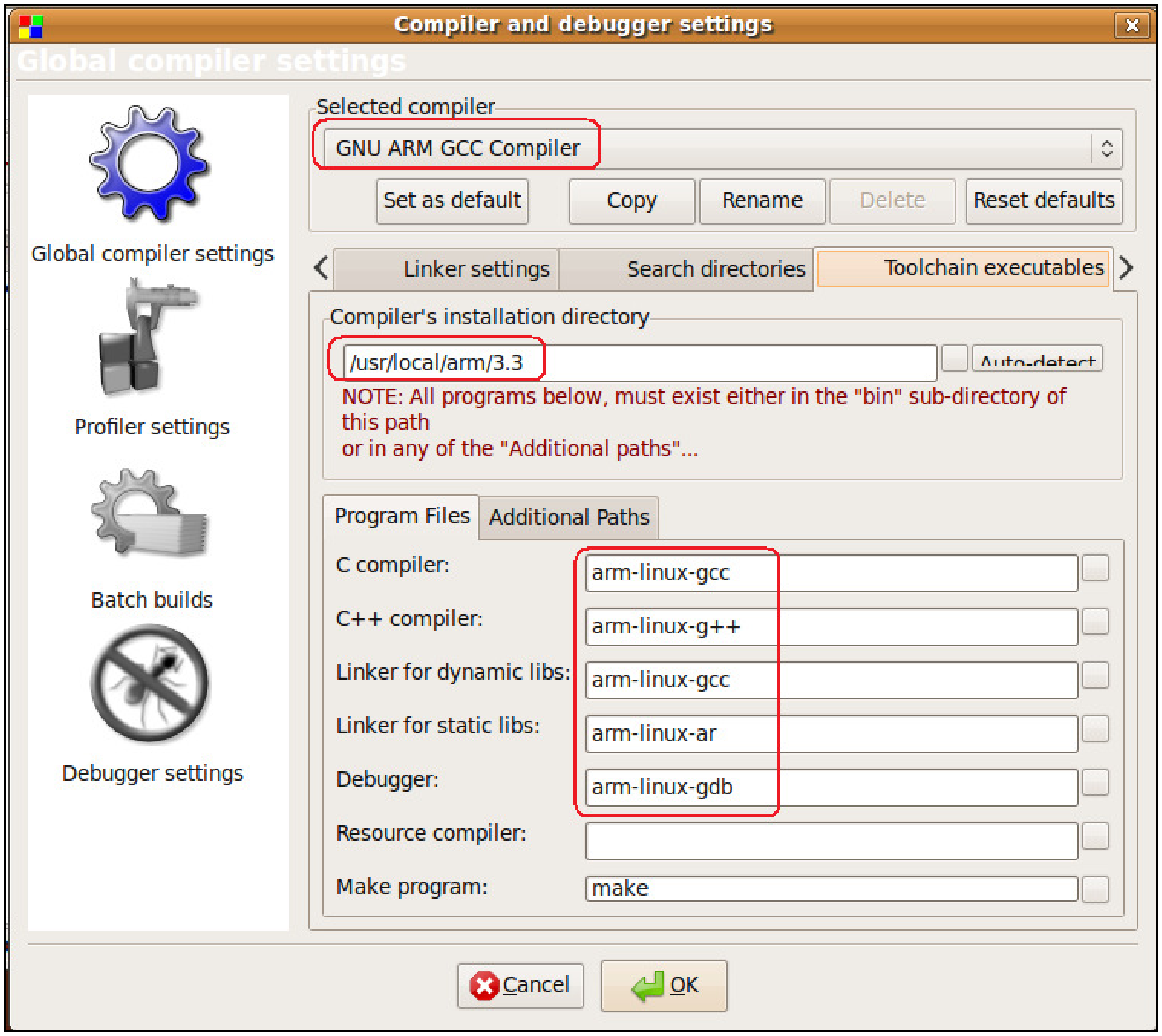
Select the file we just copied to our project and click open.



Click the Select All button and click OK.



If you haven't already set up the cross compiler, check the settings for the ARM compiler.



Compiler and debugger settings

Global compiler settings

Selected compiler: GNU ARM GCC Compiler

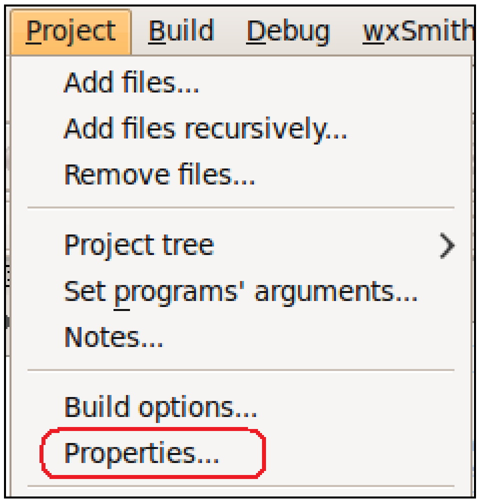
Compiler's installation directory: /usr/local/arm/3.3

NOTE: All programs below, must exist either in the "bin" sub-directory of this path
or in any of the "Additional paths"...

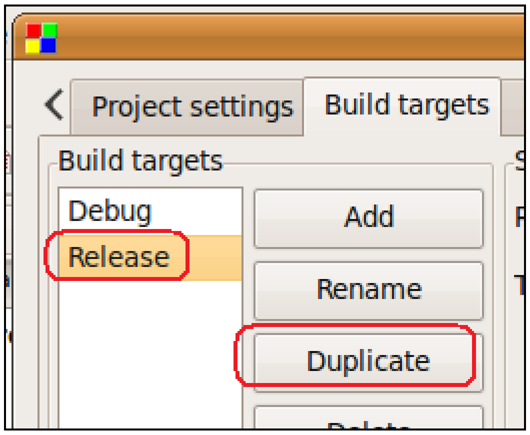
Program Files

C compiler:	arm-linux-gcc
C++ compiler:	arm-linux-g++
Linker for dynamic libs:	arm-linux-gcc
Linker for static libs:	arm-linux-ar
Debugger:	arm-linux-gdb
Resource compiler:	
Make program:	make

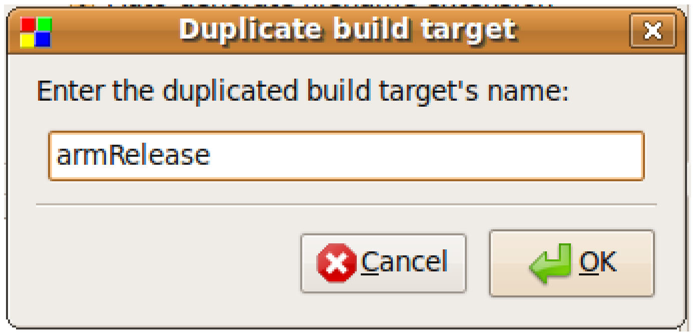
The settings should look like this if using the 3.3 compiler.



Click on Project Properties... so we can set up an additional build target for the ARM processor.



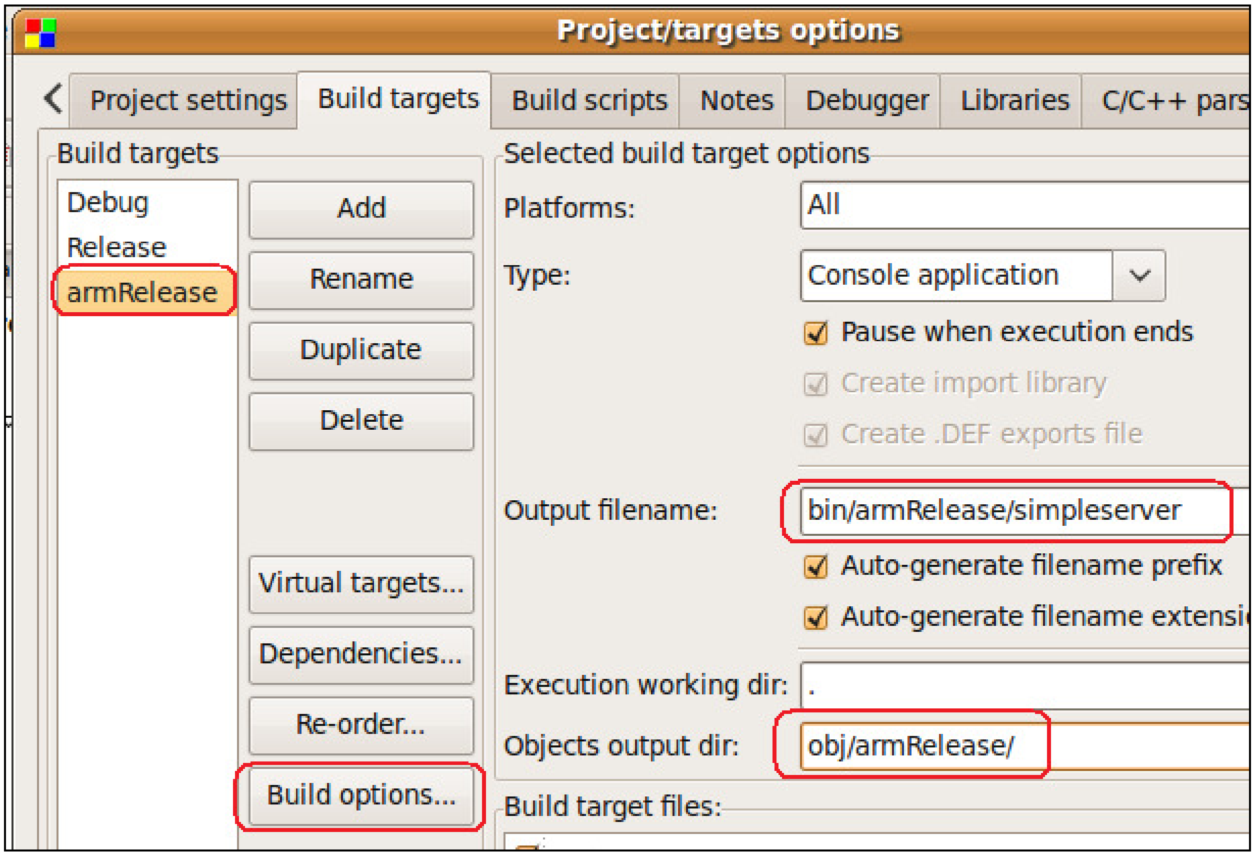
Click on the Build targets tab, click the Release target and click Duplicate.



Enter the duplicated build target's name:

armRelease

Give it a name and click OK.


Project/targets options

Project settings | Build targets | Build scripts | Notes | Debugger | Libraries | C/C++ pars

Build targets: Debug, Release, armRelease

Add | Rename | Duplicate | Delete | Virtual targets... | Dependencies... | Re-order... | Build options...

Selected build target options

Platforms: All

Type: Console application

☑ Pause when execution ends

☑ Create import library

☑ Create .DEF exports file

Output filename: bin/armRelease/simpleserver

☑ Auto-generate filename prefix

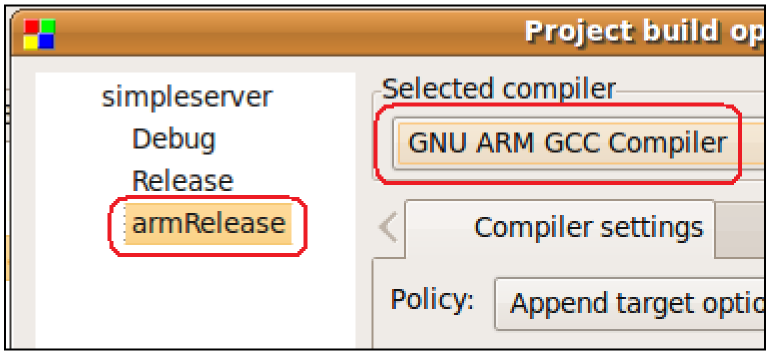
☑ Auto-generate filename extensi

Execution working dir: .

Objects output dir: obj/armRelease/

Build target files:

Make sure the armRelease target is selected. Change the Output filename and Objects output dir so we don't overwrite our regular ones when we build. After you have changed the directories, click the Build options...


Project build op

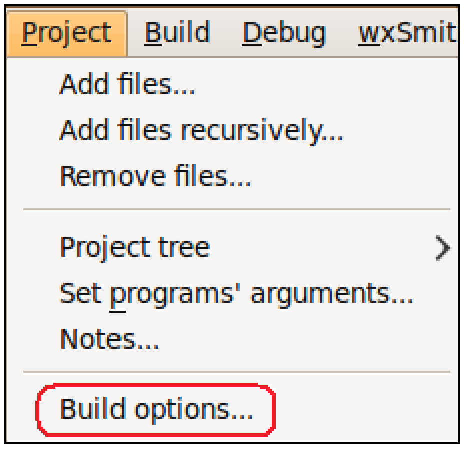
simpleserver
Debug
Release
armRelease

Selected compiler: GNU ARM GCC Compiler

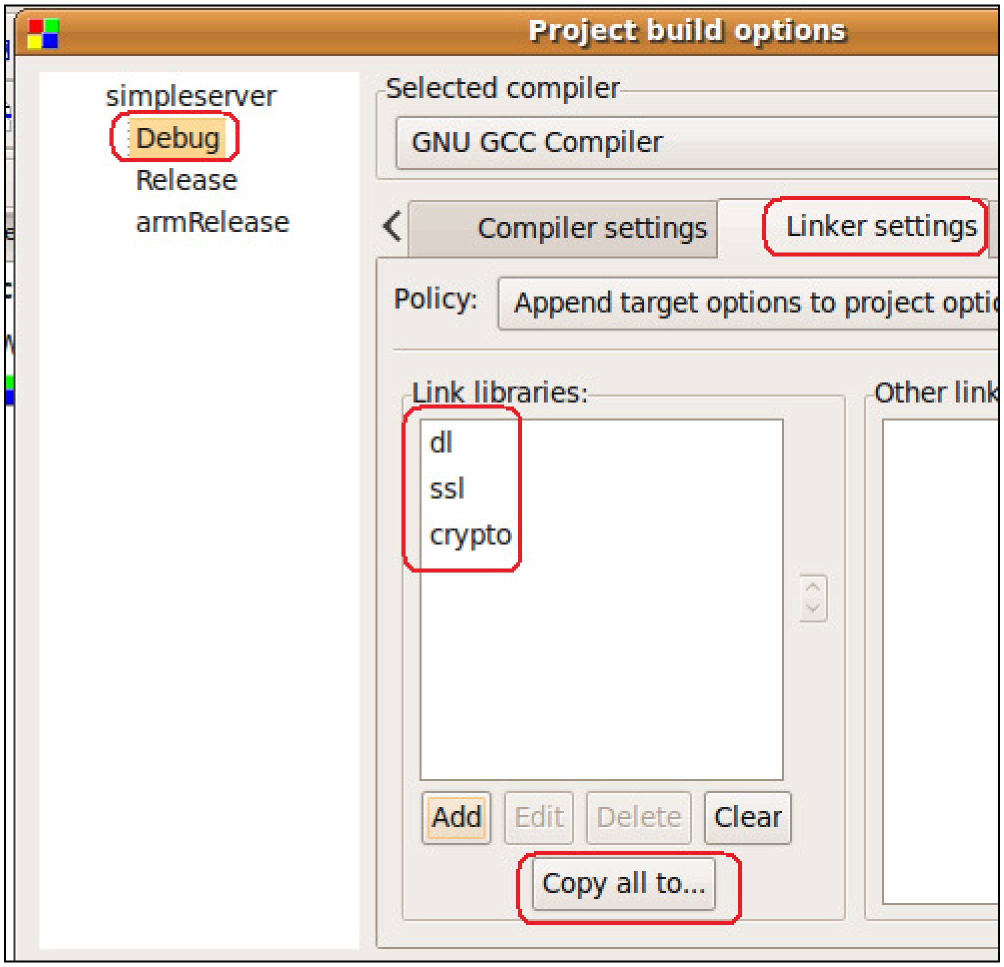
Compiler settings

Policy: Append target optic

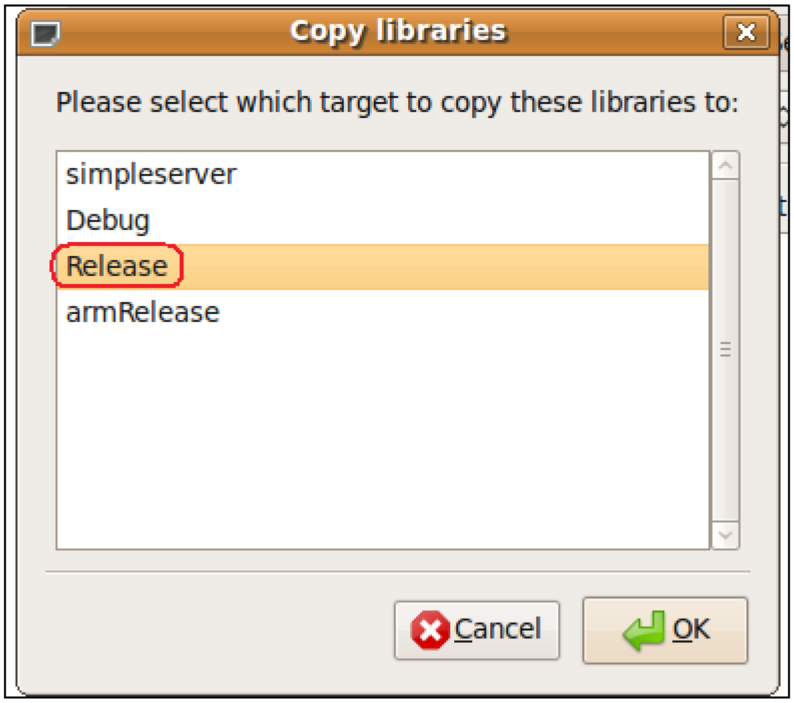
Change the compiler to the ARM GCC compiler.
Click OK to this window and OK to the other window to save the settings.



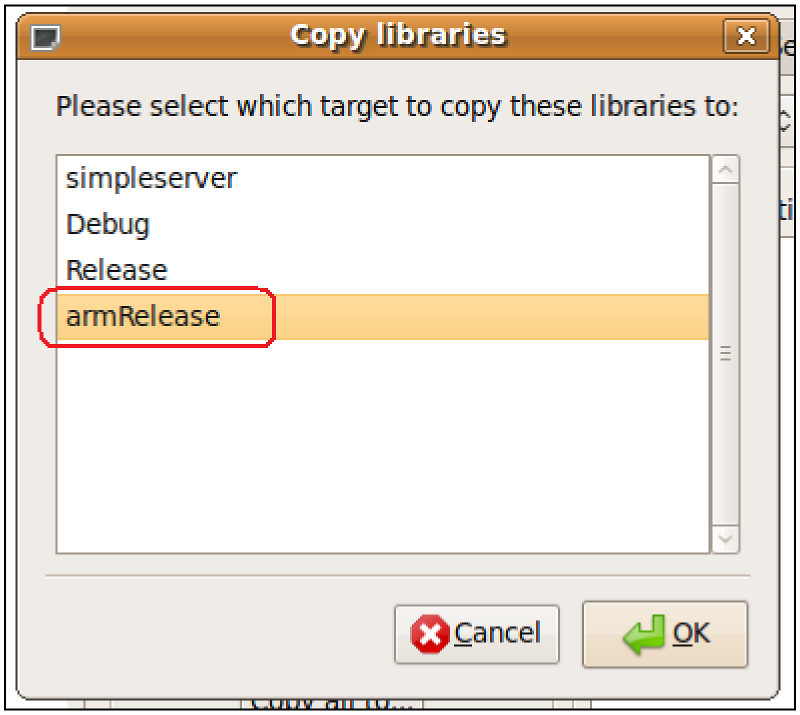
Click on Project -> Build options... to set up all the build dependencies.



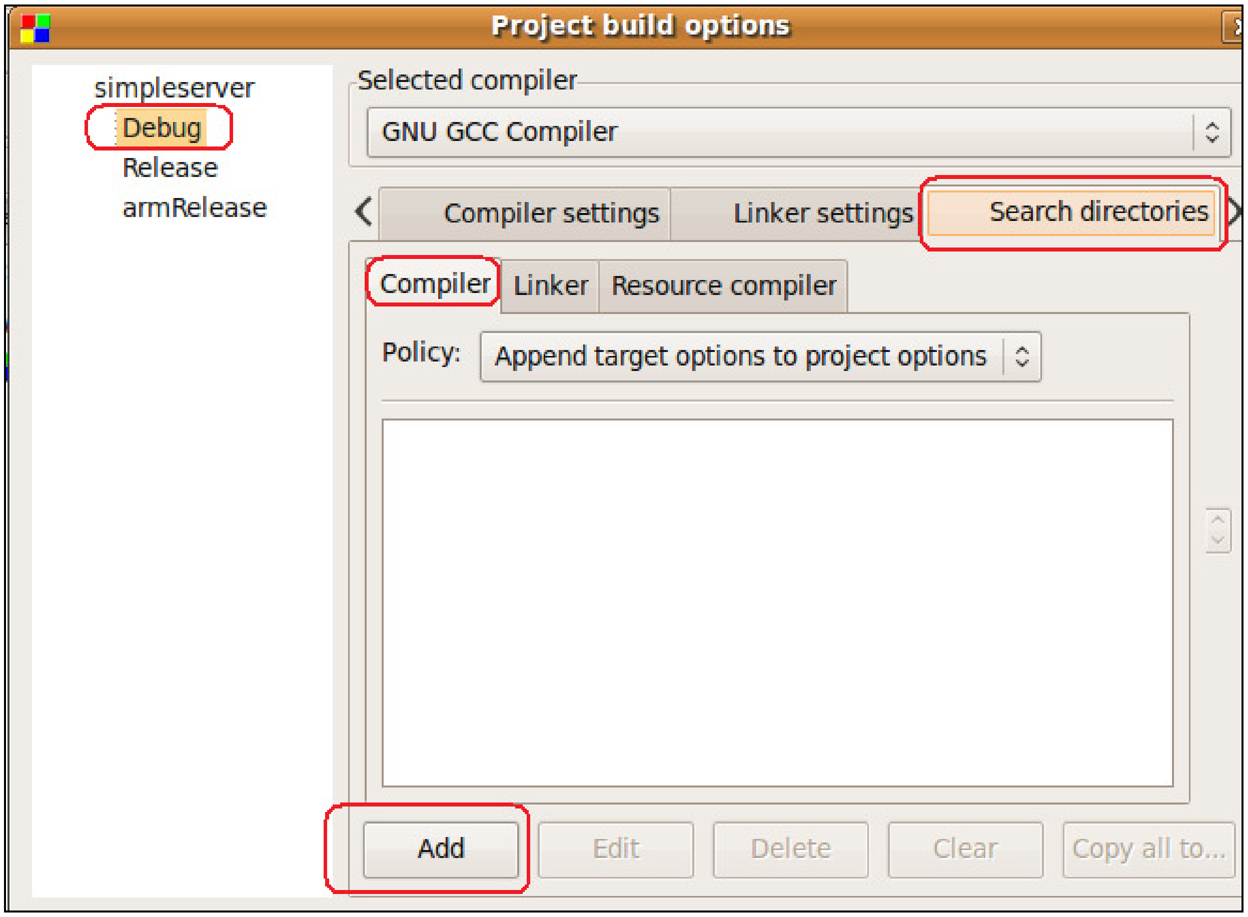
Click on the Debug target. Click the Linker settings and then add the following Link libraries IN THIS ORDER. The click the Copy all to... button.



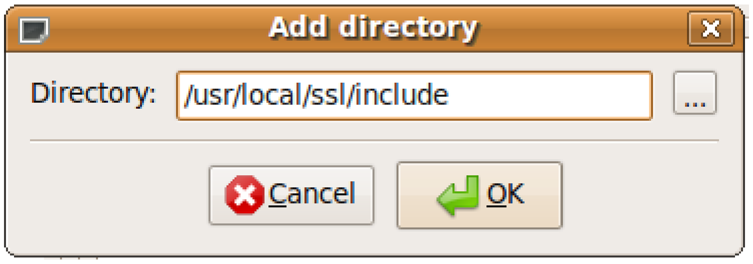
Copy it to the Release target.



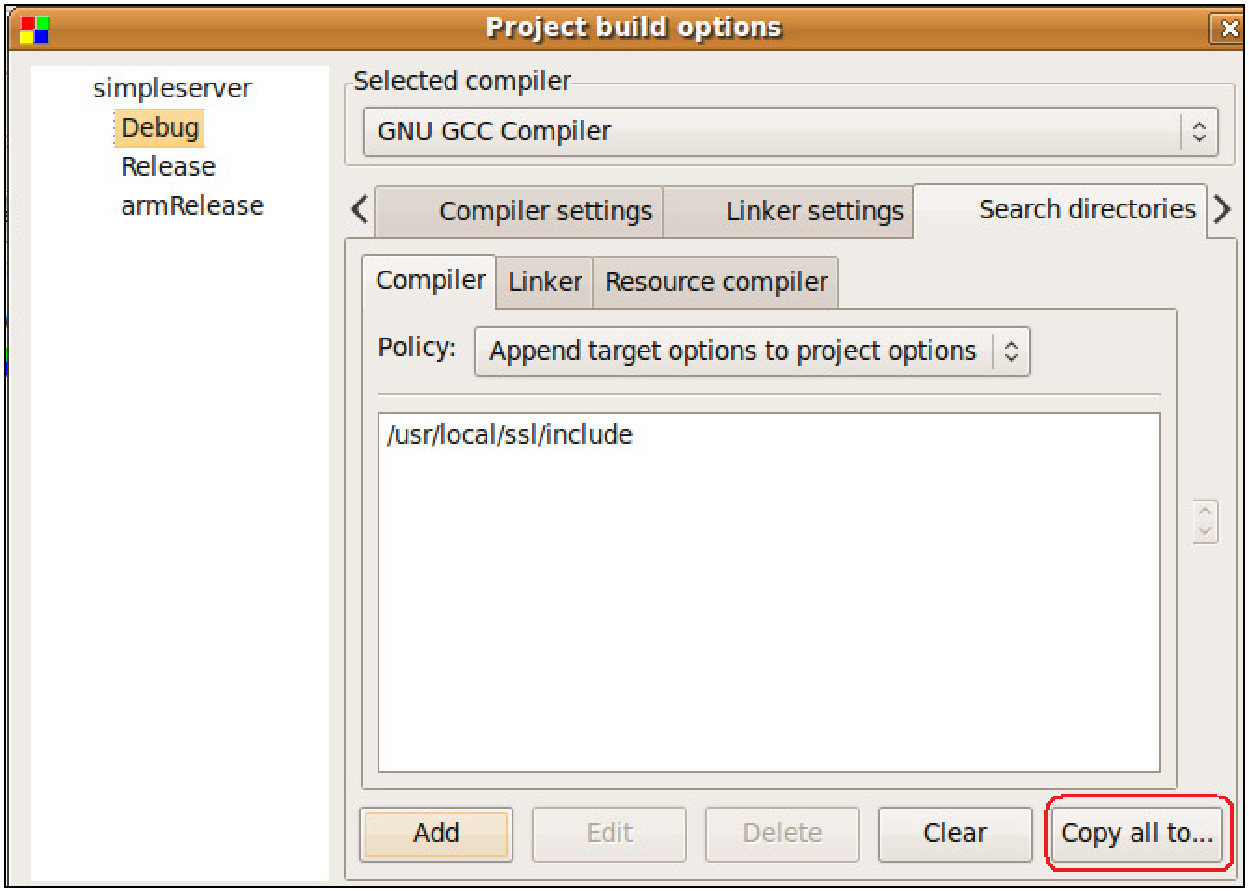
Also copy it to the armRelease target.



Select the Search directories tab for the Debug target. Click the Add button.



Add **/usr/local/ssl/include** to the list. Note: This is NOT the ARM version. This is the regular Linux version.

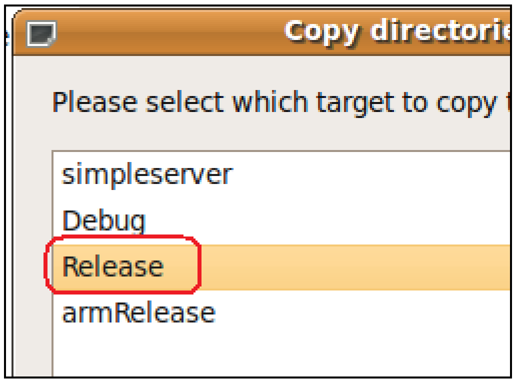


Selected compiler: GNU GCC Compiler

Policy: Append target options to project options

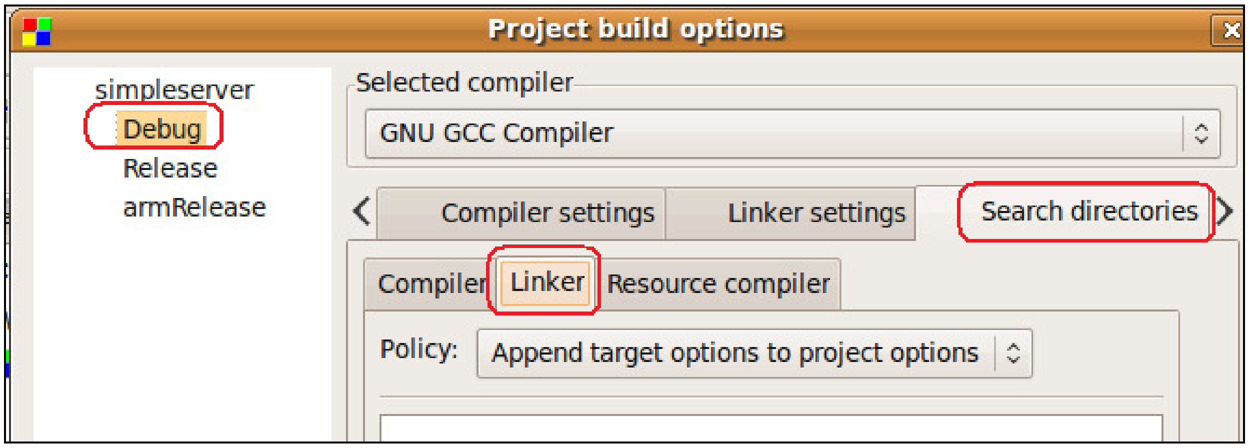
/usr/local/ssl/include

Now click the Copy to all... button.

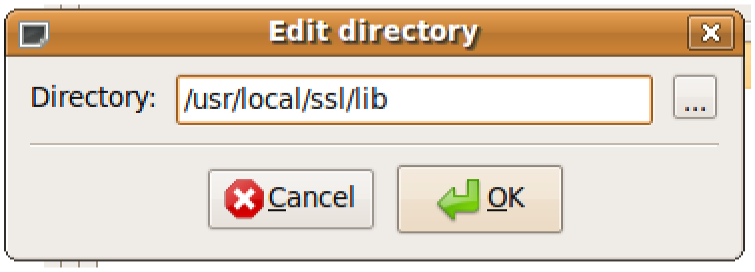


Please select which target to copy t

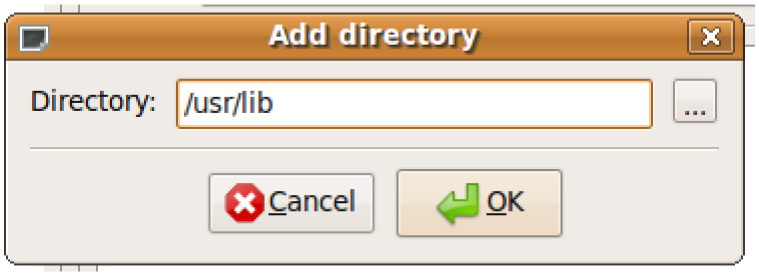
Click the Release target and click OK. Do NOT add this to the armRelease target. These are the wrong includes for ARM.



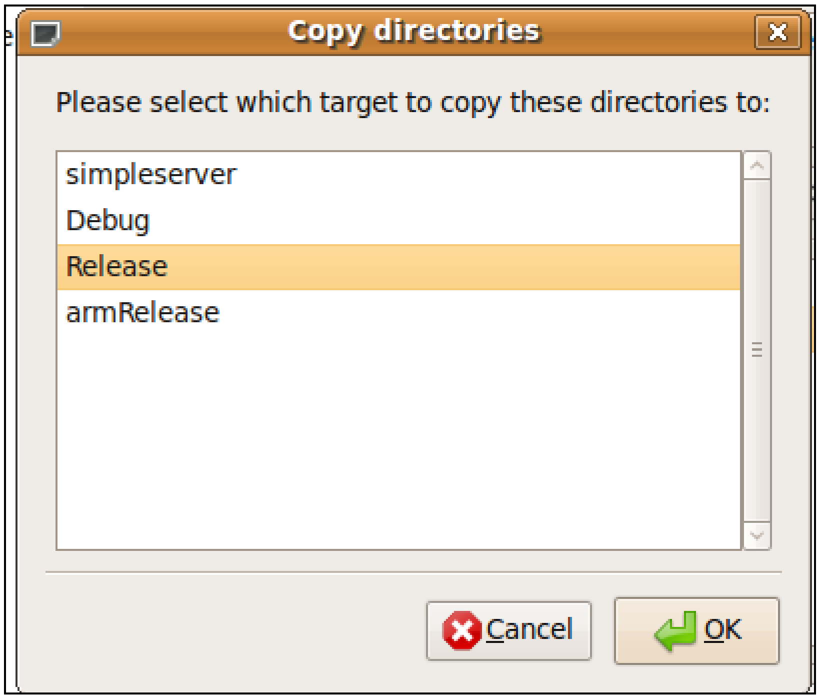
Click the Linker tab under the Search directories on the Debug target.



Click the Add button and **/usr/local/ssl/lib**



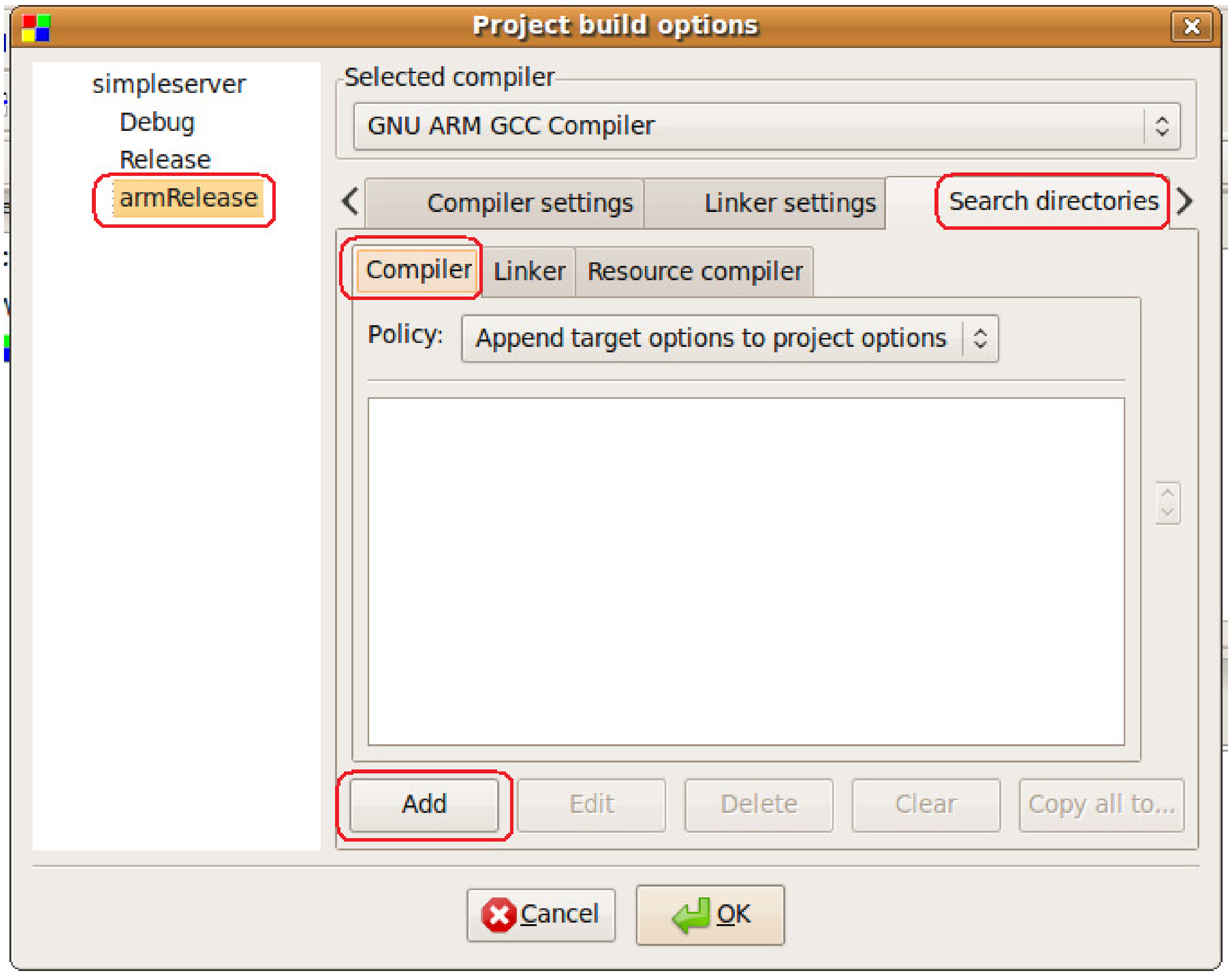
Also add **/usr/lib**


Copy directories

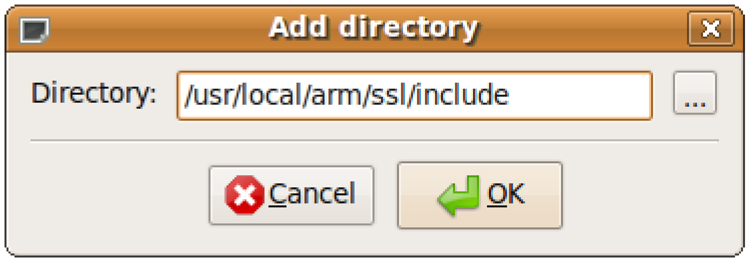
Please select which target to copy these directories to:

simpleserver
Debug
Release
armRelease

Click the Copy all to... button and select the Release target and clock OK. Do NOT add these to the armRelease target.

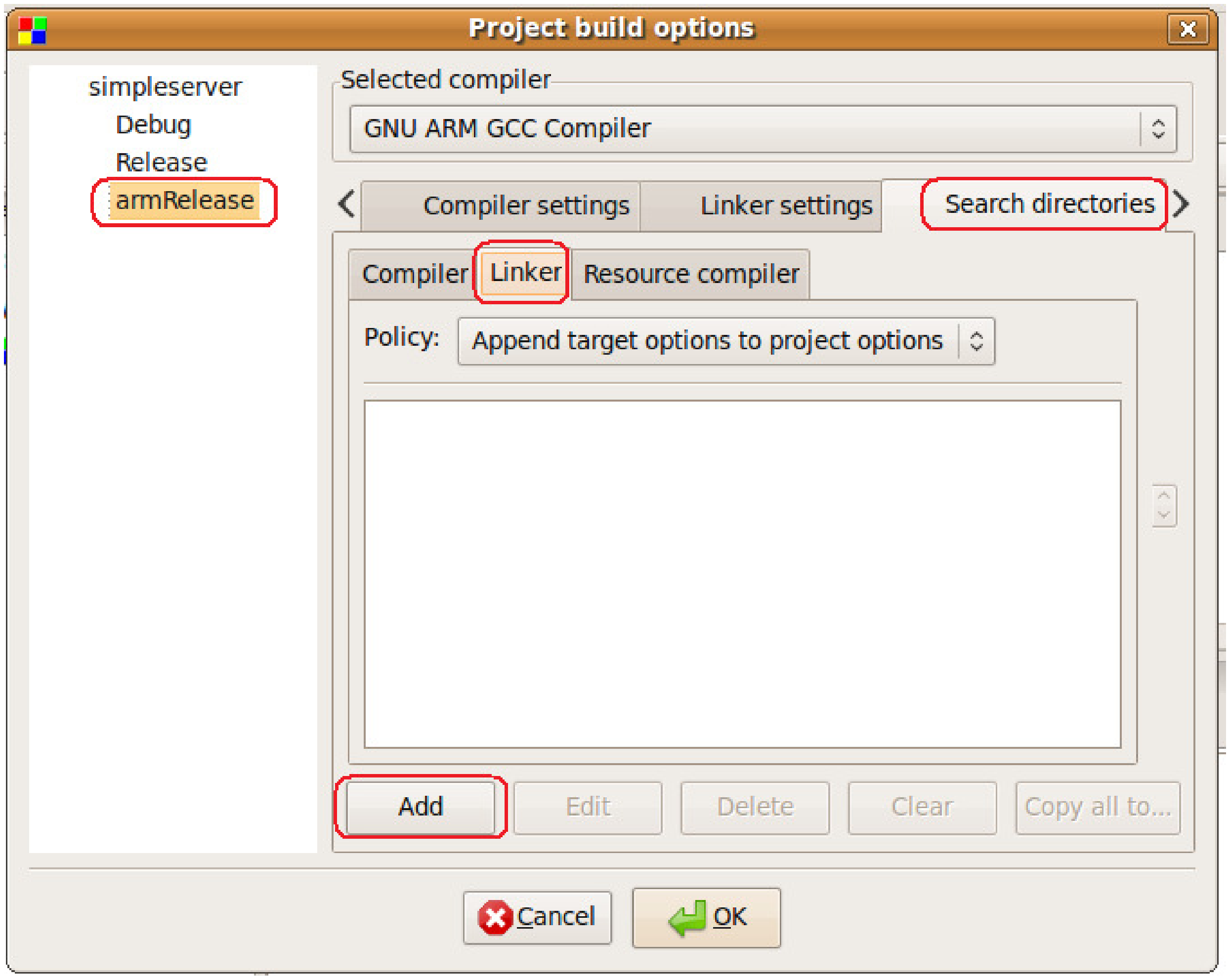


Now click the armRelease target. Click on the Search directories tab and then the Compiler tab. Click Add .



Directory: /usr/local/arm/ssl/include

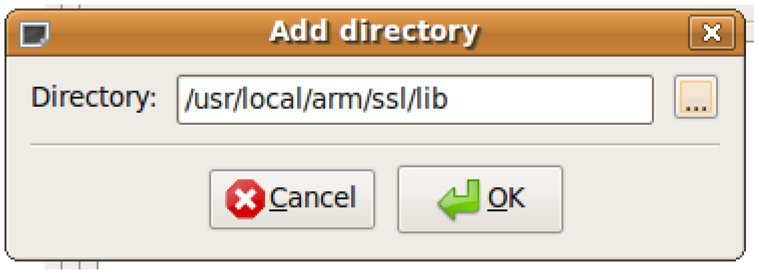
Enter the path to the ARM ssl includes. Enter **/usr/local/arm/ssl/include**



Selected compiler: GNU ARM GCC Compiler

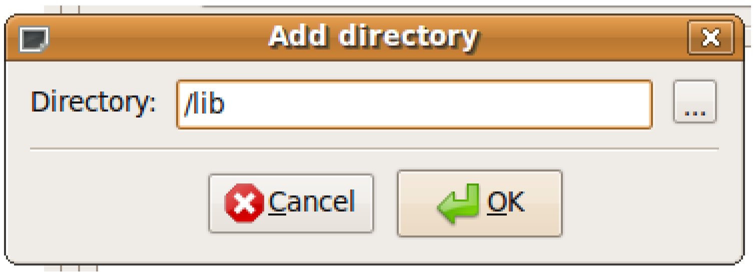
Policy: Append target options to project options

Now click the Linker tab under the Search directories. Click Add.



Directory: /usr/local/arm/ssl/lib

Add **/usr/local/arm/ssl/lib** to the path. This is the path to the ARM SSL libraries. We will need this path later when we transfer files to the OmniFlash. These libraries will have to be transferred too.

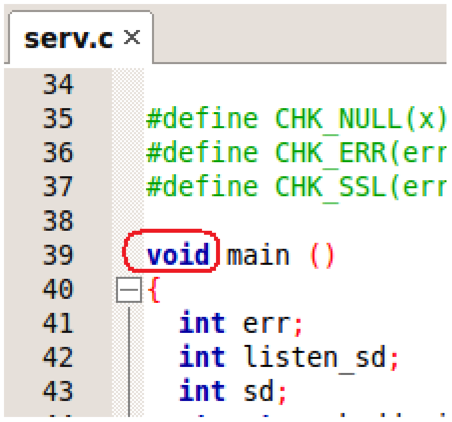


Add **/lib** to the list. This is the directory on the OmniFlash where shared libraries live.

Now click OK to save all the settings.

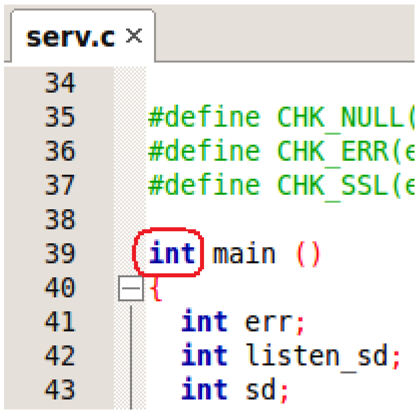
Code changes to make it compile

There are a couple code changes we need to make in order to get a clean compile.



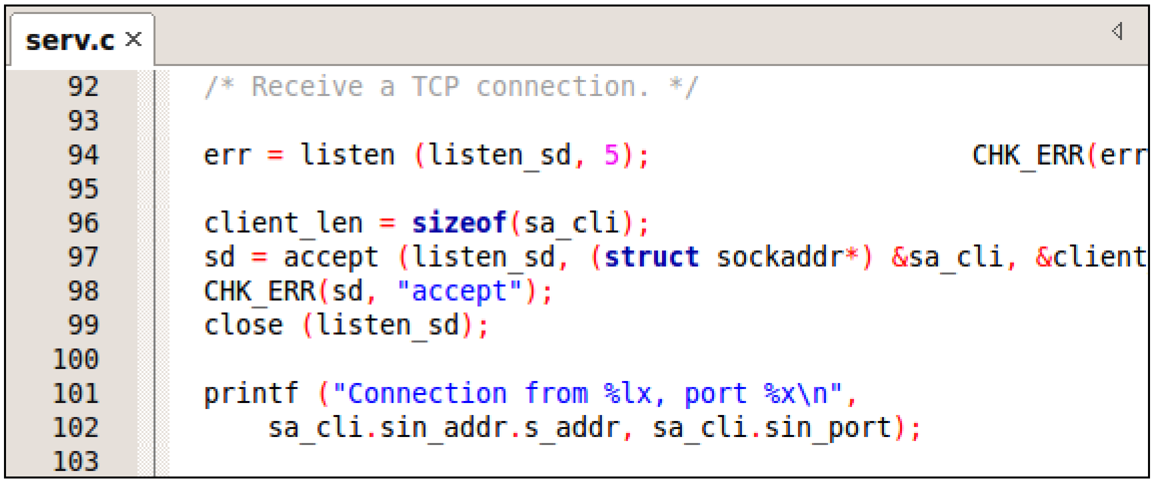
```
#define CHK_NULL(x)
#define CHK_ERR(err
#define CHK_SSL(err

void main ()
{
  int err;
  int listen_sd;
  int sd;
```

TO



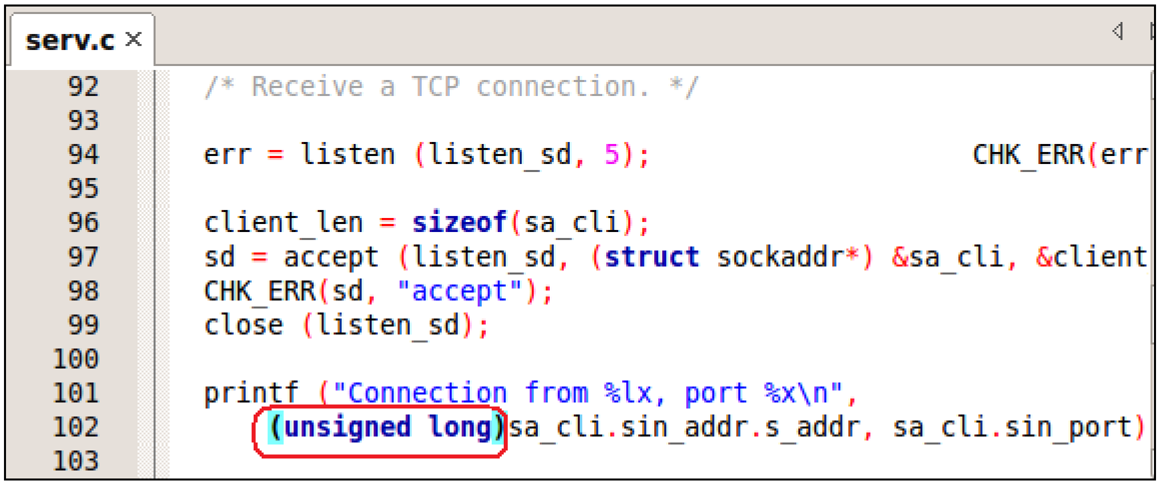
```
#define CHK_NULL(
#define CHK_ERR(e
#define CHK_SSL(e

int main ()
{
  int err;
  int listen_sd;
  int sd;
```

Change the function main type from void to int.



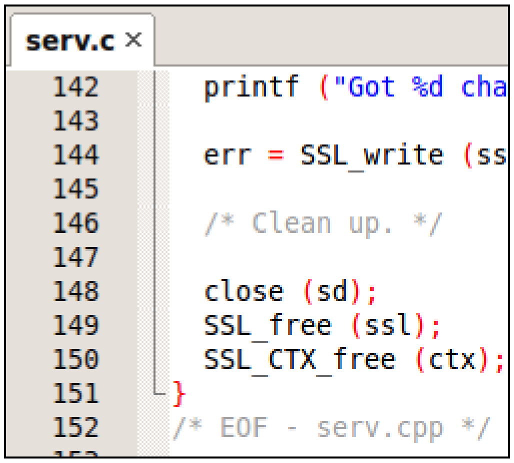
```
/* Receive a TCP connection. */

err = listen (listen_sd, 5);                   CHK_ERR(err

client_len = sizeof(sa_cli);
sd = accept (listen_sd, (struct sockaddr*) &sa_cli, &client
CHK_ERR(sd, "accept");
close (listen_sd);

printf ("Connection from %lx, port %x\n",
    sa_cli.sin_addr.s_addr, sa_cli.sin_port);
```

Change TO



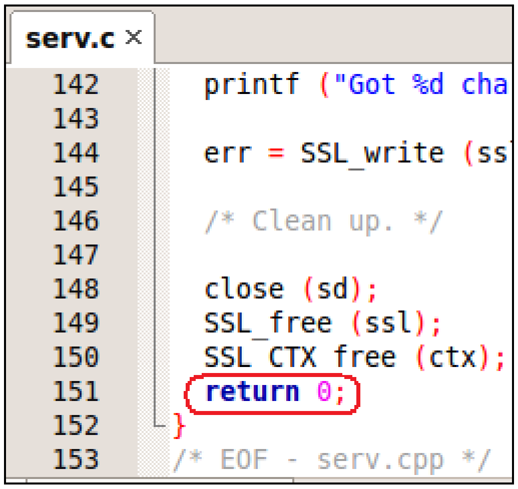
```
/* Receive a TCP connection. */

err = listen (listen_sd, 5);                    CHK_ERR(err

client_len = sizeof(sa_cli);
sd = accept (listen_sd, (struct sockaddr*) &sa_cli, &client
CHK_ERR(sd, "accept");
close (listen_sd);

printf ("Connection from %lx, port %x\n",
    (unsigned long)sa_cli.sin_addr.s_addr, sa_cli.sin_port)
```

Type cast s_addr to an unsigned long.



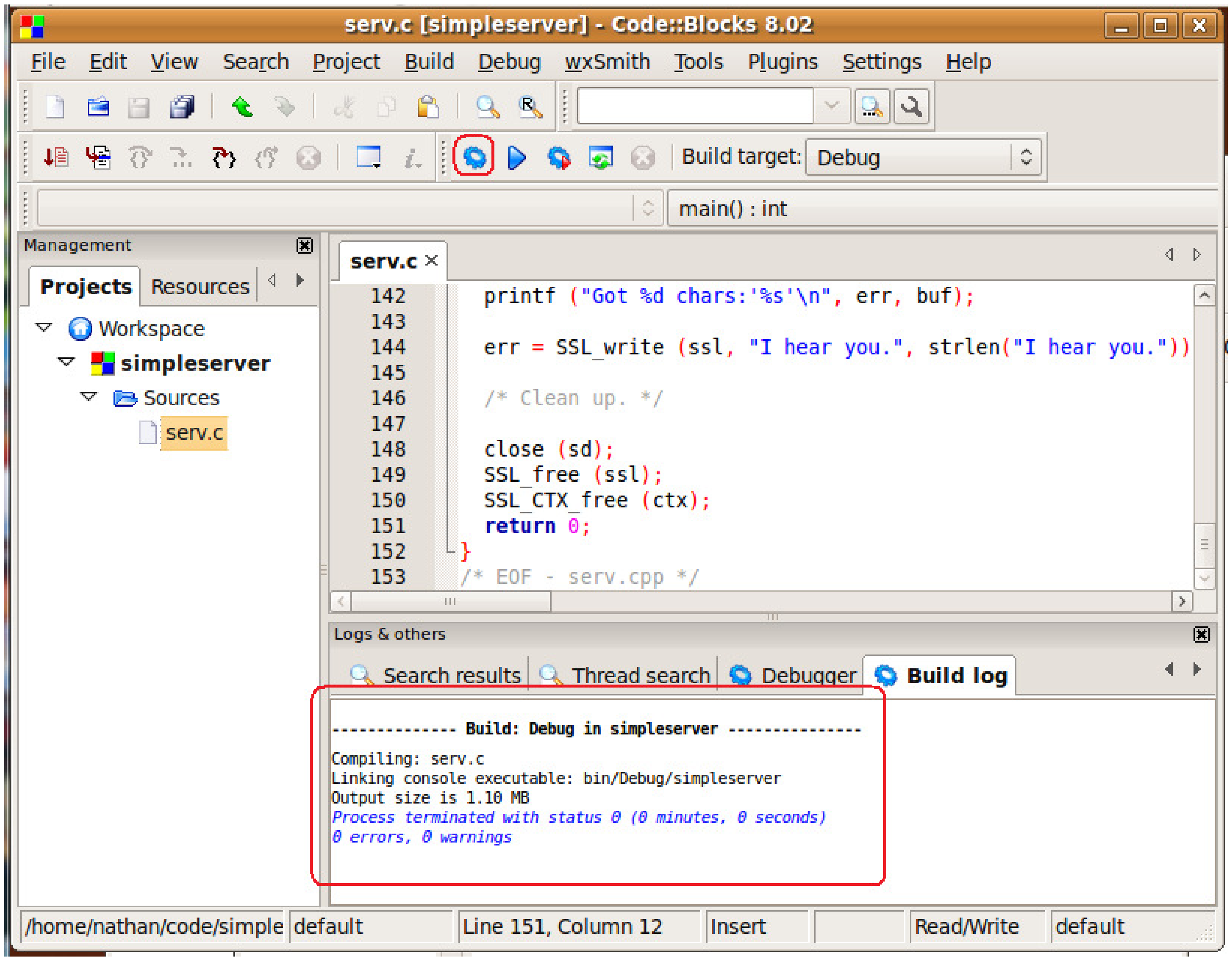
```
printf ("Got %d cha

err = SSL_write (ss

/* Clean up. */

close (sd);
SSL_free (ssl);
SSL_CTX_free (ctx);
}
/* EOF - serv.cpp */
```

TO



```
printf ("Got %d cha

err = SSL_write (ss

/* Clean up. */

close (sd);
SSL_free (ssl);
SSL_CTX_free (ctx);
return 0;
}
/* EOF - serv.cpp */
```

Add a return 0 to the end of function main.

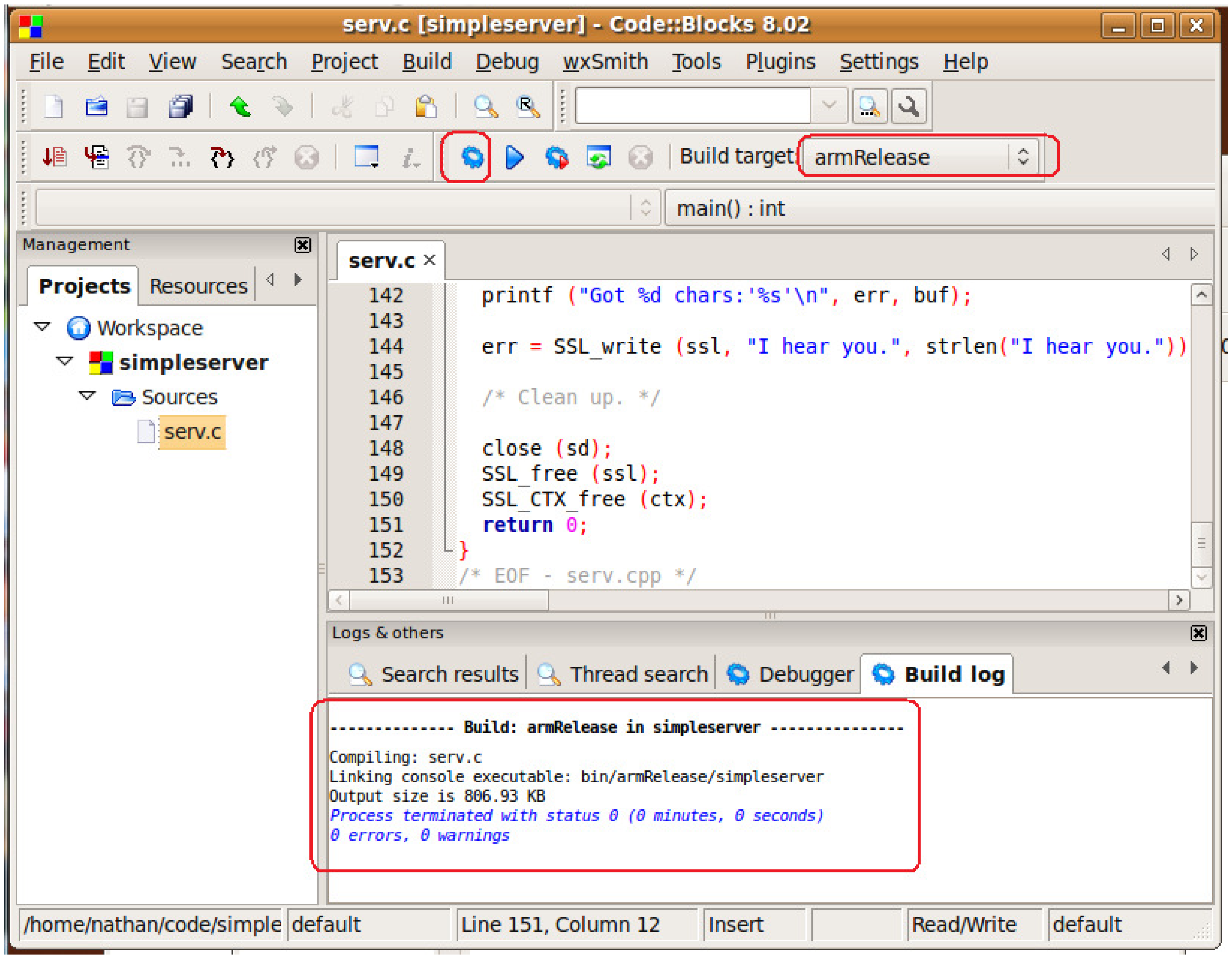

serv.c [simpleserver] - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
main() : int
Management
Projects Resources
Workspace
simpleserver
Sources
serv.c
Logs & others
Search results Thread search Debugger Build log
/home/nathan/code/simple default Line 151, Column 12 Insert Read/Write default

```
    printf ("Got %d chars:'%s'\n", err, buf);

    err = SSL_write (ssl, "I hear you.", strlen("I hear you."))

    /* Clean up. */

    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);
    return 0;
}
/* EOF - serv.cpp */
```

```
-------------- Build: Debug in simpleserver ---------------
Compiling: serv.c
Linking console executable: bin/Debug/simpleserver
Output size is 1.10 MB
Process terminated with status 0 (0 minutes, 0 seconds)
0 errors, 0 warnings
```

Now click the compile button. You should get a successful compile.
This program will run on our Ubuntu Linux box now. We could run this as a server and also run our client we built earlier and we would have a working Client / Server set of programs.

Let's try compiling this for ARM next.


serv.c [simpleserver] - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: armRelease
main() : int
Management
Projects Resources
Workspace
simpleserver
Sources
serv.c
Logs & others
Search results Thread search Debugger Build log
/home/nathan/code/simple default Line 151, Column 12 Insert Read/Write default

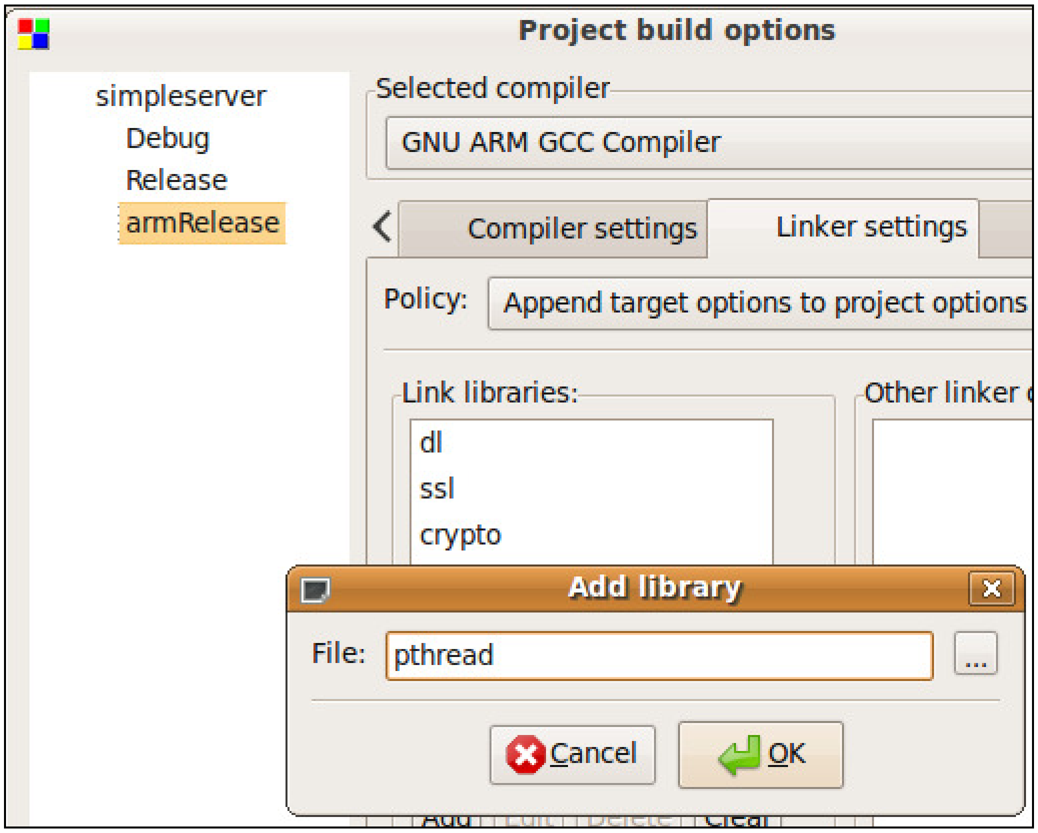
```
    printf ("Got %d chars:'%s'\n", err, buf);

    err = SSL_write (ssl, "I hear you.", strlen("I hear you."))

    /* Clean up. */

    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);
    return 0;
}
/* EOF - serv.cpp */
```

```
-------------- Build: armRelease in simpleserver ---------------
Compiling: serv.c
Linking console executable: bin/armRelease/simpleserver
Output size is 806.93 KB
Process terminated with status 0 (0 minutes, 0 seconds)
0 errors, 0 warnings
```

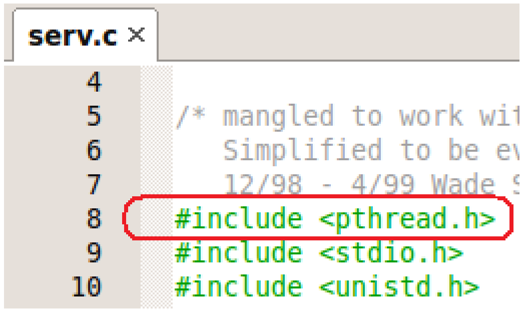
Change the Build target to armRelease and click the compile button. You should get a successful compile.

Multithreaded support

If you are going to make a threaded application, you need to add one more library to the list.



Add **pthread** to each target.

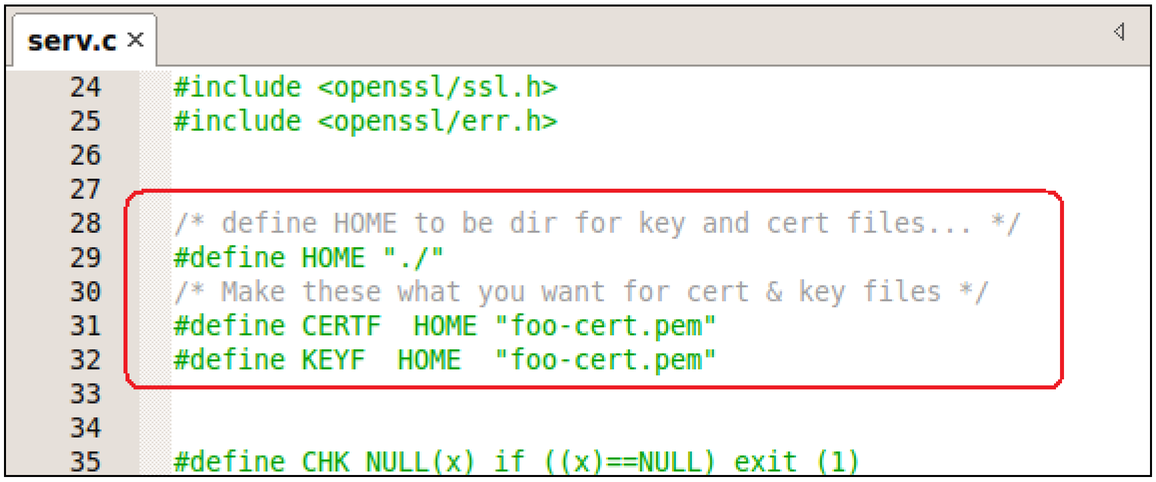


```
/* mangled to work wit
   Simplified to be ev
   12/98 - 4/99 Wade S
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
```

You must also **#include <pthread.h>** as the first include. By including it first, different options are enabled in the standard runtime library includes.

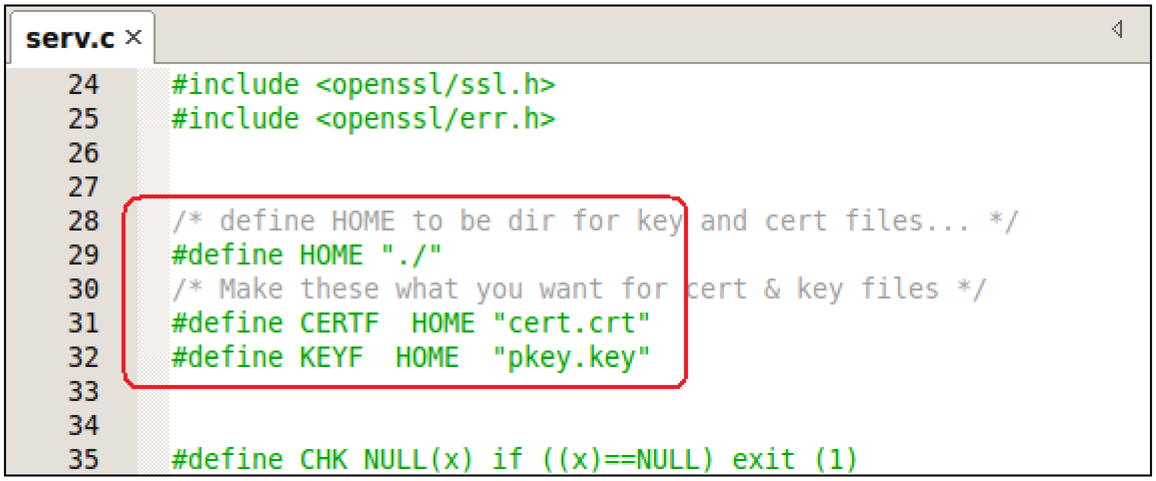
Generating an SSL certificate and Key

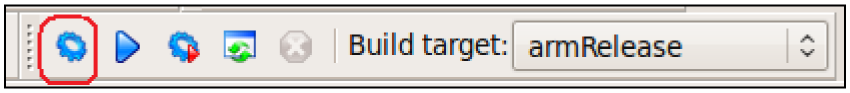
Before we can test our program, we need to generate an SSL certificate and key. We also need to tell our server program what the names of the keys are and where to find them.


```
#include <openssl/ssl.h>
#include <openssl/err.h>


/* define HOME to be dir for key and cert files... */
#define HOME "./"
/* Make these what you want for cert & key files */
#define CERTF  HOME "foo-cert.pem"
#define KEYF  HOME  "foo-cert.pem"


#define CHK_NULL(x) if ((x)==NULL) exit (1)
```

Change the names of the CERTF and KEYF defines to the following:

TO

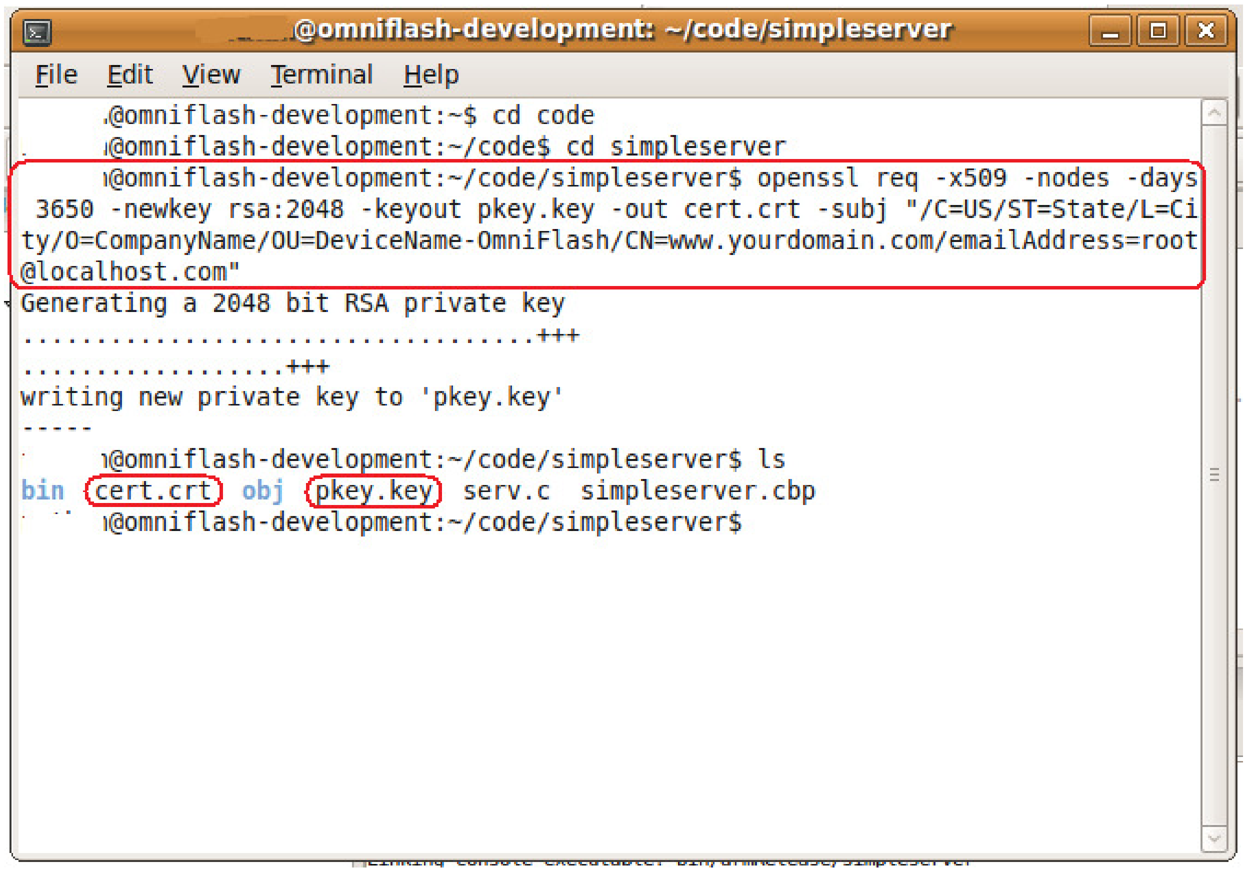

```
#include <openssl/ssl.h>
#include <openssl/err.h>


/* define HOME to be dir for key and cert files... */
#define HOME "./"
/* Make these what you want for cert & key files */
#define CERTF  HOME "cert.crt"
#define KEYF  HOME  "pkey.key"


#define CHK_NULL(x) if ((x)==NULL) exit (1)
```



Now compile the program for ARM.

Open a terminal and type this command. Note: This goes all on one line. Change the fields to match your specific needs.

```
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout pkey.key -out cert.crt -subj
"/C=US/ST=State/L=City/O=CompanyName/OU=DeviceName-
OmniFlash/CN=www.yourdomain.com/emailAddress=root@localhost.com"
```



```
@omniflash-development:~$ cd code
@omniflash-development:~/code$ cd simpleserver
@omniflash-development:~/code/simpleserver$ openssl req -x509 -nodes -days
 3650 -newkey rsa:2048 -keyout pkey.key -out cert.crt -subj "/C=US/ST=State/L=Ci
ty/O=CompanyName/OU=DeviceName-OmniFlash/CN=www.yourdomain.com/emailAddress=root
@localhost.com"
Generating a 2048 bit RSA private key
...................................+++
..................+++
writing new private key to 'pkey.key'
-----
@omniflash-development:~/code/simpleserver$ ls
bin  cert.crt  obj  pkey.key  serv.c  simpleserver.cbp
@omniflash-development:~/code/simpleserver$
```

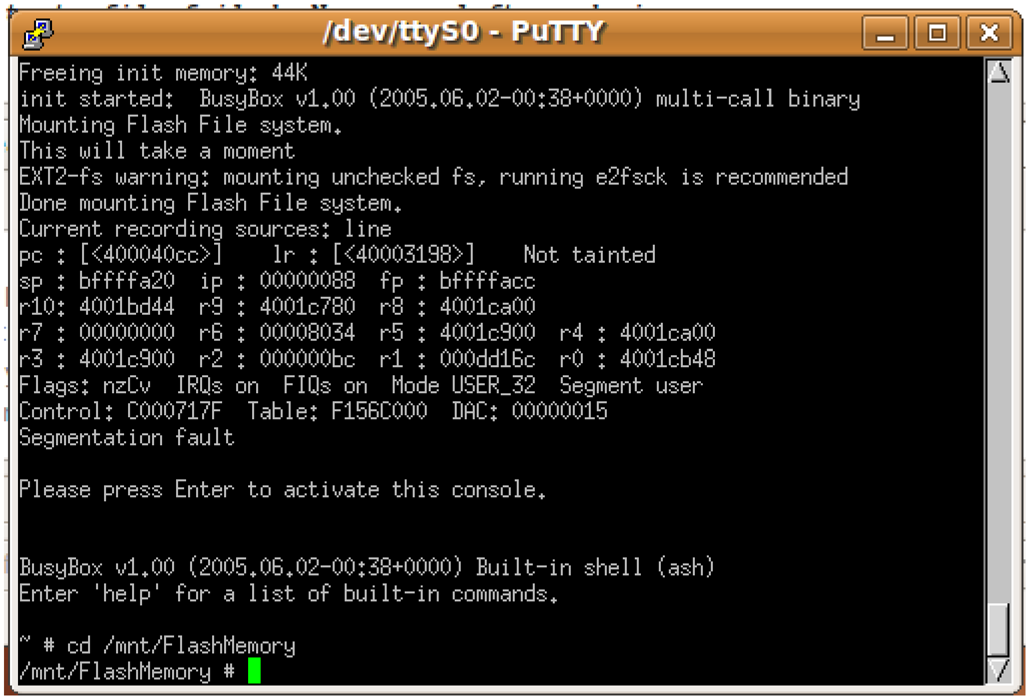
Once we call openssl, we end up with our certificate and key files. We need to copy these to the OmniFlash along with our program.

Copying files to the OmniFlash

In order to test our program, we need to copy over a few files to the OmniFlash.

Note: For this test, we must write to **/mnt/FlashMemory** because of the size of all the files we have.

Start PuTTY and boot the system.

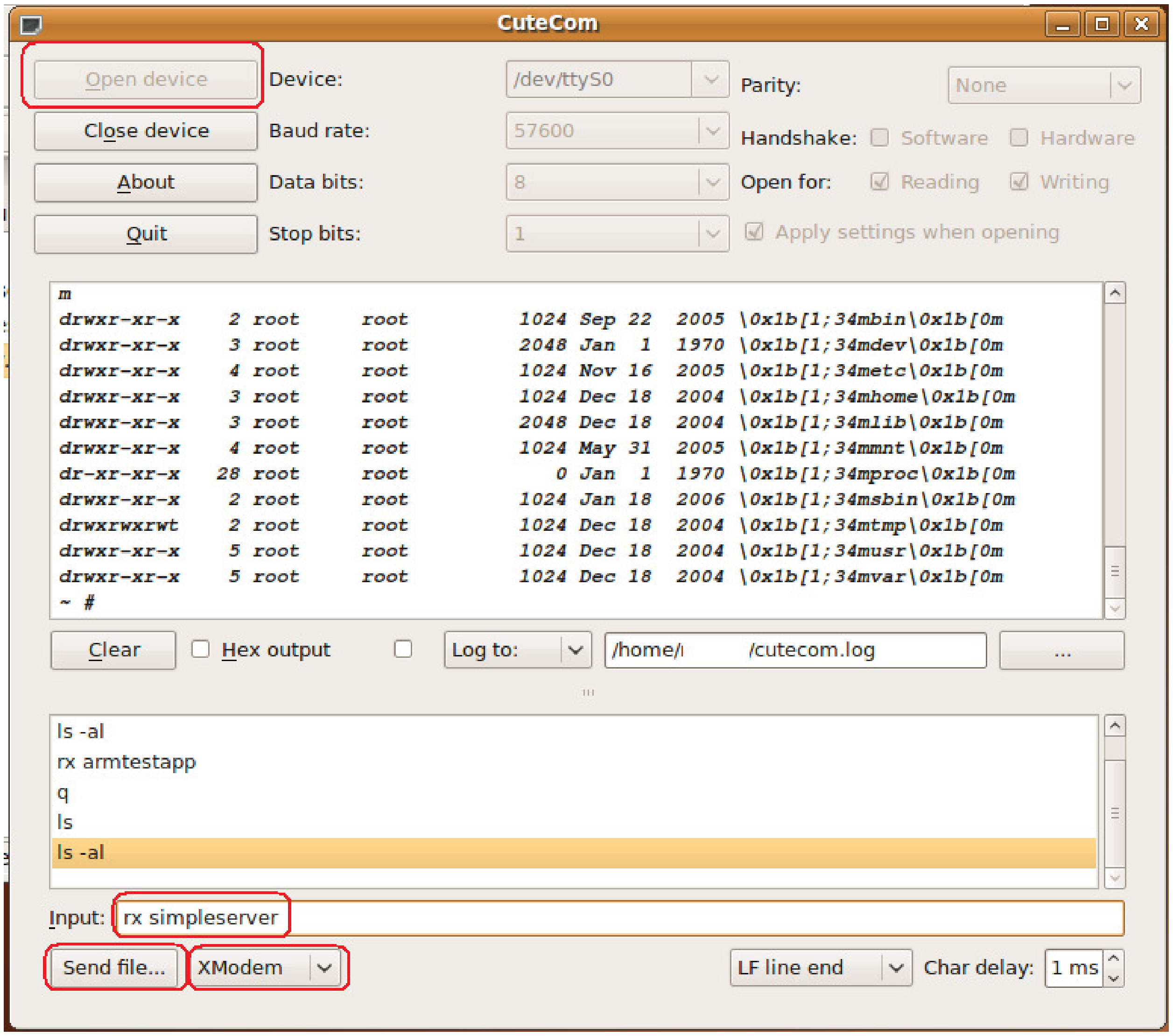


```
Freeing init memory: 44K
init started:  BusyBox v1.00 (2005.06.02-00:38+0000) multi-call binary
Mounting Flash File system.
This will take a moment
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
Done mounting Flash File system.
Current recording sources: line
pc : [<400040cc>]    lr : [<40003198>]    Not tainted
sp : bffffa20  ip : 00000088  fp : bffffacc
r10: 4001bd44  r9 : 4001c780  r8 : 4001ca00
r7 : 00000000  r6 : 00008034  r5 : 4001c900  r4 : 4001ca00
r3 : 4001c900  r2 : 000000bc  r1 : 000dd16c  r0 : 4001cb48
Flags: nzCv  IRQs on  FIQs on  Mode USER_32  Segment user
Control: C000717F  Table: F156C000  DAC: 00000015
Segmentation fault

Please press Enter to activate this console.


BusyBox v1.00 (2005.06.02-00:38+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # cd /mnt/FlashMemory
/mnt/FlashMemory #
```

Once booted, **cd** into **/mnt/FlashMemory**

Now that we are in the right directory, close PuTTy so the serial port is available again.

Start the **CuteCom** application and click the **Open Device** button.



Device: /dev/ttyS0
Baud rate: 57600
Data bits: 8
Stop bits: 1
Parity: None
Handshake: ☐ Software ☐ Hardware
Open for: ☑ Reading ☑ Writing
☑ Apply settings when opening

```
m
drwxr-xr-x    2 root      root         1024 Sep 22  2005 \0x1b[1;34mbin\0x1b[0m
drwxr-xr-x    3 root      root         2048 Jan  1  1970 \0x1b[1;34mdev\0x1b[0m
drwxr-xr-x    4 root      root         1024 Nov 16  2005 \0x1b[1;34metc\0x1b[0m
drwxr-xr-x    3 root      root         1024 Dec 18  2004 \0x1b[1;34mhome\0x1b[0m
drwxr-xr-x    3 root      root         2048 Dec 18  2004 \0x1b[1;34mlib\0x1b[0m
drwxr-xr-x    4 root      root         1024 May 31  2005 \0x1b[1;34mmnt\0x1b[0m
dr-xr-xr-x   28 root      root            0 Jan  1  1970 \0x1b[1;34mproc\0x1b[0m
drwxr-xr-x    2 root      root         1024 Jan 18  2006 \0x1b[1;34msbin\0x1b[0m
drwxrwxrwt    2 root      root         1024 Dec 18  2004 \0x1b[1;34mtmp\0x1b[0m
drwxr-xr-x    5 root      root         1024 Dec 18  2004 \0x1b[1;34musr\0x1b[0m
drwxr-xr-x    5 root      root         1024 Dec 18  2004 \0x1b[1;34mvar\0x1b[0m
~ #
```

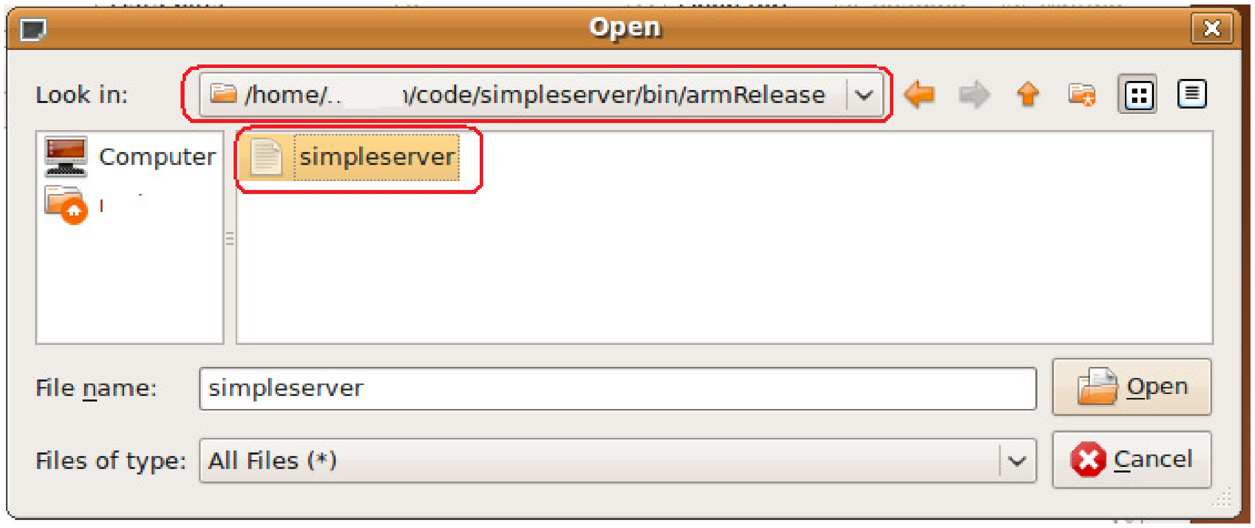
☐ Hex output ☐ Log to: /home/ /cutecom.log

```
ls -al
rx armtestapp
q
ls
ls -al
```

Input: rx simpleserver

XModem   LF line end   Char delay: 1 ms

After the OmniFlash has booted, we need to send the program we just compiled. Make sure XModem is selected. In the Input box, type **rx simpleserver** and press **enter**.. Then click the Send file... button. Note: Once we have entered a command, we can double-click it from the list and it will be sent for us.



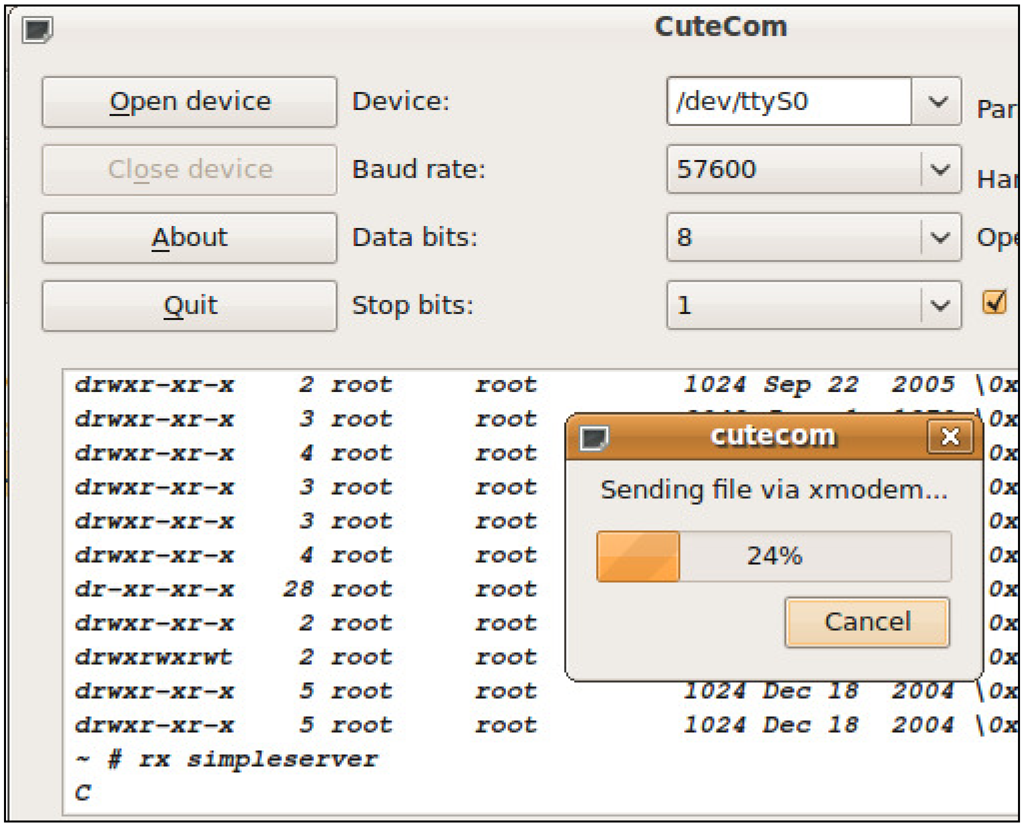
Look in: /home/.. ı/code/simpleserver/bin/armRelease

simpleserver

File name: simpleserver

Files of type: All Files (*)

Browse for the file we just compiled and click Open to send it.



Device: /dev/ttyS0

Baud rate: 57600

Data bits: 8

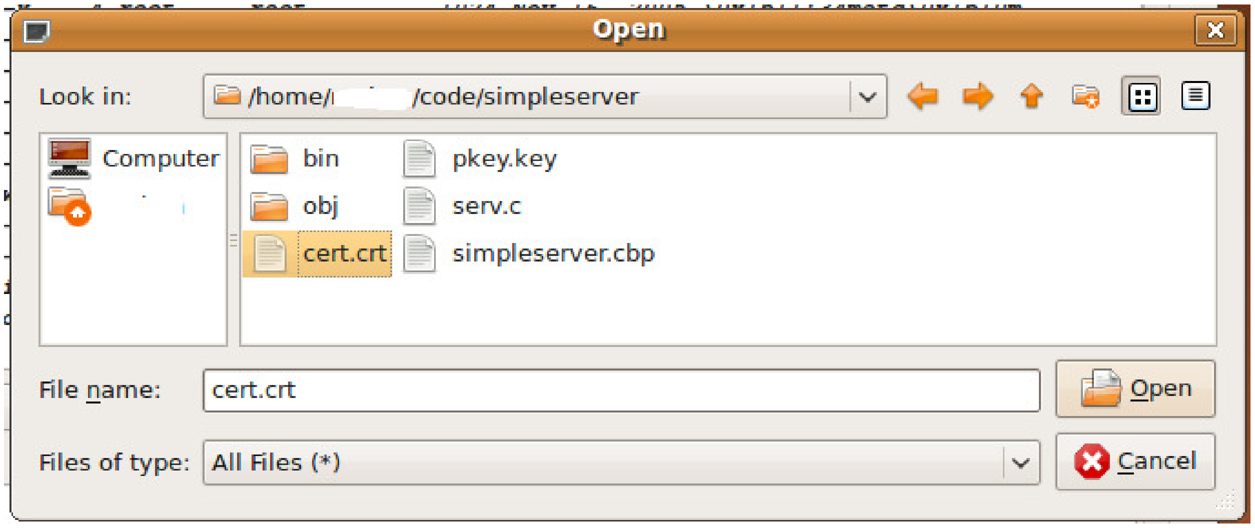
Stop bits: 1

```
drwxr-xr-x    2 root     root         1024 Sep 22  2005 \0x
drwxr-xr-x    3 root     root                            \0x
drwxr-xr-x    4 root     root                             0x
drwxr-xr-x    3 root     root                             0x
drwxr-xr-x    3 root     root                             0x
drwxr-xr-x    4 root     root                             0x
dr-xr-xr-x   28 root     root                             0x
drwxr-xr-x    2 root     root                             0x
drwxrwxrwt    2 root     root                             0x
drwxr-xr-x    5 root     root         1024 Dec 18  2004 \0x
drwxr-xr-x    5 root     root         1024 Dec 18  2004 \0x
~ # rx simpleserver
C
```

Sending file via xmodem...

24%

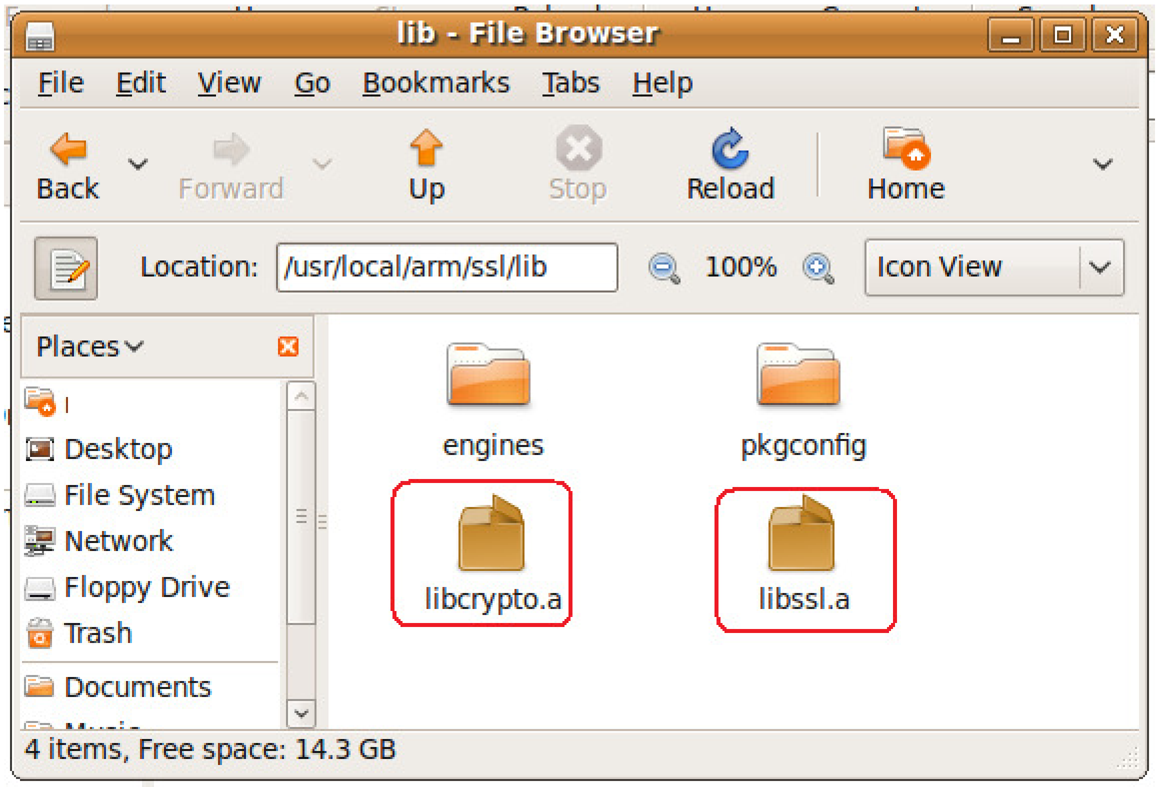
Wait while the file is being sent.



Do the same for the following files...

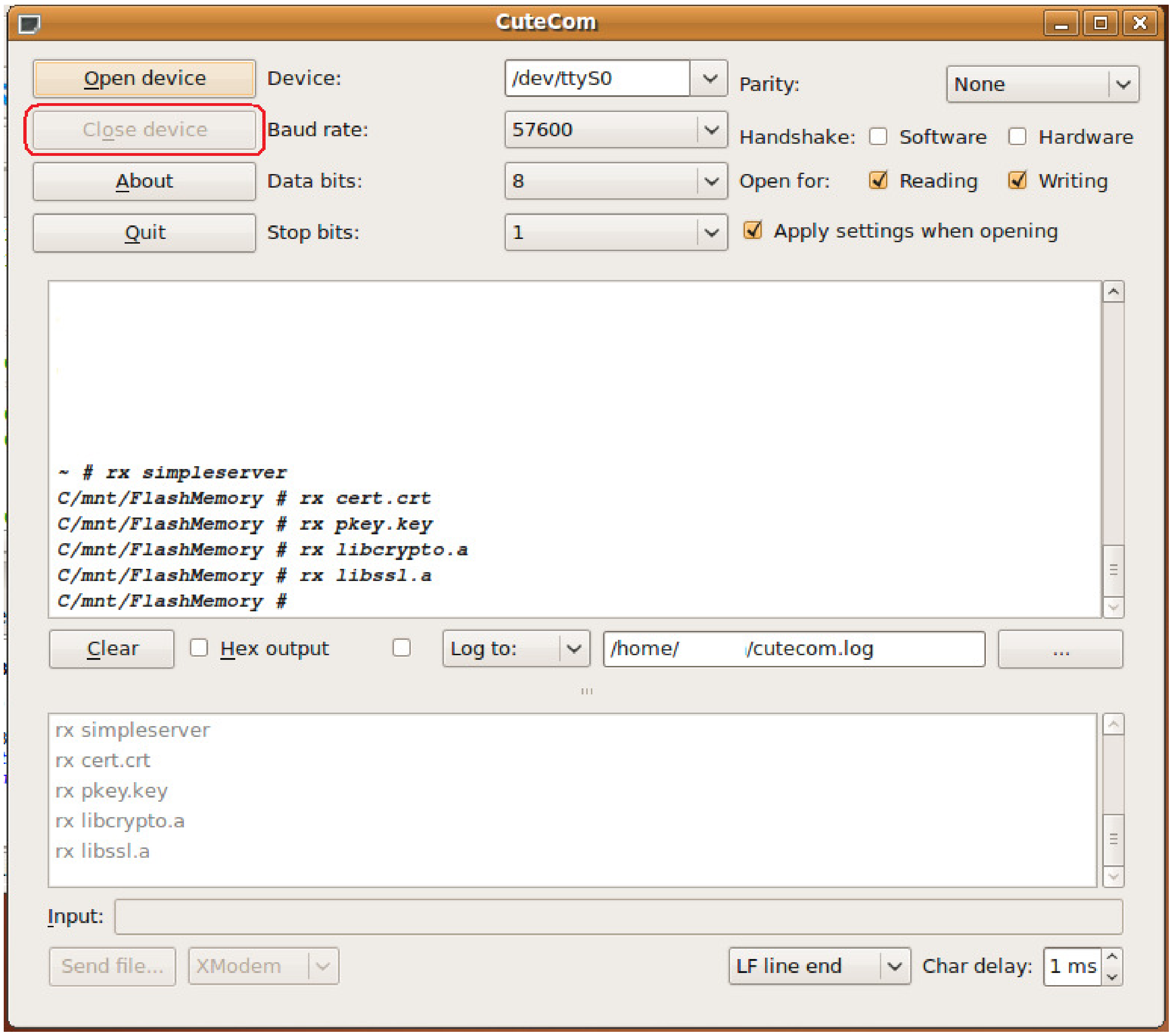
cert.crt and **pkey.key**

Now we must send the OpenSSL libraries.



The SSL libraries are in the folder we built earlier.

Using CuteCom, send these two libraries also.



Device: /dev/ttyS0
Baud rate: 57600
Data bits: 8
Stop bits: 1
Parity: None
Handshake: [ ] Software [ ] Hardware
Open for: [✓ Reading] [✓ Writing]
[✓ Apply settings when opening]

```
~ # rx simpleserver
C/mnt/FlashMemory # rx cert.crt
C/mnt/FlashMemory # rx pkey.key
C/mnt/FlashMemory # rx libcrypto.a
C/mnt/FlashMemory # rx libssl.a
C/mnt/FlashMemory #
```

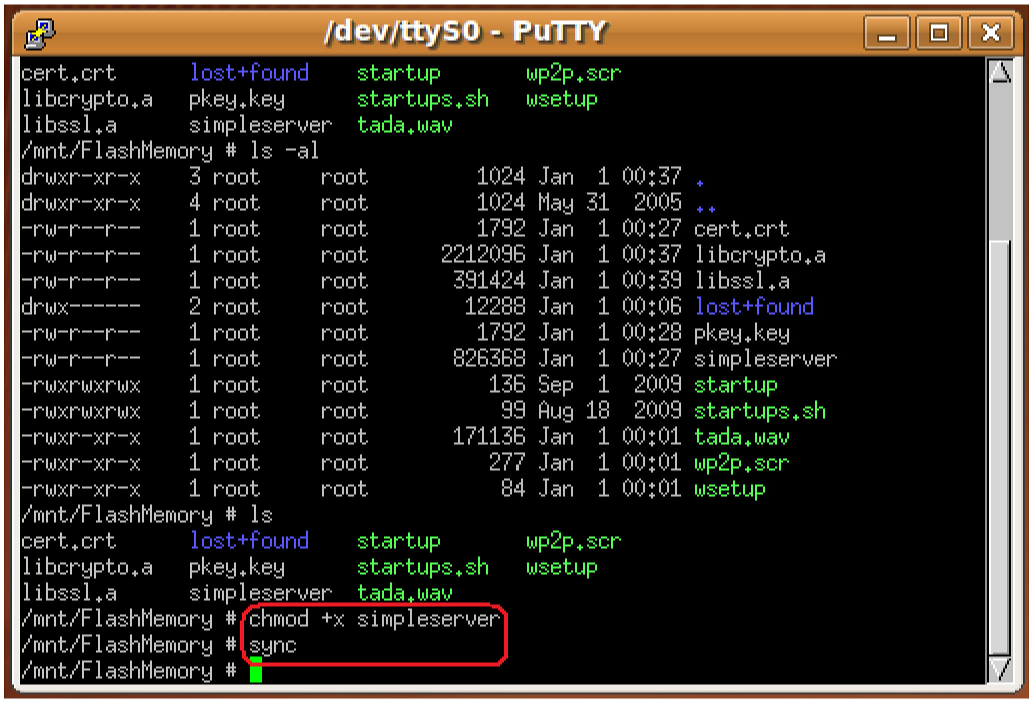
[ ] Hex output
Log to: /home/ /cutecom.log

```
rx simpleserver
rx cert.crt
rx pkey.key
rx libcrypto.a
rx libssl.a
```

Input:
LF line end
Char delay: 1 ms

Once all the files have been sent, click the Close device button.

Now start PuTTY again.



```
cert.crt       lost+found    startup       wp2p.scr
libcrypto.a    pkey.key      startups.sh   wsetup
libssl.a       simpleserver  tada.wav
/mnt/FlashMemory # ls -al
drwxr-xr-x    3 root     root         1024 Jan  1 00:37 .
drwxr-xr-x    4 root     root         1024 May 31  2005 ..
-rw-r--r--    1 root     root         1792 Jan  1 00:27 cert.crt
-rw-r--r--    1 root     root      2212096 Jan  1 00:37 libcrypto.a
-rw-r--r--    1 root     root       391424 Jan  1 00:39 libssl.a
drwx------    2 root     root        12288 Jan  1 00:06 lost+found
-rw-r--r--    1 root     root         1792 Jan  1 00:28 pkey.key
-rw-r--r--    1 root     root       826368 Jan  1 00:27 simpleserver
-rwxrwxrwx    1 root     root          136 Sep  1  2009 startup
-rwxrwxrwx    1 root     root           99 Aug 18  2009 startups.sh
-rwxr-xr-x    1 root     root       171136 Jan  1 00:01 tada.wav
-rwxr-xr-x    1 root     root          277 Jan  1 00:01 wp2p.scr
-rwxr-xr-x    1 root     root           84 Jan  1 00:01 wsetup
/mnt/FlashMemory # ls
cert.crt       lost+found    startup       wp2p.scr
libcrypto.a    pkey.key      startups.sh   wsetup
libssl.a       simpleserver  tada.wav
/mnt/FlashMemory # chmod +x simpleserver
/mnt/FlashMemory # sync
/mnt/FlashMemory #
```

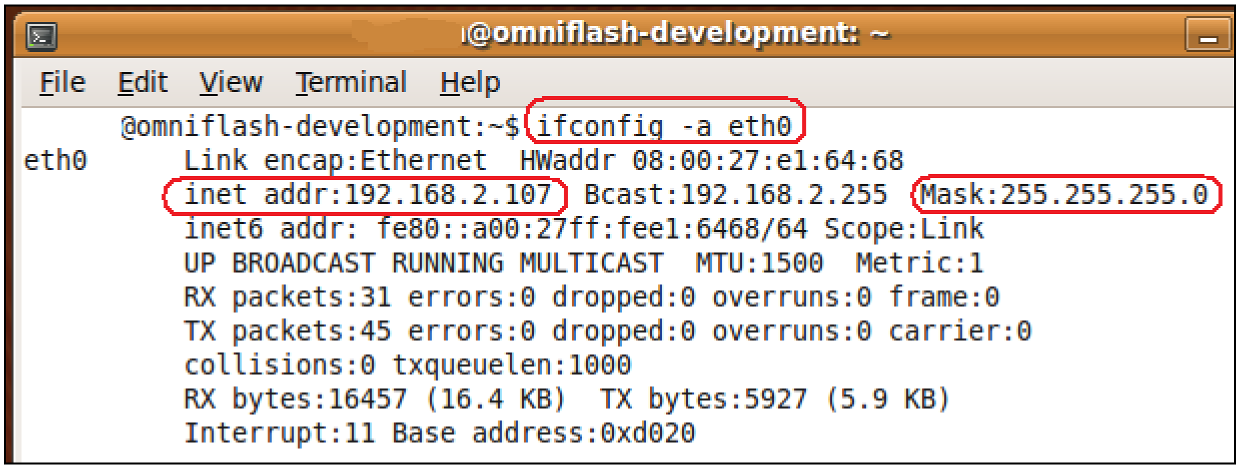
If we list the contents of the folder, we can see all the files we just sent.

We need to make the program executable. Do this by changing the mode to executable with **chmod +x simpleserver** . We follow that up with a **sync** to write the contents to the flash memory so that if we crash the system and have to reboot that we don't corrupt the flash memory.

Testing our server and client programs

In order to test our programs, we need to configure the OmniFlash's IP address and start up the ethernet device.

On our Ubuntu Linux device, we need to get our IP address.

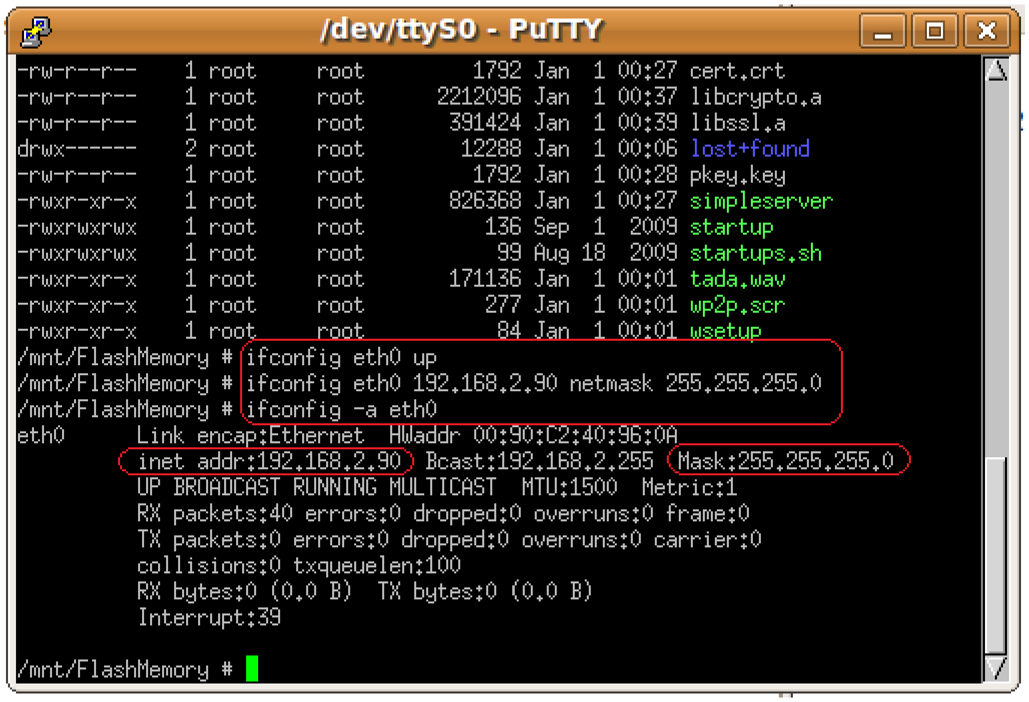

[redacted]@omniflash-development: ~
File Edit View Terminal Help

```
[redacted]@omniflash-development:~$ ifconfig -a eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:e1:64:68
          inet addr:192.168.2.107  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee1:6468/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:45 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16457 (16.4 KB)  TX bytes:5927 (5.9 KB)
          Interrupt:11 Base address:0xd020
```

We do this by bringing up a terminal and typing **ifconfig -a eth0** (or eth1 for whatever interface we have). Make note of **the inet addr** and the **Mask**. We need to set something similar on the OmniFlash.

Note: Make sure you have a network cable plugged into the OmniFlash.

In your PuTTY window, type the following commands.

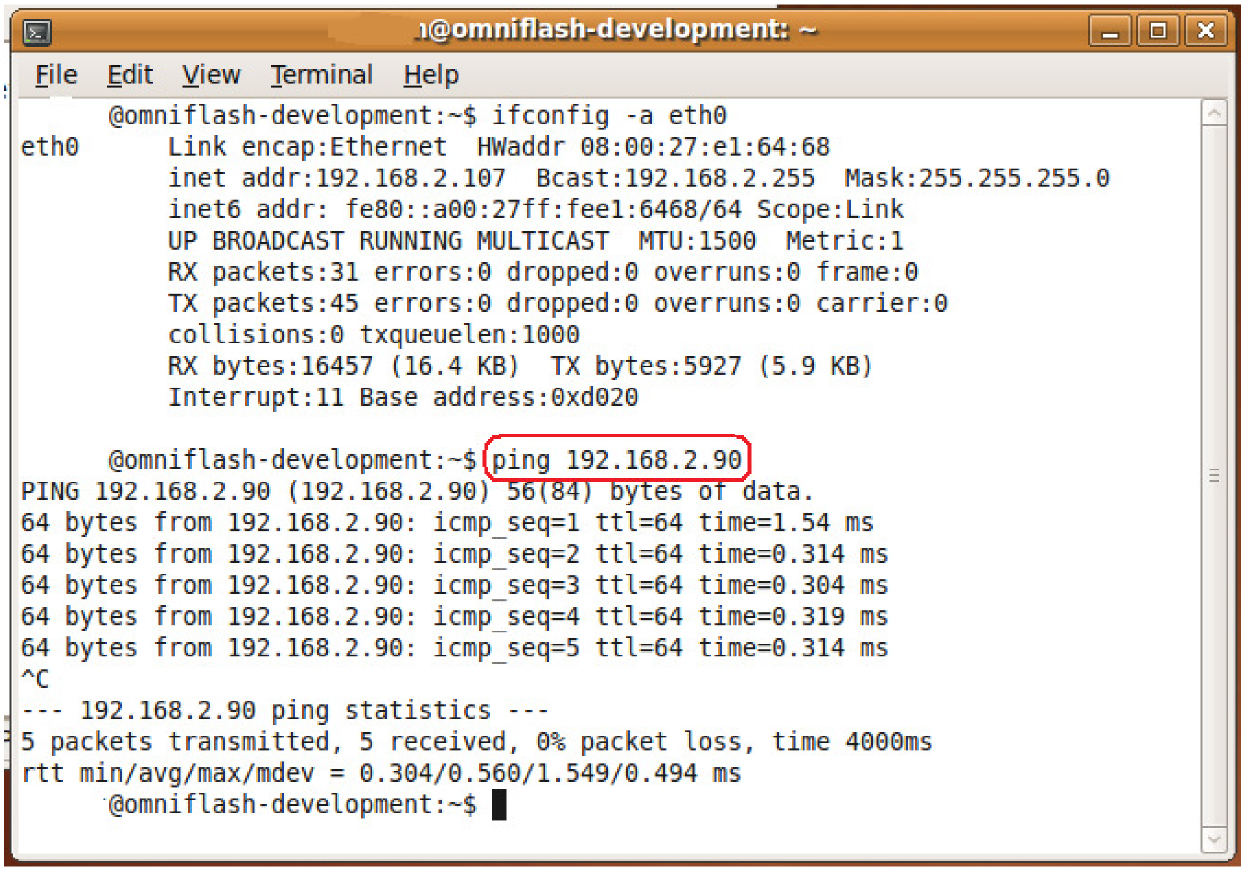


```
-rw-r--r--    1 root     root         1792 Jan  1 00:27 cert.crt
-rw-r--r--    1 root     root      2212096 Jan  1 00:37 libcrypto.a
-rw-r--r--    1 root     root       391424 Jan  1 00:39 libssl.a
drwx------    2 root     root        12288 Jan  1 00:06 lost+found
-rw-r--r--    1 root     root         1792 Jan  1 00:28 pkey.key
-rwxr-xr-x    1 root     root       826368 Jan  1 00:27 simpleserver
-rwxrwxrwx    1 root     root          136 Sep  1  2009 startup
-rwxrwxrwx    1 root     root           99 Aug 18  2009 startups.sh
-rwxr-xr-x    1 root     root       171136 Jan  1 00:01 tada.wav
-rwxr-xr-x    1 root     root          277 Jan  1 00:01 wp2p.scr
-rwxr-xr-x    1 root     root           84 Jan  1 00:01 wsetup
/mnt/FlashMemory # ifconfig eth0 up
/mnt/FlashMemory # ifconfig eth0 192.168.2.90 netmask 255.255.255.0
/mnt/FlashMemory # ifconfig -a eth0
eth0      Link encap:Ethernet  HWaddr 00:90:C2:40:96:0A
          inet addr:192.168.2.90  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:40 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:39

/mnt/FlashMemory #
```

ifconfig eth0 up Bring up the adapter.

ifconfig eth0 192.168.2.90 netmask 255.255.255.0 Set up your IP address.

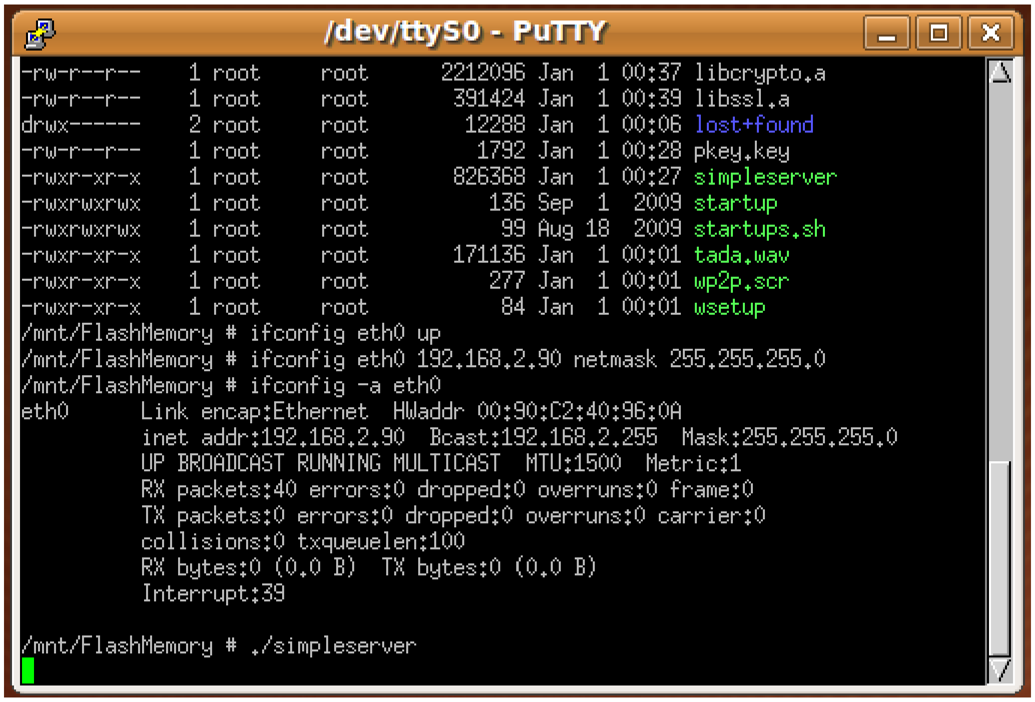
ifconfig -a eth0 List the IP address.



```
      @omniflash-development:~$ ifconfig -a eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:e1:64:68
          inet addr:192.168.2.107  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee1:6468/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:45 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16457 (16.4 KB)  TX bytes:5927 (5.9 KB)
          Interrupt:11 Base address:0xd020

      @omniflash-development:~$ ping 192.168.2.90
PING 192.168.2.90 (192.168.2.90) 56(84) bytes of data.
64 bytes from 192.168.2.90: icmp_seq=1 ttl=64 time=1.54 ms
64 bytes from 192.168.2.90: icmp_seq=2 ttl=64 time=0.314 ms
64 bytes from 192.168.2.90: icmp_seq=3 ttl=64 time=0.304 ms
64 bytes from 192.168.2.90: icmp_seq=4 ttl=64 time=0.319 ms
64 bytes from 192.168.2.90: icmp_seq=5 ttl=64 time=0.314 ms
^C
--- 192.168.2.90 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.304/0.560/1.549/0.494 ms
      @omniflash-development:~$ █
```

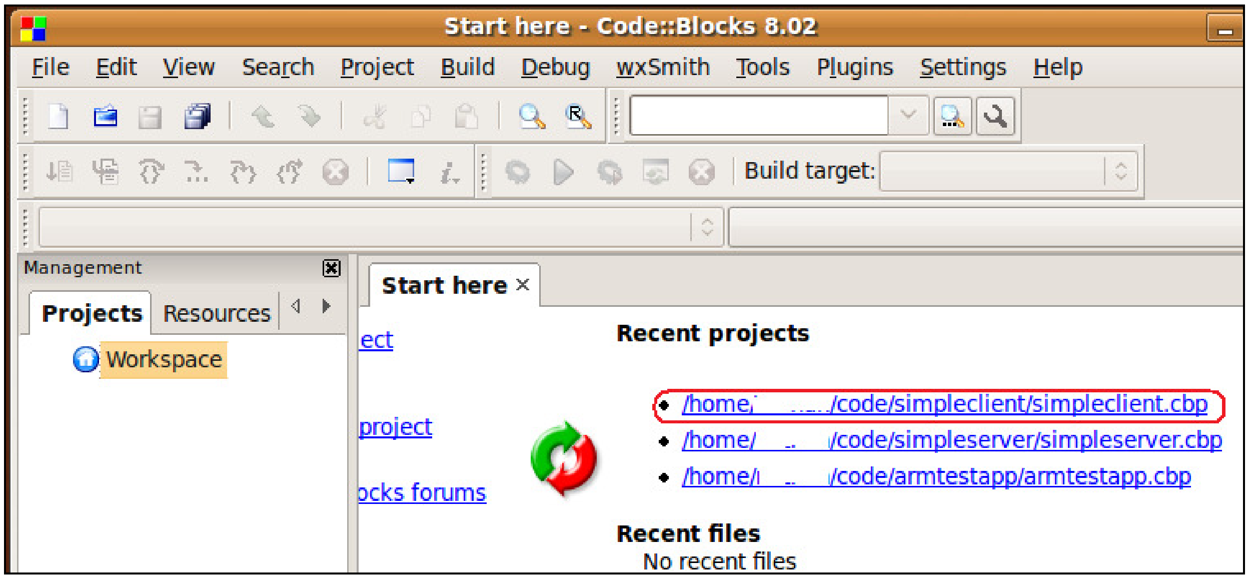
Now from our Ubuntu Linux box, we need to verify we can talk to the OmniFlash.
Ping the IP address you gave to the OmniFlash and verify you get a response. <ctrl> C gets you out.

Let's start up the server on the OmniFlash and verify we can talk to it.


```
-rw-r--r--    1 root     root      2212096 Jan  1 00:37 libcrypto.a
-rw-r--r--    1 root     root       391424 Jan  1 00:39 libssl.a
drwx------    2 root     root        12288 Jan  1 00:06 lost+found
-rw-r--r--    1 root     root         1792 Jan  1 00:28 pkey.key
-rwxr-xr-x    1 root     root       826368 Jan  1 00:27 simpleserver
-rwxrwxrwx    1 root     root          136 Sep  1  2009 startup
-rwxrwxrwx    1 root     root           99 Aug 18  2009 startups.sh
-rwxr-xr-x    1 root     root       171136 Jan  1 00:01 tada.wav
-rwxr-xr-x    1 root     root          277 Jan  1 00:01 wp2p.scr
-rwxr-xr-x    1 root     root           84 Jan  1 00:01 wsetup
/mnt/FlashMemory # ifconfig eth0 up
/mnt/FlashMemory # ifconfig eth0 192.168.2.90 netmask 255.255.255.0
/mnt/FlashMemory # ifconfig -a eth0
eth0      Link encap:Ethernet  HWaddr 00:90:C2:40:96:0A
          inet addr:192.168.2.90  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:40 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:39

/mnt/FlashMemory # ./simpleserver
```

So far so good.

Let's bring up the client program. Save all your work in CodeBlocks and open up the simpleclient program we built earlier.


Start here - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target:
Management
Projects Resources
Workspace
Start here
ect
project
ocks forums

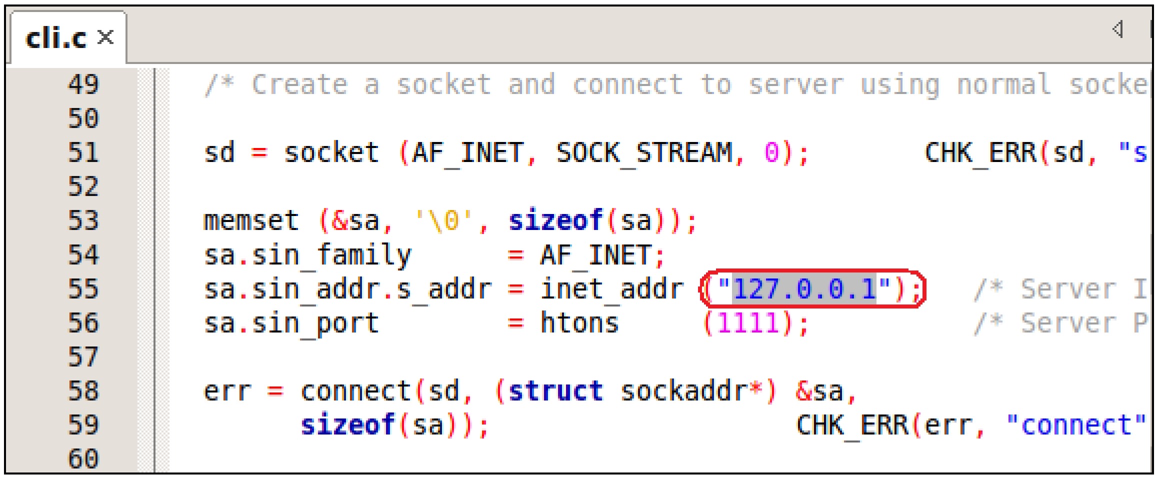
Recent projects

- /home/ ... /code/simpleclient/simpleclient.cbp
- /home/ .. /code/simpleserver/simpleserver.cbp
- /home/ .. /code/armtestapp/armtestapp.cbp

Recent files

No recent files

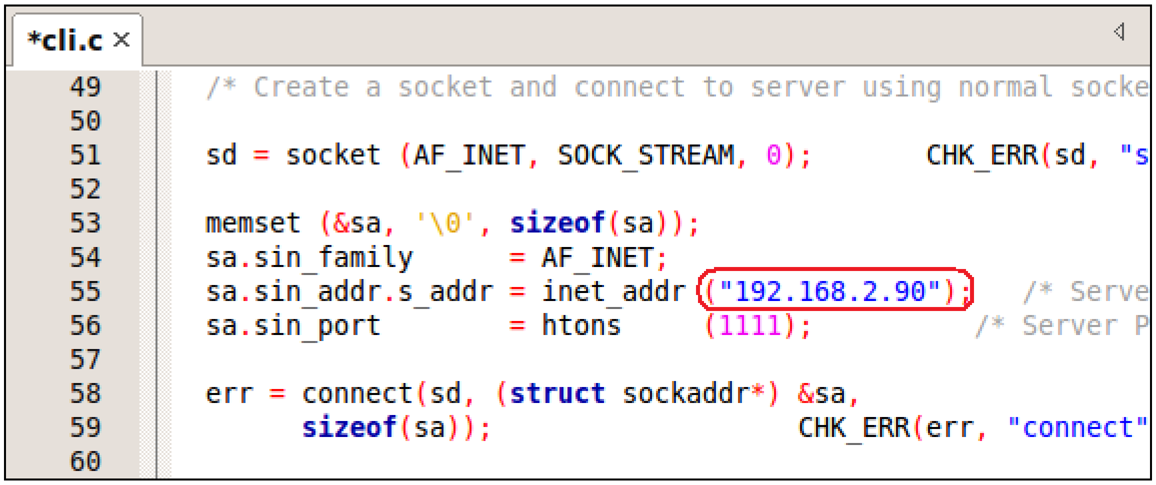
Open the simpleclient we built earlier.



cli.c

```
/* Create a socket and connect to server using normal socke

sd = socket (AF_INET, SOCK_STREAM, 0);        CHK_ERR(sd, "s

memset (&sa, '\0', sizeof(sa));
sa.sin_family      = AF_INET;
sa.sin_addr.s_addr = inet_addr ("127.0.0.1");     /* Server I
sa.sin_port        = htons     (1111);            /* Server P

err = connect(sd, (struct sockaddr*) &sa,
      sizeof(sa));                     CHK_ERR(err, "connect"
```

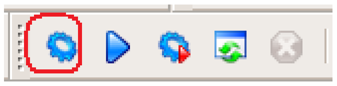
Change the default IP address to the IP address we gave the OmniFlash.



*cli.c

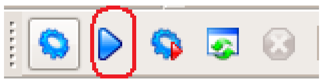
```
/* Create a socket and connect to server using normal socke

sd = socket (AF_INET, SOCK_STREAM, 0);        CHK_ERR(sd, "s

memset (&sa, '\0', sizeof(sa));
sa.sin_family      = AF_INET;
sa.sin_addr.s_addr = inet_addr ("192.168.2.90");     /* Serve
sa.sin_port        = htons     (1111);            /* Server P

err = connect(sd, (struct sockaddr*) &sa,
      sizeof(sa));                     CHK_ERR(err, "connect"
```

The IP address of the OmniFlash.

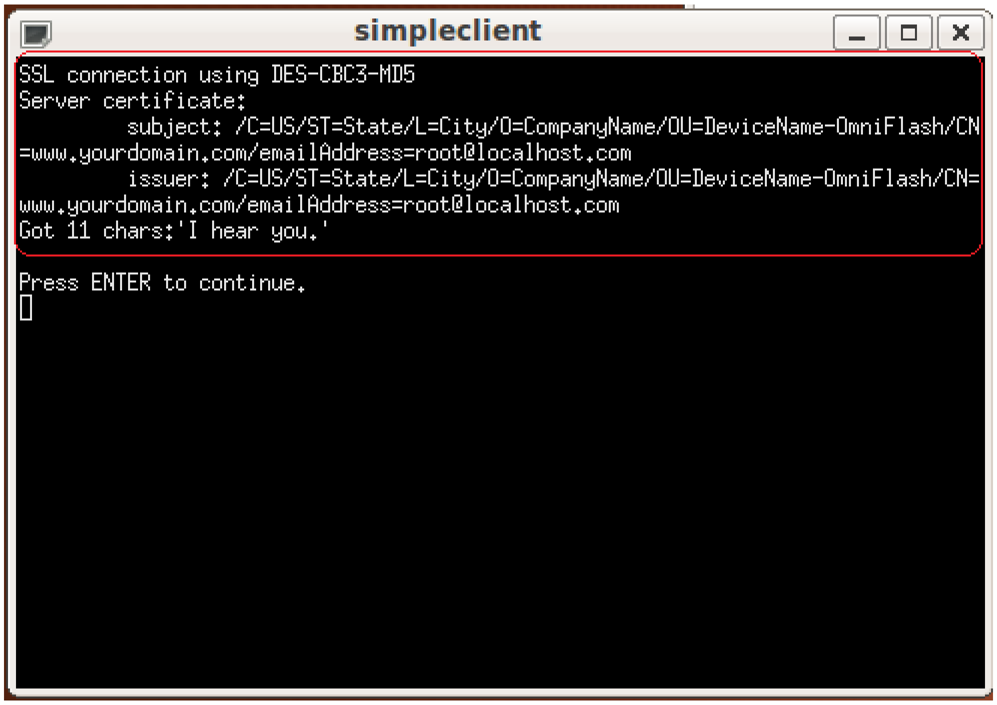


Compile the application again.

Now let's run it and see if they talk.

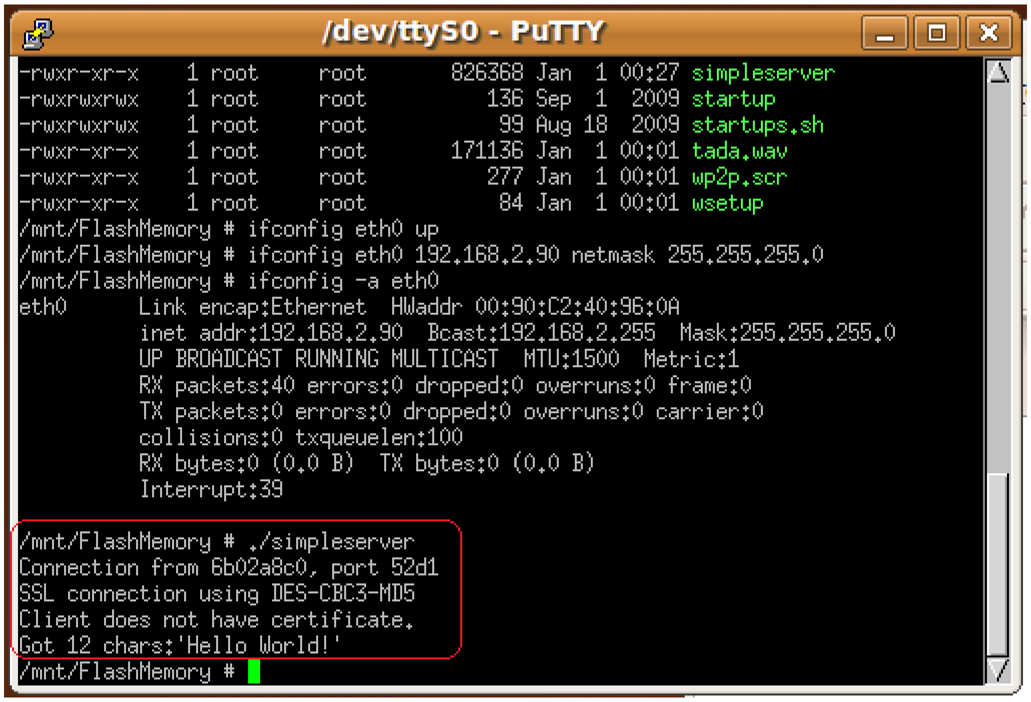


Click the Run button.



```
SSL connection using DES-CBC3-MD5
Server certificate:
         subject: /C=US/ST=State/L=City/O=CompanyName/OU=DeviceName-OmniFlash/CN
=www.yourdomain.com/emailAddress=root@localhost.com
         issuer: /C=US/ST=State/L=City/O=CompanyName/OU=DeviceName-OmniFlash/CN=
www.yourdomain.com/emailAddress=root@localhost.com
Got 11 chars:'I hear you.'

Press ENTER to continue.
```

Here is the output of our Client program.



```
-rwxr-xr-x    1 root     root       826368 Jan  1 00:27 simpleserver
-rwxrwxrwx    1 root     root          136 Sep  1  2009 startup
-rwxrwxrwx    1 root     root           99 Aug 18  2009 startups.sh
-rwxr-xr-x    1 root     root       171136 Jan  1 00:01 tada.wav
-rwxr-xr-x    1 root     root          277 Jan  1 00:01 wp2p.scr
-rwxr-xr-x    1 root     root           84 Jan  1 00:01 wsetup
/mnt/FlashMemory # ifconfig eth0 up
/mnt/FlashMemory # ifconfig eth0 192.168.2.90 netmask 255.255.255.0
/mnt/FlashMemory # ifconfig -a eth0
eth0      Link encap:Ethernet  HWaddr 00:90:C2:40:96:0A
          inet addr:192.168.2.90  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:40 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:39

/mnt/FlashMemory # ./simpleserver
Connection from 6b02a8c0, port 52d1
SSL connection using DES-CBC3-MD5
Client does not have certificate.
Got 12 chars:'Hello World!'
/mnt/FlashMemory #
```

Here is the output of our Server program. And there you have it. OpenSSL running on an OmniFlash.

If you wanted to debug the server program and test it, launch a new instance of CodeBlocks and open the Server project. (CodeBlocks needs a configuration change to allow more than one instance). Change the IP address back to 127.0.0.1 (This means local device), change the target to Debug and then debug it. When the kinks are worked out, recompile it for ARM and send it over to your OmniFlash and run it.