# BCM963XX
# Linux Software Build Guide

## Version 3.0

16215 Alton Parkway • P.O. Box 57013 • Irvine, CA 92619-7013 USA • Phone: (949) 450-8700 • Fax: (949) 450-8710

# *Table of Contents*

# *BCM963XX Linux Software Build Guide*

## REVISION HISTORY

| *Revision Number* | *Date* | *Change Description* |
|---|---|---|
| V3.0 | 12/28/04 | Updated source code tree according to 3.0 release with 2.6 Linux kernel |
| V2.0 | 05/01/03 | Changed screen captures in Section 3.2 and 4.2 for new applications in Release 2.10 |
| V1.0 | 11/18/02 | Initial release. |

## 1.0 INTRODUCTION

This document provides detailed information on the software build framework for the BCM963xx xDSL router or IAD device reference design platforms. Developers may use the framework to build a software image including both Linux kernel and user applications. Under the framework, developers have the flexibility to choose a variety of configurations as well as software components for different platforms such as the BCM96345R or BCM96345GW etc. tailored to their specific needs and feature set. For example, if a developer does not need firewall or NAT, they can exclude it from the build configurations to obtain a smaller image size. The file specifying build configurations for a hardware platform is called a *platform build profile*. This framework is generic and designed with scalability, meaning future platforms can be added with little effort.



## 2.0 THE ARCHITECTURE OF BUILD FRAMEWORK

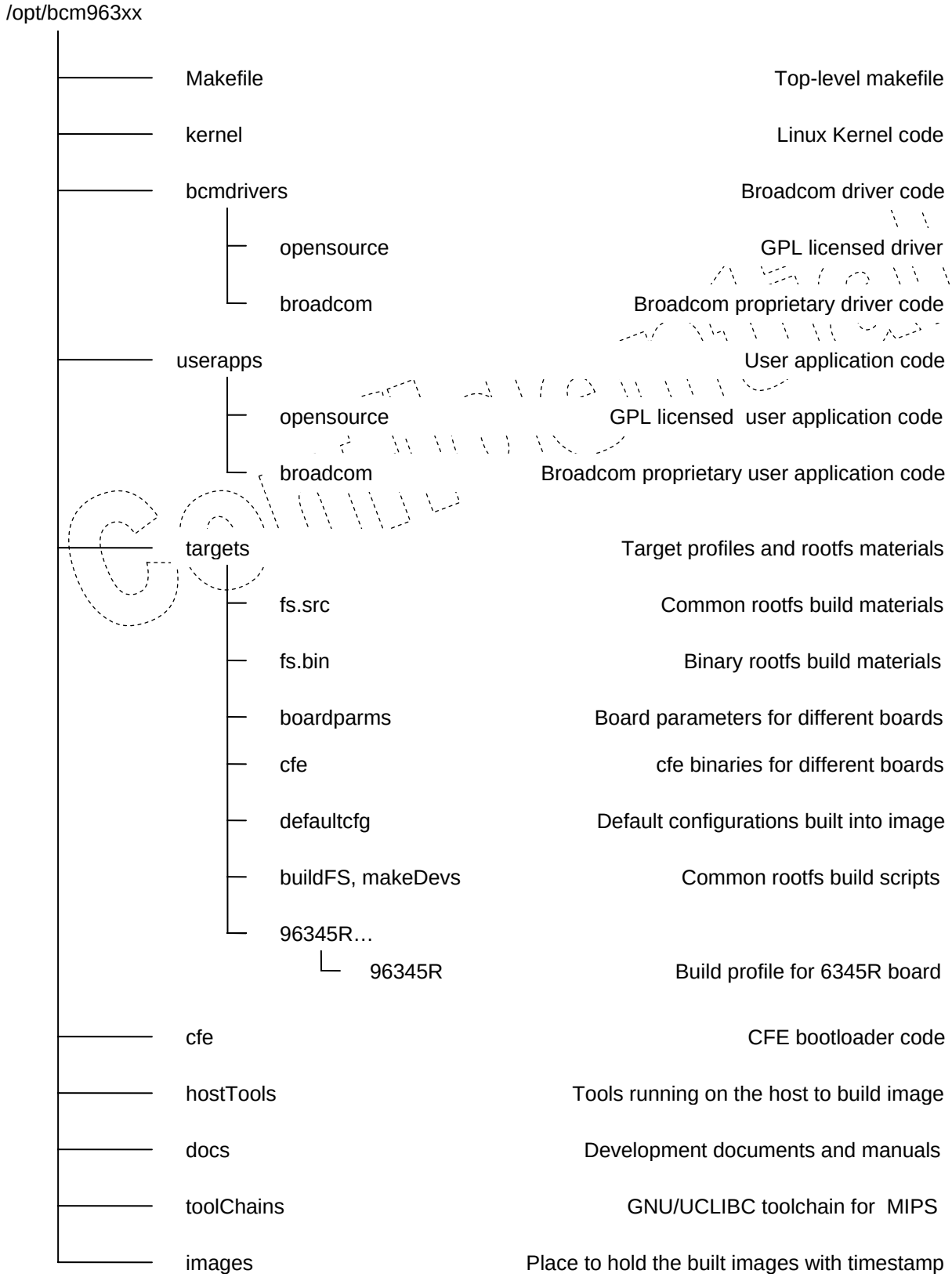
In the framework, kernel, drivers, user applications and configuration files are separated into different directories. The top-level Makefile controls the whole build process according to the chosen profile of a build. The next page shows the whole source code structure.

*kernel* directory is the Linux kernel source directory as well as board support package (BSP) for BCM963XX boards. 3.0 release uses 2.6.8.1 Linux kernel.

*bcmdrivers* directory includes both Broadcom proprietary and GPL licensed driver code. o*pensource* sub-directory contains GPL licensed code. *broadcom* sub-directory contains the proprietary code.

*userapps* directory is the user space application source directory. o*pensource* sub-directory contains GPL licensed code from the public domain. *broadcom* sub-directory contains the proprietary code developed by Broadcom.

*targets* directory is the place to hold different build profiles for different hardware platforms. Several default profiles are included in this directory such as 96345R, 96348GW etc. Developers can create their own profiles in this directory using a GUI-based tool. Each profile corresponds to a sub-directory in the *targets* directory. The sub-directory must have the same name as that of the profile. The final target root file system and flash images generated by the build process for a chosen hardware platform are also included in the sub-directory.


/opt/bcm963xx
Makefile
Top-level makefile
kernel
Linux Kernel code
bcmdrivers
Broadcom driver code
opensource
GPL licensed driver
broadcom
Broadcom proprietary driver code
userapps
User application code
opensource
GPL licensed user application code
broadcom
Broadcom proprietary user application code
targets
Target profiles and rootfs materials
fs.src
Common rootfs build materials
fs.bin
Binary rootfs build materials
boardparms
Board parameters for different boards
cfe
cfe binaries for different boards
defaultcfg
Default configurations built into image
buildFS, makeDevs
Common rootfs build scripts
96345R…
96345R
Build profile for 6345R board
cfe
CFE bootloader code
hostTools
Tools running on the host to build image
docs
Development documents and manuals
toolChains
GNU/UCLIBC toolchain for MIPS
images
Place to hold the built images with timestamp

# 3.0 PLATFORM BUILD PROFILE

A platform build profile specifies various configuration items when building a software image for a board. It is a central and single place to create configurations for a build. When developers need to build a customized image, they need to create a build profile and specify configurations. Note that a hardware board may have any number of build profiles, with different configurations specified.

## 3.1 Supported configuration items

The value of each configuration item should be specified in the build profile. Current supported configuration items include (but not limited to):

- Chip ID (6345, 6348 and 6338)
- Target board flash size, flash block size, Number of MAC address and default config file
- ADSL standard (ANNEX_A, ANNEX_B, ANNEX_C or SADSL)
- Broadcom drivers such as ADSL, ATM, Ethernet, USB, Wireless or Voice
- User applications
- Toolchain such as compiler, C libraries or pthread support
- Kernel netfilter
    - Firewall
    - NAT
    - ALGs (Application Level Gateway)
        - Event Logging such as intrusion detection
        - Special module combination for remote access in PPP IP extension mode
- Kernel debugging
- Root file system type (SQUASHFS, CRAMFS or NFS)
- Phone/voice signaling (SIP or MGCP)
- SOAP for chip diagnostics



## 3.2 Use GUI to create a build profile

The build framework provides a GUI interface to make the configuration process easier.

Within a build profile, a form of “name=value” is used to specify a configuration item. Different values have different meanings.

For kernel drivers and modules:

- "m" means “build the driver as kernel module”
- "y" means “statically link with kernel”
- "# ... is not set" means “do not build the driver”

For user applications:

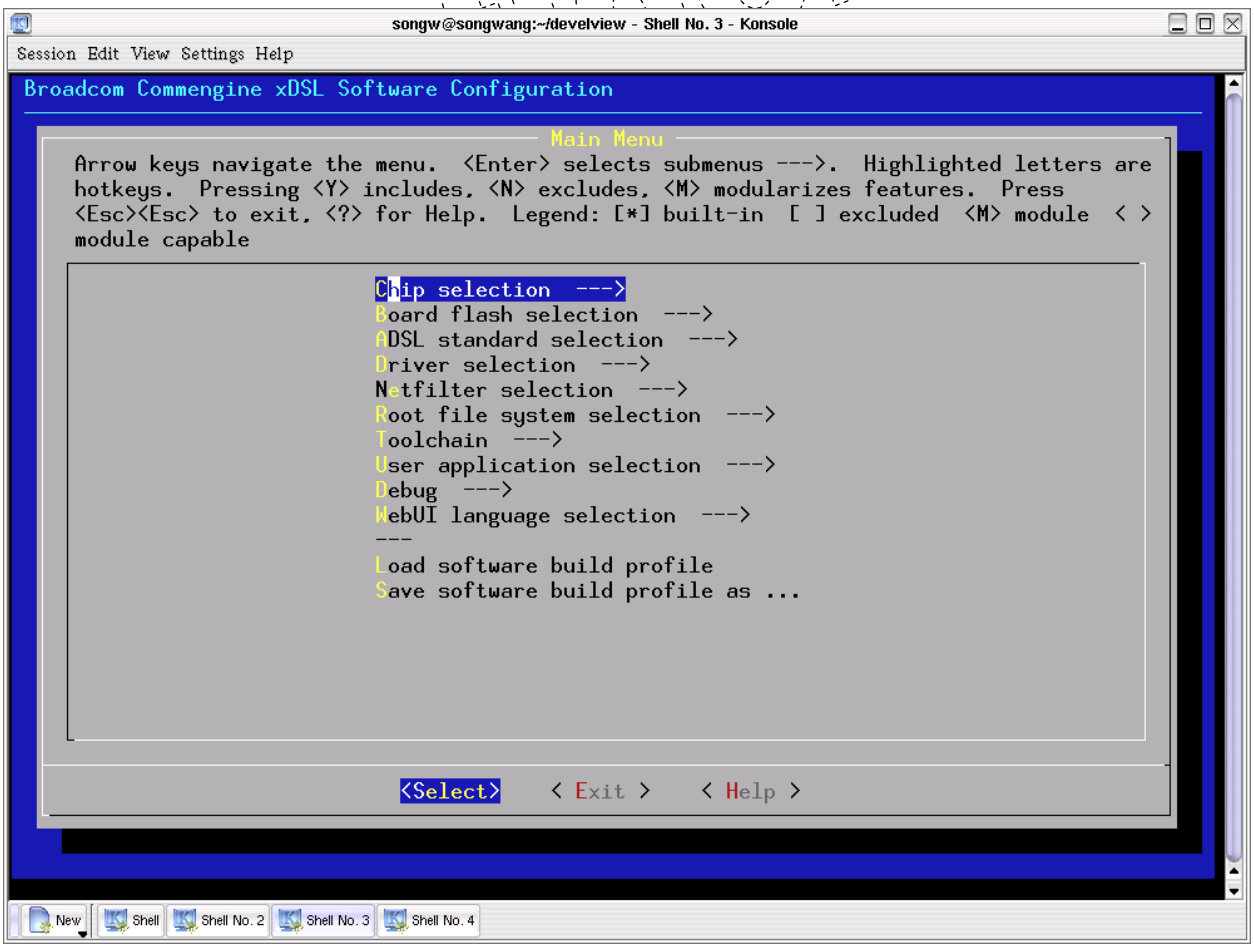
- "dynamic" means that the user application will be built as an executable
- "static" means the user application will be built as a static library
- "# ... is not set" means“ do not include the application in the build

For other configuration options:

- "y" means that the option is enabled
- "# … is not set" means the option is disabled.



At the top-level of source code directory, type **make menuconfig**, then a GUI interface appears.
.

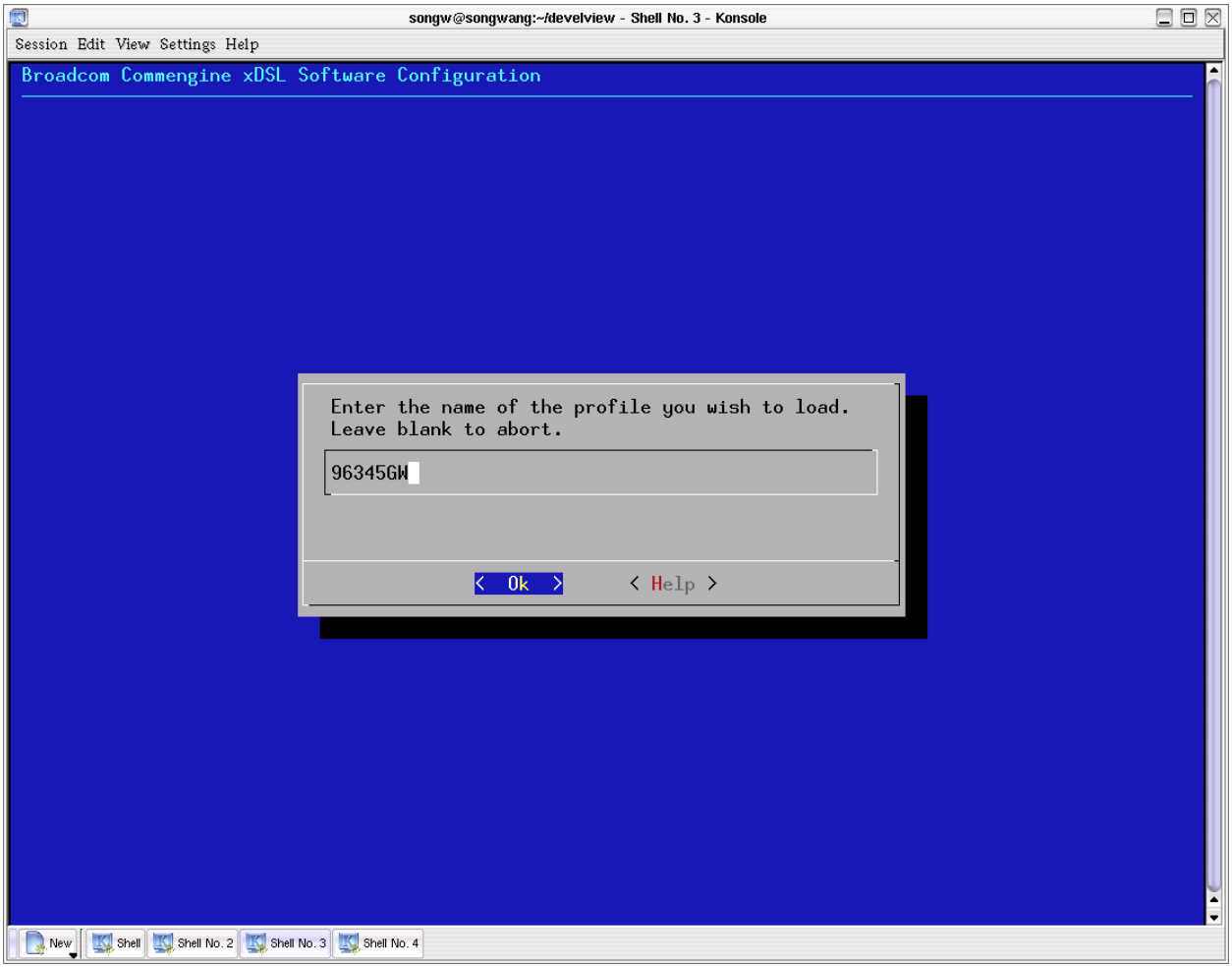



By selecting “**Load software build profile**”, you can load an existing profile, such as 96345GW or 96345R, and then you can modify configurations and select “**Save software build profile as…**” to create your own platform build profile. You can also start from scratch to create a profile. Following steps show how to create a customized profile based upon 96345GW profile.

By default, the 96345GW profile supports all the netfilter features including NAT and firewall. In this example, we show how to create a profile that does not support NAT and firewall, but it needs remote access in PPP IP extension mode. Consequently, the user applications, upnp and reaim (transparent proxy of AOL/MSN messenger) are not needed, and are excluded from the new build profile.

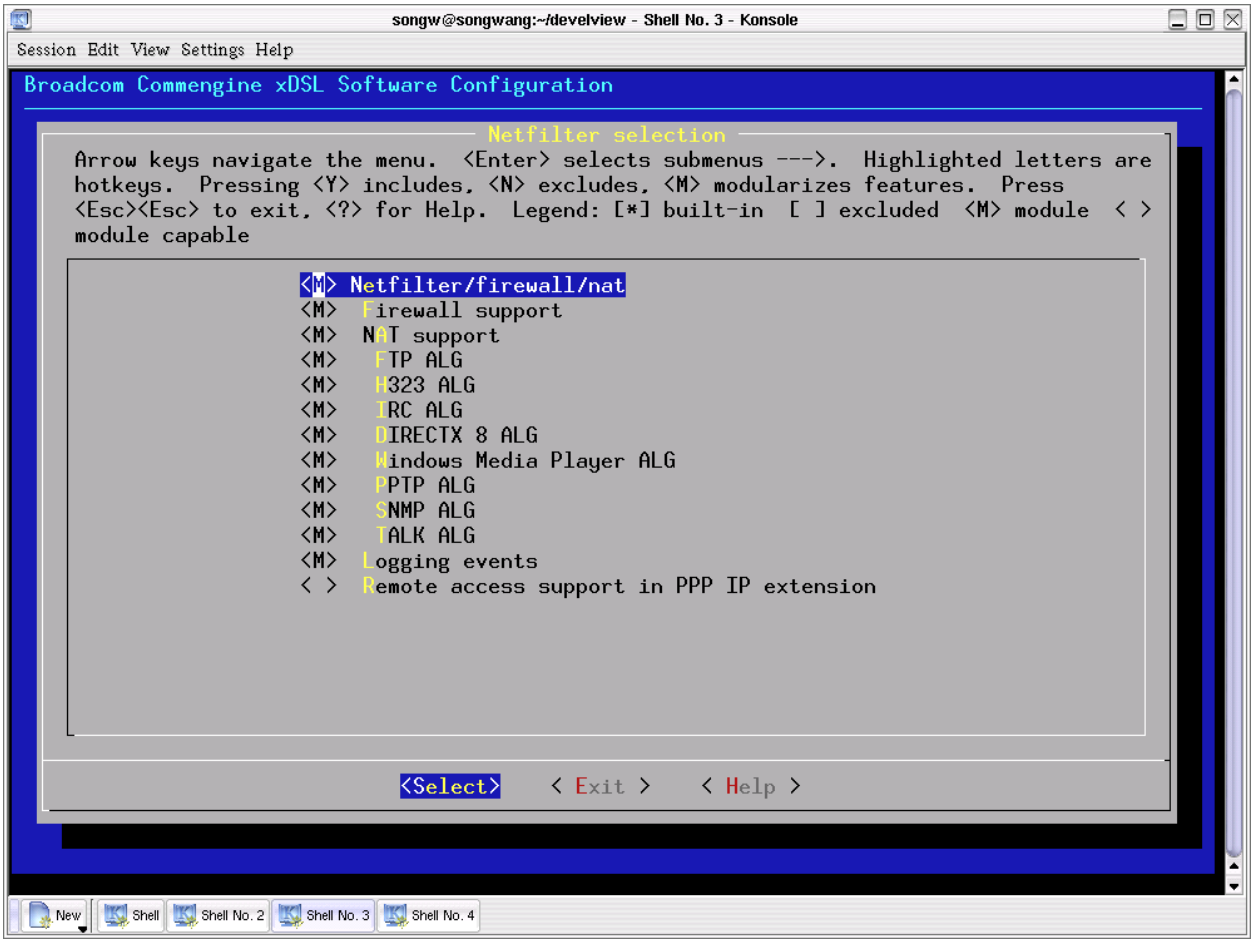
**Step 1:**

Load BCM96345GW profile by selecting "**Load software build profile**",




**Step 2:**

Select "**Netfilter selection**". NAT, firewall and ALGs are all supported by the default BCM96345GW profile.




**Step 3:**

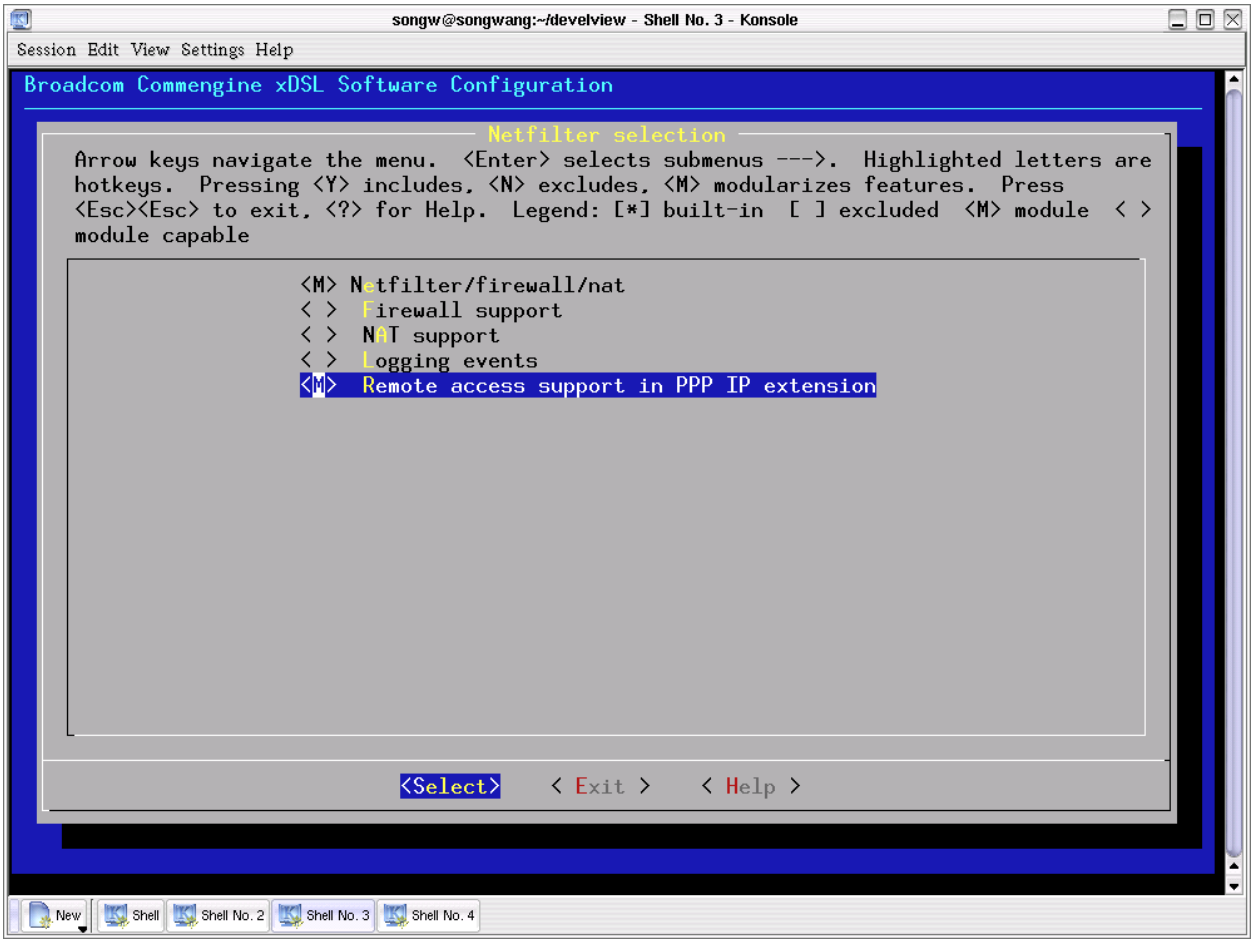
Clear "**Firewall support**", "**NAT support**" and "**Logging events**" by pressing space or "n", and then select "**Remote access support in PPP IP extension**" by pressing "m" if built as kernel modules or "y" if built as statically linked with the kernel.

Note:

When "**Firewall support**" is enabled, several netfilter kernel modules will be built into the root file system including ip_tables.o, ipt_state.o, iptable_filter.o, ipt_TCPMSS.o.

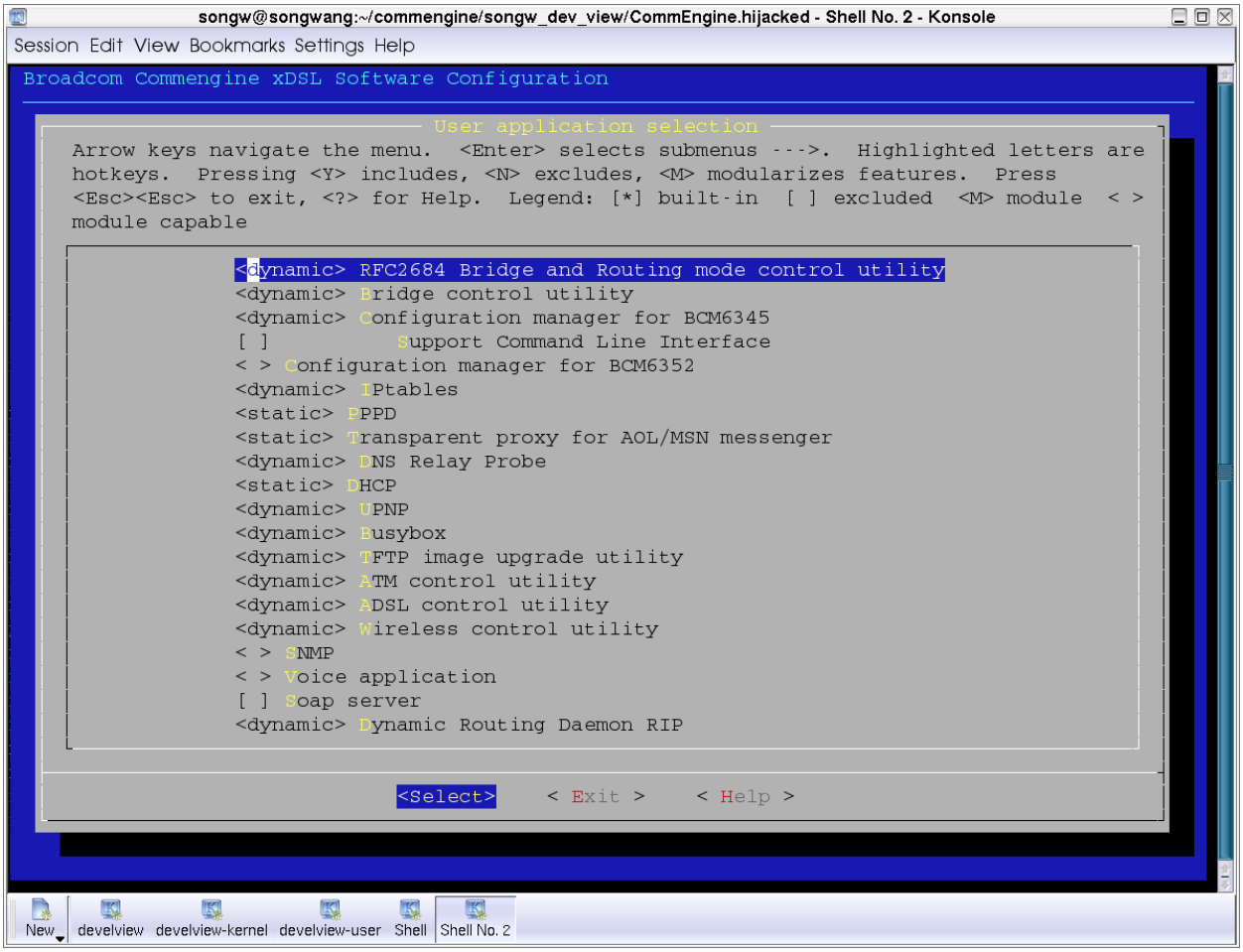
When "**NAT support**" is enabled, several netfilter kernel modules will be automatically built into the root file system including ip_tables.o, ip_conntrack.o, iptable_nat.o, ipt_MASQUERADE.o, ipt_REDIRECT.o. You can enable those ALG modules as well, such as H.323 (ip_conntrack_h323.o, ip_nat_h323.o).

When "**Remote access support in PPP IP extension**" is enabled, several netfilter kernel modules will be built into the root file system including ip_tables.o, ip_conntrack.o, iptable_nat.o, iptable_filter.o and ipt_TCPMSS.o



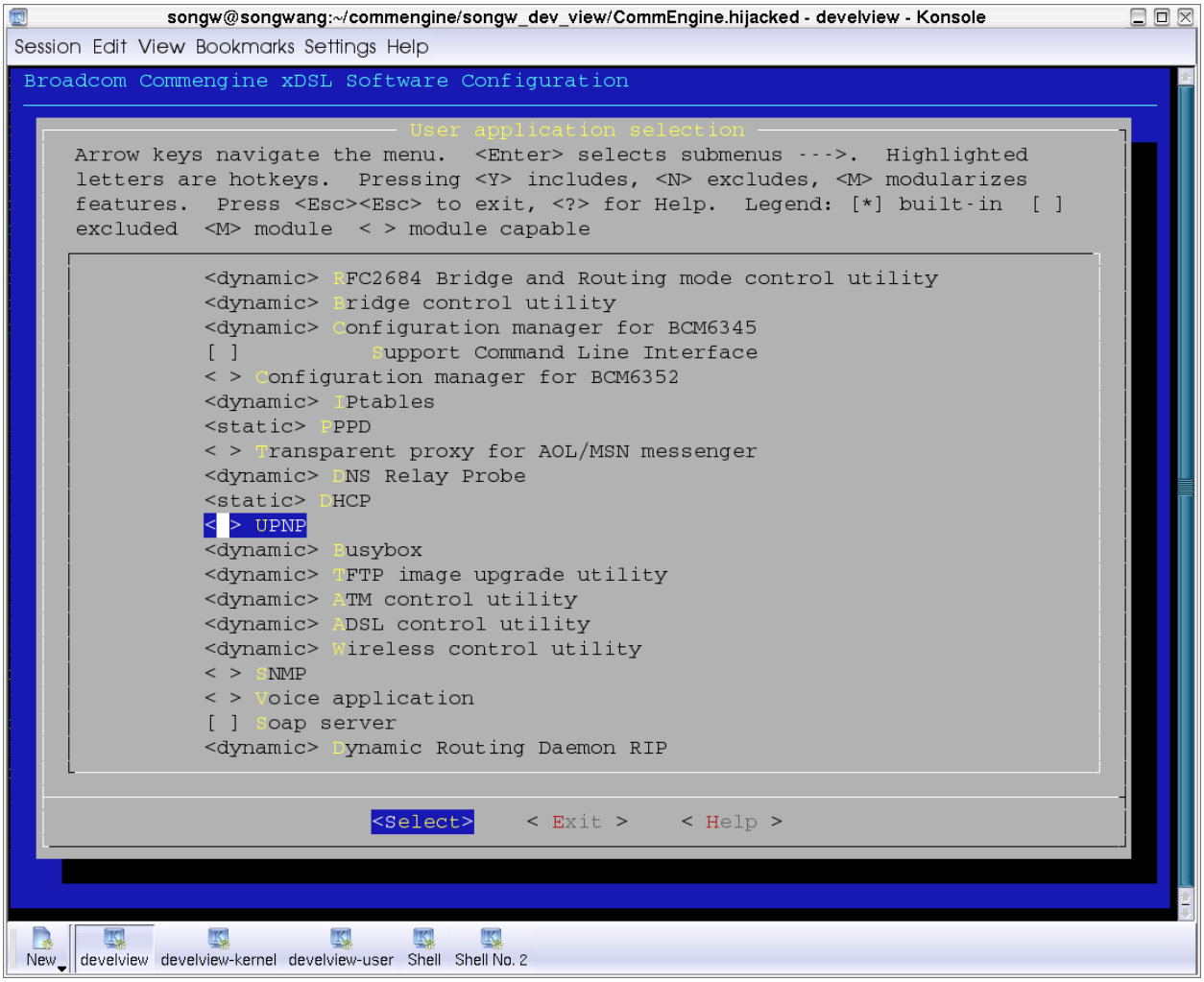

**Step 4:**

Select "**Exit**" and go back to the main menu. Select "**User application selection**". All the included applications for the BCM96345GW board are listed.



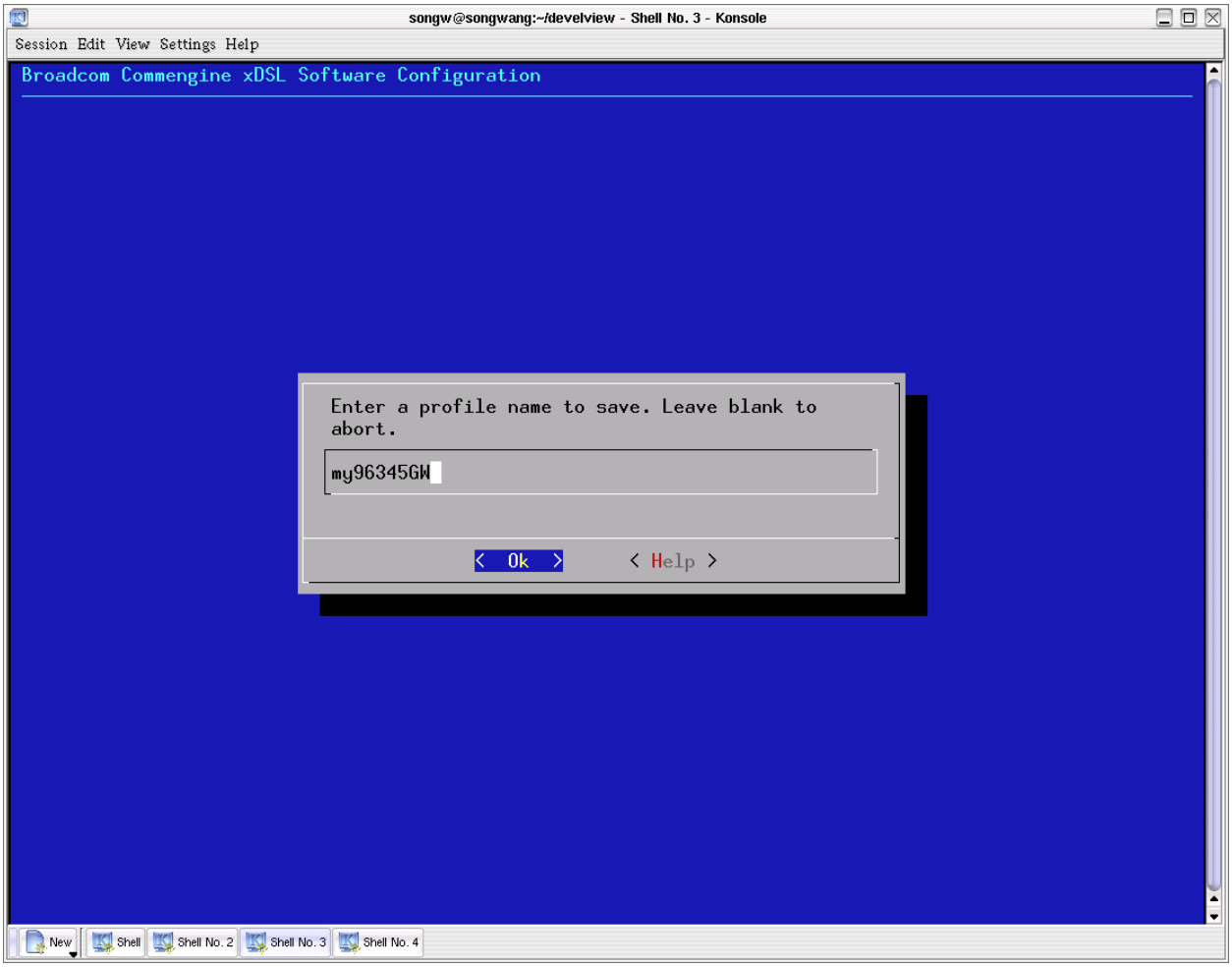

**Step 5:**

Clear “Transparent proxy for AOL/MSN messenger” and “UPNP” by pressing space or “n”,




**Step 6:**

Select "**Exit**" and go back to the main menu. Select "**Save software build profile as…**" and then type "**my96345GW**".




After the profile named “my96345GW” is saved, a new directory “my96345GW” is created under “targets” directory and the profile named “my96345GW” is also created. You can use the new profile to build your customized image by following commands described in the next section.

## 4.0 BUILD IMAGES WITH PROFILE

The image created by the build process is located in the”/opt/bcm963xx/targets/<profile_name>” directory. The same image is copied to “/opt/bcm963xx/images” directory, but with a build timestamp and version tag attached to the file name for archive purposes.

Since we may build different images with different configurations in one place, it is required to specify the profile name in all the following build commands.

### 4.1 Image build commands

The following commands list how to build the image under different circumstances.

1. Build image

$**make PROFILE=<profile name>**

2. Build kernel and final image only

$**make PROFILE=<profile name> kernel**

If you just change a kernel file and then build an image, you should use this command. This saves build time.

3. Build kernel modules and final image only

$**make PROFILE=<profile name> modules**

If you just change kernel module code such as Broadcom proprietary drivers in "bcmdrivers/broadcom" directory and then build an image, you should use this command.

4. Build applications and final image only

$**make PROFILE=<profile name> app**

If you just change a user application file and then build an image, you should use this command.

5. Build final image only

$**make PROFILE=<profile name> buildimage**

If you just change a configuration file such as a file in "fs.src" directory, you should use this command.

6. Clean up all directories

$**make PROFILE=<profile name> clean**

7. Clean up kernel and modules only

$**make PROFILE=<profile name> kernel_clean**

8. Clean up applications and target only

$**make PROFILE=<profile name> app_clean**

## 4.2 Build image with default settings

Default settings (configurations) are quite desirable for end-users because they do not need to configure the software themselves and thus it makes customer support easier. A developer can specify which default settings file to be included in the image by using GUI-based "make menuconfig" when creating a build profile.

The steps are:

1. Select "Board configuration selection"

2. Select "default settings file name" and type in the file name. The file must be located in "targets/defaultcfg" directory.

After the default settings file is specified, the build framework will copy the file from "targets/defaultcfg" directory to the target root file system, which is "fs/etc/default.cfg". When the xDSL router boots, it will use the default settings in that file to properly configure itself.

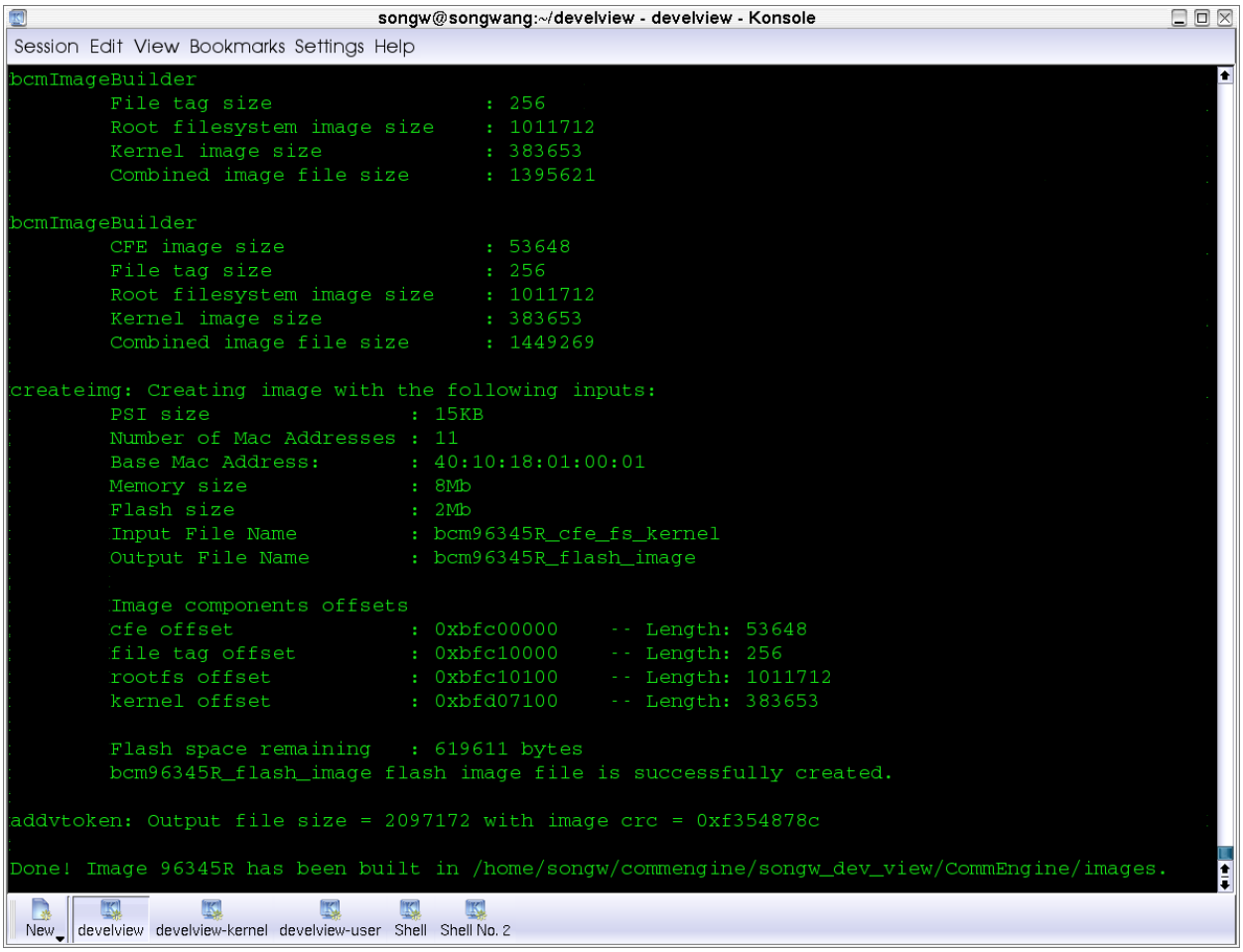
Note that the design of default settings support is backward compatible, which means that a default settings file is optional. If you do not want a default settings file and do not specify it in your profile, everything stays the same as before (the default settings are empty.)

The way to generate a default settings file for adding into the build framework is easy. Just use WebUI to configure all the settings that are expected to be included in the default settings, then select "Management/Settings/Backup" to export the settings to a file. Put the file under "targets/defaultcfg" directory for including when building an image. In this way, the default settings file can include any possible configurations.



## 4.3 Flash layout and image size report

After the profile is specified and the image built, the layout and size information of all the components in your target board flash memory are reported, including CFE bootloader, root file system, kernel, and remaining free space.




## 4.4 Other image build materials

In the *targets* directory, the *fs.src* sub-directory has configuration files and scripts that need to be copied into the target root file system when the image is built. The *fs.bin* sub-directory may contain binaries to be included in the root file system. In this case, the binaries (kernel modules or user applications) must be organized in the same way as the root file system is organized.

The "buildFS" script specifies how to build the final root file system, i.e., how to pull out the necessary files in "fs.src" into the "fs" directory. The "makeDevs" script specifies how to create the device files in the target root file system.

For each profile, its directory "targets/<profile name>/fs" directory which is generated by the build process, contains the final target root file system.

# 5.0 ADD AN APPLICATION TO THE BUILD FRAMEWORK

This section describes how to add a new application to the "userapps" directory. We will use a simple application ‘reaim’ as the example.

Note that the “static” here implies that an application will be built as a static library. It is useful if you want to build the application as the busybox applet. It DOES NOT mean that the application is a self-contained application that statically links in C library code.

## 5.1 Steps to add applications

**Step 1:**

Add your application code under the directory "userapps". If your application is based on public-domain code, you may put it under “opensource” directory, otherwise under “broadcom” directory. The recommended way to add opensource code is to add the original source tar ball such as reaim.tar.bz2 under opensource, add a directory such as reaim, and then only add whatever code you modified under that directory. In this way, it is easier to track the files you modified.

**Step 2:**

Modify the Makefile in your application source directory by adding 'dynamic' and 'static' target. The following lists the Makefile of 'reaim'.

```
#################################################################
CFLAGS= -s -Os -fomit-frame-pointer
LDFLAGS= -Wl,-allow-shlib-undefined
ifeq ($(strip $(BUILD_REAIM)), static)
CFLAGS += -DBUILD_STATIC
endif
all: reaim
install:
	install -m 755 reaim $(INSTALL_DIR)/bin
	$(STRIP) $(INSTALL_DIR)/bin/reaim
dynamic: reaim install
static: reaim.a

reaim: reaim.c
	$(CC) $(CFLAGS) $(LDFLAGS) -o reaim reaim.c
reaim.a: reaim.c
	$(CC) $(CFLAGS) -c -o reaim.o reaim.c
	$(AR) rcs reaim.a  reaim.o
clean:
	rm -f reaim reaim.o reaim.a
#################################################################
```

'dynamic' target builds dynamic-linked executable application, ‘reaim’. 'static' target builds a static library, for example, 'reaim.a'".

In CFLAGS, '-Os' to enable optimization for size. (If -Os causes problem, go back to -O2)

**Step 3:**

Modify the source file that contains the 'main' function like the following:

```
#ifdef BUILD_STATIC
int reaim_main(int argc, char *argv[])
#else
int main(int argc, char *argv[])
#endif
```

**Step 4:**

In all your profiles, define whether your application is going to be built as dynamic-linked or static library, like

BUILD_REAIM=dynamic or BUILD_REAIM=static

**Step 5:**

Add the application to the build framework and make it configurable through the GUI 'make menuconfig'. Add the following line into targets/config.in file:

tristate2 'Transparent proxy for AOL/MSN Messenger' BUILD_REAIM

**Step 6:**

In the top-level Makefile, add the following:

- Add your application source directory to either SUBDIRS_OPENSOURCE or SUBDIRS_BROADCOM
- Add your application name to either OPENSOURCE_APPS or BROADCOM_APPS
- Add the target to make your application, for example

```
ifneq ($(strip $(BUILD_REAIM)),)
reaim:
        $(MAKE) -C $(OPENSOURCE_DIR)/reaim $(BUILD_REAIM)
else
reaim:
endif
```

If you added the application as opensource code following the recommended way in step 1, you need to add one more line before $(MAKE) command to untar the original source code tar ball.

```
cd $(OPENSOURCE_DIR);   (tar xkfj reaim.tar.bz2 2> /dev/null || true)
```

- Export the variable BUILD_<YOURAPPNAME> such as BUILD_REAIM at the end of the Makefile so that the variable is accessible throughout all the Makefiles in the build framework.