



## Creating packages

One of the things that we've attempted to do with OpenWrt's template system is make it incredibly easy to port software to OpenWrt. If you look at a typical package directory in OpenWrt you'll find two things:

- package/Makefile
- package/patches
- package/files

The patches directory is optional and typically contains bug fixes or optimizations to reduce the size of the executable. The files directory is optional. It typically includes default config or init files.

The package makefile is the important item because it provides the steps actually needed to download and compile the package.

Looking at one of the package makefiles, you'd hardly recognize it as a makefile. Through what can only be described as blatant disregard and abuse of the traditional make format, the makefile has been transformed into an object oriented template which simplifies the entire ordeal.

Here for example, is package/bridge/Makefile:

```
include $(TOPDIR)/rules.mk

PKG_NAME:=bridge
PKG_VERSION:=1.0.6
PKG_RELEASE:=1

PKG_BUILD_DIR:=$(BUILD_DIR)/bridge-utils-$(PKG_VERSION)
PKG_SOURCE:=bridge-utils-$(PKG_VERSION).tar.gz
PKG_SOURCE_URL:=@SF/bridge
PKG_MD5SUM:=9b7dc52656f5cbec846a7ba3299f73bd
PKG_CAT:=zcat

include $(INCLUDE_DIR)/package.mk

define Package/bridge
    SECTION:=base
    CATEGORY:=Network
    TITLE:=Ethernet bridging configuration utility
    #DESCRIPTION:=This variable is obsolete. use the Package/name/description define instead!
    URL:=http://bridge.sourceforge.net/
endef

define Package/bridge/description
    Ethernet bridging configuration utility
    Manage ethernet bridging; a way to connect networks together to
    form a larger network.
endef

define Build/Configure
    $(call Build/Configure/Default,--with-linux-headers=$(LINUX_DIR))
endef

define Package/bridge/install
    $(INSTALL_DIR) $(1)/usr/sbin
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/brctl brctl $(1)/usr/sbin/
endef

$(eval $(call BuildPackage,bridge))
```

### BuildPackage variables

As you can see, there's not much work to be done; everything is hidden in other makefiles and abstracted to the point where you only need to specify a few variables.

- PKG\_NAME - The name of the package, as seen via menuconfig and ipkg
- PKG\_VERSION - The upstream version number that we're downloading
- PKG\_RELEASE - The version of this package Makefile
- PKG\_LICENSE - The license(s) the package is available under, SPDX form.
- PKG\_LICENSE\_FILE - file containing the license text
- PKG\_BUILD\_DIR - Where to compile the package
- PKG\_SOURCE - The filename of the original sources
- PKG\_SOURCE\_URL - Where to download the sources from (directory)
- PKG\_MD5SUM - A checksum to validate the download
- PKG\_CAT - How to decompress the sources (zcat, bzip, unzip)
- PKG\_BUILD\_DEPENDS - Packages that need to be built before this package, but are not required at runtime. Uses the same syntax as DEPENDS below.
- PKG\_INSTALL - Setting it to "1" will call the package's original "make install" with prefix set to PKG\_INSTALL\_DIR
- PKG\_INSTALL\_DIR - Where "make install" copies the compiled files
- PKG\_FIXUP - ???
- PKG\_SOURCE\_PROTO - the protocol to use for fetching the sources (git, svn)
- PKG\_REV - the svn revision to use, must be specified if proto is "svn"
- PKG\_SOURCE\_SUBDIR - must be specified if proto is "svn" or "git", e.g. "PKG\_SOURCE\_SUBDIR:=\$(PKG\_NAME)-\$(PKG\_VERSION)"
- PKG\_SOURCE\_VERSION - must be specified if proto is "git", the commit hash to check out
- PKG\_CONFIG\_DEPENDS - specifies which config options depend on this package being selected

The PKG\_\* variables define where to download the package from; @SF is a special keyword for downloading packages from sourceforge. The md5sum is used to verify the package was downloaded correctly and PKG\_BUILD\_DIR defines where to find the package after the sources are uncompressed into \$(BUILD\_DIR). PKG\_INSTALL\_DIR defines where the files will be copied after calling "make install" (set with the PKG\_INSTALL variable), and after that you can package them in the install section.

At the bottom of the file is where the real magic happens, "BuildPackage" is a macro setup by the earlier include statements. BuildPackage only takes one argument directly – the name of the package to be built, in this case "bridge". All other information is taken from the define blocks. This is a way of providing a level of verbosity, it's inherently clear what the DESCRIPTION variable in Package/bridge is, which wouldn't be the case if we passed this information directly as the Nth argument to BuildPackage.

## PKG\_FIXUP

Some/many packages that think autotools is a good idea end up needing "fixes" to work around autotools "accidentally" knowing better and using host tools instead of the build environment tools. OpenWrt defines some PKG\_FIXUP rules to help work around this. 🐛**Fix Me!** snarkiness probably not required

```
PKG_FIXUP:=autoreconf
PKG_FIXUP:=patch-libtool
PKG_FIXUP:=gettext-version
```

Any variations of this you see in the wild are simply aliases for these.

### autoreconf

This fixup performs

- autoreconf -f -i
- touch required but maybe missing files
- ensures that openwrt-libtool is linked
- suppresses autopoint/gettext

### patch-libtool

If the shipped automake recipes are broken beyond repair then simply find instances of libtool, detect their version and apply openwrt fix patches to it

### gettext-version

This fixup suppress version mismatch errors in automake's gettext support

### Tips

Packages that are using Autotools should work with simply "PKG\_FIXUP:=autoreconf". However there might be issues with required versions.

🚩 Instead of patching ./configure one should fix the file from which ./configure is generated in autotools: configure.ac (or configure.in, for very old packages). Another important file is Makefile.am from which Makefiles (with configure output) are generated.

## Package Sourcecode

OpenWrt Buildroot supports many different ways to download external source code.

### Use packed source code archive

Most packages use a packed .tar.gz, .tar.bz2, .tar.xz or similar source code file.

### Use repository

PKG\_SOURCE\_PROTO supports download from various repositories to integrate development versions.

```
PKG_SOURCE_PROTO:=bzip
PKG_SOURCE_PROTO:=cvs
PKG_SOURCE_PROTO:=darcs
PKG_SOURCE_PROTO:=git
PKG_SOURCE_PROTO:=hg
PKG_SOURCE_PROTO:=svn
```

### Bundle source code with OpenWrt Makefile

It is also possible to have the source code in the package/<packagename> directory. Often a ./src/ subdirectory is used.

Examples: px5g , px5g-standalone

### Download override

Bundled source code does not need overriding.

You can download additional data from external sources.

```
USB_IDS_VERSION:=2013-01-16
USB_IDS_MD5SUM:=2a2344907b6344f0935c86efaf9de620
USB_IDS_FILE:=usb.ids.${USB_IDS_VERSION}.gz
```

[...parts missing...]

```
define Download/usb_ids
    FILE:=$(USB_IDS_FILE)
    URL:=http://mirror2.openwrt.org/sources
    MD5SUM:=$(USB_IDS_MD5SUM)
endef
$(eval $(call Download,usb_ids))
```

and unpack it or integrate it into the build process

```
define Build/Prepare
    $(Build/Prepare/Default)
```

```

        echo '#!/bin/sh' > $(PKG_BUILD_DIR)/update-usbids.sh.in
        echo 'cp $(DL_DIR)/$(USB_IDS_FILE) usb.ids.gz' >> $(PKG_BUILD_DIR)/update-usbids.sh.in
    endef

```

You can modify UNPACK\_CMD or call/modify PKG\_UNPACK manually in your Build/Prepare section.

```
UNPACK_CMD=ar -p "$(DL_DIR)/$(PKG_SOURCE)" data.tar.xz | xzcat | tar -C $(1) -xf -
```

```

define Build/Prepare
    $(PKG_UNPACK)
#    we have to download additional stuff before patching
    (cd $(PKG_BUILD_DIR) && ./contrib/download_prerequisites)
    $(Build/Patch)
endef

```

Examples: px5g, px5g-standalone, usbutils, debootstrap, gcc,

## BuildPackage defines

### Package/

matches the argument passed to buildroot, this describes the package the menuconfig and ipkg entries. Within Package/ you can define the following variables:

- SECTION - The type of package (currently unused)
- CATEGORY - Which menu it appears in menuconfig
- TITLE - A short description of the package
- DESCRIPTION - (deprecated) A long description of the package
- URL - Where to find the original software
- MAINTAINER - (required for new packages) Who to contact concerning the package
- DEPENDS - (optional) Which packages must be built/installed before this package. See [below](#) for the syntax.
- USERID - (optional) a username:groupname pair to create at package installation time.

### Package/confiles (optional)

A list of config files installed by this package, one file per line.

### Package/description

A free text description of the package

### Build/Prepare (optional)

A set of commands to unpack and patch the sources. You may safely leave this undefined.

### Build/Configure (optional)

You can leave this undefined if the source doesn't use configure or has a normal config script, otherwise you can put your own commands here or use "\$(call Build/Configure/Default,)" as above to pass in additional arguments for a standard configure script.

### Build/Compile (optional)

How to compile the source; in most cases you should leave this undefined, because then the default is used, which calls make. If you want to pass special arguments to make, use e.g. "\$(call Build/Compile/Default,FOO=bar)

### Build/Install (optional)

How to install the compiled source. The default is to call make install. Again, to pass special arguments or targets, use "\$(call Build/Install/Default,install install-foo) Note that you need put all the needed make arguments here. If you just need to add something to the "install" argument, don't forget the 'install' itself.

### Build/InstallDev (optional)

For things needed to compile packages against it (static libs, header files), but that are of no use on the target device.

### Package/install

A set of commands to copy files into the ipkg which is represented by the \$(1) directory. As source you can use relative paths which will install from the unpacked and compiled source, or \$(PKG\_INSTALL\_DIR) which is where the files in the Build/Install step above end up.

### Package/preinst

The actual text of the script which is to be executed before installation. Dont forget to include the #!/bin/sh. If you need to abort installation have the script return false.

### Package/postinst

The actual text of the script which is to be executed after installation. Dont forget to include the #!/bin/sh.

### Package/prem

The actual text of the script which is to be executed before removal. Dont forget to include the #!/bin/sh. If you need to abort removal have the script return false.


### Package/postrm

The actual text of the script which is to be executed after removal. Dont forget to include the #!/bin/sh.


The reason that some of the defines are prefixed by "Package/" and others are simply "Build" is because of the possibility of generating multiple packages from a single source. OpenWrt works under the assumption of one source per package makefile, but you can split that source into as many packages as desired. Since you only need to compile the sources once, there's one global set of "Build" defines, but you can add as many "Package/" defines as you want by adding extra calls to BuildPackage – see the dropbear package for an example.

## Dependency Types

Various types of dependencies can be specified, which require a bit of explanation for their differences.

|            |   |
|------------|---|
| +<foo>     | Package will depend on package <foo> and will select it when selected.  |
| <foo>      | Package will depend on package <foo> and will be invisible until <foo> is selected.   |
| @FOO       | Package depends on the config symbol CONFIG_FOO and will be invisible unless CONFIG_FOO is set. This usually used for depending on certain Linux versions or targets, e.g. @TARGET_foo will make a package only available for target foo. You can also use boolean expressions for complex dependencies, e.g. @(TARGET_foo&&!TARGET_bar) will make the package unavailable for foo and bar. |
| +FOO:<bar> | Package will depend on <bar> if CONFIG_FOO is set, and will select <bar> when it is selected itself. The typical use case would be if there compile time options for this package toggling features that depend on external libraries.  Note that the + replaces the @.                                    |
| @FOO:<bar> | Package will depend on <bar> if CONFIG_FOO is set, and will be invisible until <bar> is selected when CONFIG_FOO is set.  |

Some typical config symbols for (conditional) dependencies are:

|                                   |   |
|-----------------------------------|---|
| TARGET_<foo>                      | Target <foo> is selected  |
| TARGET_<foo>_<bar>                | If the target <foo> has subtargets, subtarget <foo> is selected. If not, profile <foo> is selected. This is in addition to TARGET_<foo>   |
| TARGET_<foo>_<bar>_<baz>          | Target <foo> with subtarget <bar> and profile <baz> is selected.  |
| LINUX_3_X                         | Linux version used is 3.x.*   |
| LINUX_2_6_X                       | Linux version used is 2.6.x.* (:1: only used for backfire and earlier)  |
| LINUX_2_4                         | Linux version is 2.4  only used in backfire and earlier, and only for target bcm-2.4)          |
| USE_UCLIBC, USE_GLIBC, USE_EGLIBC | To (not) depend on a certain libc.  |
| BROKEN                            | Package doesn't build or work, and should only be visible if "Show broken targets/packages" is selected. Prevents the package from failing builds by accidentally selecting it. |
| IPV6                              | IPv6 support in packages is selected.   |

## Configure a package source

Example:

```
CONFIGURE_ARGS += \
    --disable-native-affinity \
    --disable-unicode \
    --enable-hwloc

CONFIGURE_VARS += \
    ac_cv_file__proc_stat=yes \
    ac_cv_file__proc_meminfo=yes \
    ac_cv_func_malloc_0_nonnull=yes \
    ac_cv_func_realloc_0_nonnull=yes
```

To set variables (autoconfig internal ones or CPPFLAGS,CFLAGS, CXXFLAGS, LDFLAGS for example) or configure arguments. Setting configure arguments is common. Setting VARS is needed when the configure.ac autoconf source script does not work well on cross compilation or finding libraries.

### Host tools required

If your package needs some private tools built on the host, you can use the following snippet as a pointer of where to look for more info

```
HOST_BUILD_DEPENDS:=<packagename>/host
PKG_BUILD_DEPENDS:=<packagename>/host
include $(INCLUDE_DIR)/host-build.mk

define Host/Compile
define Host/Install

$(eval $(call HostBuild))
```

TODO Expand on how to use this, and include examples



Extracted from this thread on the devel mailing list: <https://lists.openwrt.org/pipermail/openwrt-devel/2014-February/023970.html> [<https://lists.openwrt.org/pipermail/openwrt-devel/2014-February/023970.html>]

### NOTES

All variables in your pre/post install/removal scripts should have double (\$\$) instead of a single (\$) string characters. This will inform "make" to not interpret the value as a variable, but rather just ignore the string and replace the double \$\$ by a single \$ – More Info [<https://forum.openwrt.org/viewtopic.php?id=85197#p85197>]

After you've created your package Makefile, the new package will automatically show in the menu the next time you run "make menuconfig" and if selected will be built automatically the next time "make" is run.

DESCRIPTION is obsolete, use Package/PKG\_NAME/description.

## Adding configuration options

If you would like configure your package installation/compilation in the menuconfig you can do the following: Add MENU:=1 to your package definition like this:

```
define Package/mjpg-streamer
    SECTION:=multimedia
    CATEGORY:=Multimedia
    TITLE:=MJPEG-streamer
    DEPENDS:=@!LINUX_2_4 +libpthread-stubs +jpeg
    URL:=http://mjpg-streamer.wiki.sourceforge.net/
    MENU:=1
endef
```

Create a config key in the Makefile:

```
define Package/mjpg-streamer/config
    source "$(SOURCE)/Config.in"
endef
```

Create a Config.in file directory where the Makefile is located with the content like this:

```
# Mjpg-streamer configuration
menu "Configuration"
    depends on PACKAGE_mjpg-streamer

config MJPEG_STREAMER_AUTOSTART
    bool "Autostart enabled"
    default n

    menu "Input plugins"
        depends on PACKAGE_mjpg-streamer
        config MJPEG_STREAMER_INPUT_FILE
            bool "File input plugin"
            help
                You can stream pictures from jpg files on the filesystem
            default n

        config MJPEG_STREAMER_INPUT_UVC
            bool "UVC input plugin"
            help
                You can stream pictures from an Universal Video Class compatible webcam
            default y

        config MJPEG_STREAMER_FPS
            depends MJPEG_STREAMER_INPUT_UVC
            int "Maximum FPS"
            default 15

        config MJPEG_STREAMER_PICT_HEIGHT
            depends MJPEG_STREAMER_INPUT_UVC
            int "Picture height"
            default 640

        config MJPEG_STREAMER_PICT_WIDTH
            depends MJPEG_STREAMER_INPUT_UVC
            int "Picture width"
            default 480

        config MJPEG_STREAMER_DEVICE
            depends MJPEG_STREAMER_INPUT_UVC
            string "Device"
            default /dev/video0

        config MJPEG_STREAMER_INPUT_GSPCA
            bool "GSPCA input plugin"
            help
                You can stream pictures from a gspca supported webcam. Note this module is deprecated,
            default n

    endmenu

# .....

endmenu
```

Above you can see examples for various type config parameters.

And finally you can check your configuration parameters in your Makefile in the following way: (Note that you can reference to the parameters value with it name prefixed with CONFIG\_)

```
ifeq ($(CONFIG_MJPEG_STREAMER_INPUT_UVC),y)
    $(CP) $(PKG_BUILD_DIR)/input_uvc.so $(1)/usr/lib
endif
```

## Working on local application source

If you are still working on the application itself, at the same time as you are working on the packaging, it can be very useful to have OpenWrt build your work in progress code, rather than a specific version+md5sum combination checked out of revision control, or downloaded from your final "release" location. There are a few ways of doing this.

### CONFIG\_SRC\_TREE\_OVERRIDE

This is an option in menuconfig. See "Advanced configuration options (for developers)" → "Enable package source tree override"

This allows you to point to a local git tree. (And only git) Say your package is defined in my\_cool\_feed/awesome\_app.

```
ln -s /path/to/local/awesome_app_tree/.git feeds/my_cool_feed/awesome_app/git-src
make package/awesome_app/{clean,compile} V=s
```

Benefits of this approach are that you don't need any special infrastructure in your package makefiles, they stay completely as they would be for a final build. The downside is that it only builds whatever is currently \_committed\_ in HEAD of your local tree. (Not <em>master</em>, this could be a private testing branch, but it must be <em>committed</em> it can't be local changes) This will also use a \_separate\_ directory for building and checking out the code. So any built objects in your local git tree (for example, build targeting a different architecture) will be left alone, but whatever \_branch\_ is checked out in your tree determines where HEAD is.

### USE\_SOURCE\_DIR

As part of deprecating package-version-override.mk (below) a method to point directly to local source was introduced.

```
make package/awesome_app/clean V=s
make package/awesome_app/prepare USE_SOURCE_DIR=~/.src/awesome_src V=s
make package/awesome_app/clean V=s
```

(V=s is optional above)

This doesn't require any config change to enable rules, and doesn't require that you have a local git tree, and doesn't require any files to be committed.

At least at present however, this has the following problems:

- make clean doesn't clean the source link directory, but still seems to be removing a link
- make prepare needs to be run every time
- make package/awesome\_app/ {clean,compile} USE\_SOURCE\_DIR=~/.blah doesn't work
- Seems to have bad interactions with leaving USE\_SOURCE\_DIR set for other (dependent?) packages.

See <http://www.mail-archive.com/openwrt-devel@lists.openwrt.org/msg23122.html> [<http://www.mail-archive.com/openwrt-devel@lists.openwrt.org/msg23122.html>] for the original discussion of this new feature

### (Deprecated) package-version-override.mk

**⚠ Don't use this anymore** Support for this style of local source building was removed in: <https://dev.openwrt.org/changeset/40392> [<https://dev.openwrt.org/changeset/40392>]. This style required a permanent modification to your package makefile, and then entering a path via menuconfig to where the source was found. It was fairly easy to use, and didn't care whether your local source was in git or svn or visual source safe even, but it had the major downside that the "clean" target simply didn't work. (As it simply removed a symlink for cleaning)

If you build a current OpenWrt tree, with packages that still attempt to use this style of local building, you \_will\_ receive errors like so: ERROR: please fix package/feeds/feed\_name/application\_name/Makefile - see logs/package/feeds/feed\_name/application\_name/dump.txt for details

If you need/want to keep using this style, where it's available, make sure you include without failing if it was missing:

```
-include $(INCLUDE_DIR)/package-version-override.mk
```

## Creating packages for kernel modules

A kernel module [<http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html>] is an installable program which extends the behavior of the linux kernel. A kernel module gets loaded after the kernel itself, (e.g. using insmod).

Many kernel programs are included in the linux source distribution; typically the kernel build may be configured to, for each program,

1. compile it into the kernel as a built-in,
2. compile it as a loadable kernel module, or
3. ignore it.

See ~~***FIX:Customizing the kernel options customizing the kernel options***~~ for including it in the kernel.

To include one of these programs as a loadable module, select the corresponding kernel option in the OpenWrt configuration (see Build Configuration). If your favorite kernel module does not appear in the OpenWrt configuration menus, you must add a stanza to one of the files in the package/kernel/linux/modules directory. Here is an example extracted from .../modules/block.mk:

```
define KernelPackage/loop
    SUBMENU:=$(BLOCK_MENU)
    TITLE:=Loopback device support
    KCONFIG:= \
        CONFIG_BLK_DEV_LOOP \
        CONFIG_BLK_DEV_CRYPTOLOOP=n
    FILES:=$(LINUX_DIR)/drivers/block/loop.ko
    AUTOLOAD:=$(call AutoLoad,30,loop)
endef

define KernelPackage/loop/description
    Kernel module for loopback device support
endef

$(eval $(call KernelPackage,loop))
```

Changes to the \*.mk files are not automatically picked up by the build system. To force re-reading the meta data either touch the kernel package Makefile using touch package/kernel/linux/Makefile (on older revisions touch package/kernel/Makefile) or to delete the tmp/ directory of the buildroot.

You can also add kernel modules which are *not* part of the linux source distribution. In this case, a kernel module appears in the package/ directory, just as any other package does. The package/Makefile uses

```
KernelPackage/xxx
```

definitions in place of

```
Package/xxx
```

For example, here is package/madwifi/Makefile:

```
#
# Copyright (C) 2006 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#
# $Id$

include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk

PKG_NAME:=madwifi
PKG_VERSION:=0.9.2
PKG_RELEASE:=1
```

```

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_SOURCE_URL:=@SF/$(PKG_NAME)
PKG_MD5SUM:=a75baacbe07085ddc5cb28elfb43edbb
PKG_CAT:=bzcatt

PKG_BUILD_DIR:=$(KERNEL_BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)

include $(INCLUDE_DIR)/package.mk

RATE_CONTROL:=sample

ifeq ($(ARCH),mips)
    HAL_TARGET:=mips-be-elf
endif
ifeq ($(ARCH),mipsel)
    HAL_TARGET:=mips-le-elf
endif
ifeq ($(ARCH),i386)
    HAL_TARGET:=i386-elf
endif
ifeq ($(ARCH),armeb)
    HAL_TARGET:=xscale-be-elf
endif
ifeq ($(ARCH),powerpc)
    HAL_TARGET:=powerpc-be-elf
endif

BUS:=PCI
ifneq ($(CONFIG_LINUX_2_4_AR531X),)
    BUS:=AHB
endif
ifneq ($(CONFIG_LINUX_2_6_ARUBA),)
    BUS:=PCI AHB # no suitable HAL for AHB yet.
endif

BUS_MODULES:=
ifeq ($(findstring AHB,$(BUS)),AHB)
    BUS_MODULES+=$(PKG_BUILD_DIR)/ath/ath_ahb.$(LINUX_KMOD_SUFFIX)
endif
ifeq ($(findstring PCI,$(BUS)),PCI)
    BUS_MODULES+=$(PKG_BUILD_DIR)/ath/ath_pci.$(LINUX_KMOD_SUFFIX)
endif

MADWIFI_AUTOLOAD:= \
    wlan \
    wlan_scan_ap \
    wlan_scan_sta \
    ath_hal \
    ath_rate_$(RATE_CONTROL) \
    wlan_acl \
    wlan_ccmp \
    wlan_tkip \
    wlan_wep \
    wlan_xauth

ifeq ($(findstring AHB,$(BUS)),AHB)
    MADWIFI_AUTOLOAD += ath_ahb
endif
ifeq ($(findstring PCI,$(BUS)),PCI)
    MADWIFI_AUTOLOAD += ath_pci
endif

define KernelPackage/madwifi
    SUBMENU:=Wireless Drivers
    DEFAULT:=y if LINUX_2_6_BRCM | LINUX_2_6_ARUBA | LINUX_2_4_AR531X | LINUX_2_6_XSCALE, m if ALL
    TITLE:=Driver for Atheros wireless chipsets
    DESCRIPTION:=\
        This package contains a driver for Atheros 802.11a/b/g chipsets.
    URL:=http://madwifi.org/
    VERSION:=$(LINUX_VERSION)+$(PKG_VERSION)-$(BOARD)-$(PKG_RELEASE)
    FILES:= \
        $(PKG_BUILD_DIR)/ath/ath_hal.$(LINUX_KMOD_SUFFIX) \
        $(BUS_MODULES) \
        $(PKG_BUILD_DIR)/ath_rate/$(RATE_CONTROL)/ath_rate_$(RATE_CONTROL).$(LINUX_KMOD_SUFFIX) \
        $(PKG_BUILD_DIR)/net80211/wlan*.$(LINUX_KMOD_SUFFIX)
    AUTOLOAD:=$(call AutoLoad,50,$(MADWIFI_AUTOLOAD))
endef

MADWIFI_MAKEOPTS= -C $(PKG_BUILD_DIR) \
    PATH="$(TARGET_PATH)" \
    ARCH="$(LINUX_KARCH)" \
    CROSS_COMPILE="$(TARGET_CROSS)" \
    TARGET="$(HAL_TARGET)" \
    TOOLPREFIX="$(KERNEL_CROSS)" \
    TOOLPATH="$(KERNEL_CROSS)" \
    KERNELPATH="$(LINUX_DIR)" \
    LDOPTS=" " \
    ATH_RATE="ath_rate/$(RATE_CONTROL)" \
    DOMULTI=1

```

```

ifeq ($(findstring AHB,$(BUS)),AHB)
    define Build/Compile/ahb
        $(MAKE) $(MADWIFI_MAKEOPTS) BUS="AHB" all
    endef
endif

ifeq ($(findstring PCI,$(BUS)),PCI)
    define Build/Compile/pci
        $(MAKE) $(MADWIFI_MAKEOPTS) BUS="PCI" all
    endef
endif

define Build/Compile
    $(call Build/Compile/ahb)
    $(call Build/Compile/pci)
endef

define Build/InstallDev
    $(INSTALL_DIR) $(STAGING_DIR)/usr/include/madwifi
    $(CP) $(PKG_BUILD_DIR)/include $(STAGING_DIR)/usr/include/madwifi/
    $(INSTALL_DIR) $(STAGING_DIR)/usr/include/madwifi/net80211
    $(CP) $(PKG_BUILD_DIR)/net80211/*.h $(STAGING_DIR)/usr/include/madwifi/net80211/
endef

define KernelPackage/madwifi/install
    $(INSTALL_DIR) $(1)/etc/init.d
    $(INSTALL_DIR) $(1)/lib/modules/$(LINUX_VERSION)
    $(INSTALL_DIR) $(1)/usr/sbin
    $(INSTALL_BIN) ./files/madwifi.init $(1)/etc/init.d/madwifi
    $(CP) $(PKG_BUILD_DIR)/tools/{madwifi_multi,80211debug,80211stats,athchans,athctrl,athdebug,athkey,athstats,wlanconfig} $(1)
endef

$(eval $(call KernelPackage,madwifi))

```

## File installation macros

INSTALL\_DIR, INSTALL\_BIN, INSTALL\_DATA are used for creating a directory, copying an executable, or a data file. +x is set on the target file for INSTALL\_BIN, independent of it's mode on the host.

From the big document:

Package/<name>/install:

A set of commands to copy files out of the compiled source and into the ipkg which is represented by the \$(1) directory. Note that there are currently 4 defined install macros:

```

INSTALL_DIR
install -d -m0755
INSTALL_BIN
install -m0755
INSTALL_DATA
install -m0644
INSTALL_CONF
install -m0600

```

## Packaging a service

If you want to install a service, (something that should start/stop at boot time, that has a /etc/init.d/blah script), you should make sure that the init.d script can be run on the host. At image build time, all init.d scripts found are run on the host, looking for the START=20/STOP=99 lines.

This is what installs the symlinks in /etc/rc.d, so they are only created when you rebuild the entire image. If you want the symlinks to be created when a package is installed, such as via opkg, you should add a postinstall script which runs

```
/etc/init.d/foo enable
```

if \$IPKG\_INSTROOT is empty.

When \$IPKG\_INSTROOT is defined, you run within the buildroot, if it is empty you run on the target.

Example makefile snippet to install/remove symlinks.

```

define Package/mrelay/postinst
#!/bin/sh
# check if we are on real system
if [ -z "${IPKG_INSTROOT}" ]; then
    echo "Enabling rc.d symlink for mrelay"
    /etc/init.d/mrelay enable
fi
exit 0
endef

define Package/mrelay/prerm
#!/bin/sh
# check if we are on real system
if [ -z "${IPKG_INSTROOT}" ]; then
    echo "Removing rc.d symlink for mrelay"
    /etc/init.d/mrelay disable
fi
exit 0
endef

```

Very basic example of a suitable init.d script



❗ **procd** style init is used in some init.d scripts since: <https://dev.openwrt.org/changeset/38023> [<https://dev.openwrt.org/changeset/38023>] . See <http://wiki.openwrt.org/inbox/procd-init-scripts> [<http://wiki.openwrt.org/inbox/procd-init-scripts>] for more details on that

```
#!/bin/sh /etc/rc.common
# "new(er)" style init script
# Look at /lib/functions/service.sh on a running system for explanations of what other SERVICE_
# options you can use, and when you might want them.

START=80
APP=mrelay
SERVICE_WRITE_PID=1
SERVICE_DAEMONIZE=1

start() {
    service_start /usr/bin/$APP
}

stop() {
    service_stop /usr/bin/$APP
}

#!/bin/sh /etc/rc.common
#####
# NOTE - this is an old style init script #
#####

START=80
APP=mrelay
PID_FILE=/var/run/$APP.pid

start() {
    start-stop-daemon -S -x $APP -p $PID_FILE -m -b
}

stop() {
    start-stop-daemon -K -n $APP -p $PID_FILE -s TERM
    rm -rf $PID_FILE
}
```

## How To Submit Patches to OpenWrt

Packages are maintained in a separate repository to reduce maintenance overhead. The general guidelines for OpenWrt still apply, but see the README in the packages repository for latest information.

- <https://github.com/openwrt/packages> [<https://github.com/openwrt/packages>]
- <https://dev.openwrt.org/wiki/SubmittingPatches> [<https://dev.openwrt.org/wiki/SubmittingPatches>]

See <https://lists.openwrt.org/pipermail/openwrt-devel/2014-June/025810.html> [<https://lists.openwrt.org/pipermail/openwrt-devel/2014-June/025810.html>] for the original announcement of this change