Using the Image Builder

The Image Builder (previously called the Image Generator) is a pre-compiled environment suitable for creating custom images without the need for compiling them from source.
It downloads pre-compiled packages and integrates them in a single flashable image.

Doing so is useful if:

- you want to fit more packages in a small flash size
- you want to follow development snapshots
- your device has 32MB or less RAM and opkg does not work properly
- you want to mass-flash dozens of devices and you need a specific firmware setup

Alternative guides to achieving the same goal: Quick Image Building Guide, Beginners guide to building your own firmware.

Consider also removing packages if you have a device with very little firmware space: Saving Firmware Space.

Frontends based on imagebuilder

There are several tools that provide a frontend interface to the imagebuilder (either web-interface, or template-based)

See imagebuilder_frontends.

Prerequisites

The Image Builder runs only in 64bit linux. You can however run a 64bit linux in VM (i.e. virtualbox) even from 32bit windows.
The Image Builder has similar prerequisites as Build system setup.

Example dependencies in the most common distros:

Arch / Manjaro

```
pacman -S --needed base-devel ncurses zlib gawk git gettext \
openssl libxslt wget unzip python
```

CentOS / Fedora

```
dnf install git gawk gettext ncurses-devel zlib-devel \
openssl-devel libxslt wget which @c-development @development-tools \
@development-libs zlib-static which python3
```

Debian / Ubuntu

```
apt install build-essential libncurses5-dev libncursesw5-dev \
zlib1g-dev gawk git gettext libssl-dev xsltproc wget unzip python
```

Obtaining the Image Builder

You can download an archive that contains the **Image Builder**, it is usually located in the same download page where you find the firmware image for your device.
For example, this is the page where you can download all firmware images for **ath79/generic** devices
https://downloads.openwrt.org/snapshots/targets/ath79/generic/ (https://downloads.openwrt.org/snapshots/targets/ath79/generic/) .
and you will find a **openwrt-imagebuilder-ath79-generic.Linux-x86_64.tar.xz** archive with the image builder in it. Also, it is always created by the build system because it is needed to create the image file. If the option "**Build the OpenWrt Image Builder**" is enabled, the image builder will be generated in the same folder you find firmware images (`source/bin/targets/xxx`) and you can use it to create more images from the packages you obtained during compilation.

Add Package Repositories (optional)

The **Image Generator** you download from the OpenWrt pages is already configured to download any non-default packages from official repositories.
The package sources are configured in the `repositories.conf` file in the extracted directory. Sources are specified in *opkg* native config format. This can be either the official package repositories or custom generated repositories.

An example of the contents of the `repositories.conf` from the **openwrt-imagebuilder-18.06.0-rc2-ramips-mt7621.Linux-x86_64.tar.xz** :

```
## Place your custom repositories here, they must match the architect
ure and version.
# src/gz %n http://downloads.openwrt.org/releases/18.06.0-rc2
# src custom file:///usr/src/openwrt/bin/ramips/packages

## Remote package repositories
src/gz openwrt_core http://downloads.openwrt.org/releases/18.06.0-rc2
/targets/ramips/mt7621/packages
src/gz openwrt_base http://downloads.openwrt.org/releases/18.06.0-rc2
/packages/mipsel_24kc/base
src/gz openwrt_luci http://downloads.openwrt.org/releases/18.06.0-rc2
/packages/mipsel_24kc/luci
src/gz openwrt_packages http://downloads.openwrt.org/releases/18.06.0
-rc2/packages/mipsel_24kc/packages
src/gz openwrt_routing http://downloads.openwrt.org/releases/18.06.0-
rc2/packages/mipsel_24kc/routing
src/gz openwrt_telephony http://downloads.openwrt.org/releases/18.06.
0-rc2/packages/mipsel_24kc/telephony

## This is the local package repository, do not remove!
src imagebuilder file:packages
```

The `repositories.conf` in an imagebuilder you compile from source will lack the “Remote package repositories” links.

If you want to add a custom local repository, copy the

```
src custom file:///usr/src/openwrt/bin/ramips/packages
```

line and modify it to point to the local folder you have your packages and package lists in. If you have problems with using you local repository because the “Signature check failed” then remove the line `option check_signature` from `repositories.conf`

If you have custom repositories online, copy and modify the

```
src/gz reboot http://downloads.openwrt.org/snapshots
```

line instead.

Usage

All operations should be performed using a general (non-root) user account.

Unpacking:

```
tar xJf openwrt*.tar.xz
```

make image without specifying the `PROFILE` for your device is almost certainly not what you want.

To change this not-so-useful default behavior you can use some variables passed as arguments:

- *PROFILE* - specifies the target image to build
- *PACKAGES* - a list of packages to embed into the image
- *FILES* - directory with custom files to include
- *BIN_DIR* - alternative output directory for the images
- *EXTRA_IMAGE_NAME* - Add this to the output image filename (sanitized)
- *DISABLED_SERVICES* - Which services in /etc/init.d/ should be disabled. Use the initscript name you find in /etc/init.d, so for example "**dhcp**" for dnsmasq.

(see also the makefile used, here (https://github.com/openwrt/openwrt/blob/master/target/imagebuilder/files/Makefile))

Example syntax:

```
$ make image PROFILE=XXX PACKAGES="pkg1 pkg2 pkg3 -pkg4 -pkg5 -pkg6"
FILES=files/
```

See the sections below for a more in-depth explanation. After the make command is finished, the generated images are stored in the bin*/device-architecture* directory, just like if you were compiling them.

here the output of **make help**:

```
Available Commands:
        help:   This help text
        info:   Show a list of available target profiles
        clean:  Remove images and temporary build files
        image:  Build an image (see below for more information).

Building images:
        By default 'make image' will create an image with the default
        target profile and package set. You can use the following par
ameters
        to change that:

        make image PROFILE="<profilename>" # override the default tar
get profile
        make image PACKAGES="<pkg1> [<pkg2> [<pkg3> ...]]" # include
extra packages
        make image FILES="<path>" # include extra files from <path>
        make image BIN_DIR="<path>" # alternative output directory fo
r the images
        make image EXTRA_IMAGE_NAME="<string>" # Add this to the outp
ut image filename (sanitized)
        make image DISABLED_SERVICES="<svc1> [<svc2> [<svc3> ..]]" #
Which services in /etc/init.d/ should be disabled

Print manifest:
        List "all" packages which get installed into the image.
        You can use the following parameters:

        make manifest PROFILE="<profilename>" # override the default
target profile
        make manifest PACKAGES="<pkg1> [<pkg2> [<pkg3> ...]]" # inclu
de extra packages
```

The built image will be found under the subdirectory `./bin/targets/<target>/generic` or look inside . `/build_dir/` for a files `*-squashfs-sysupgrade.bin` and `*-squashfs-factory.bin` (e.g. `/build_dir/target-mips_24kc_musl/linux-ar71xx_tiny /tmp/openwrt-18.06.2-ar71xx-tiny-tl-wr740n-v6-squashfs-factory.bin`)

PROFILE Variable

Syntax:

```
$ make image PROFILE=NAME_OF_PROFILE
```

Pre-defined Profiles

Run `make info` to obtain a list of defined profiles. Example output from `make info` is listed below.

ar71xx-generic Profiles

```
Available Profiles:

Default:
    Default Profile
    Packages: kmod-usb-core kmod-usb2 kmod-usb-ohci kmod-usb-ledtrig-
usbport
ai-br100:
    Aigale Ai-BR100
    Packages: kmod-usb2 kmod-usb-ohci
rp-n53:
    Asus RP-N53
    Packages:
rt-n14u:
    Asus RT-N14u
    Packages:
whr-1166d:
    Buffalo WHR-1166D
    Packages:
whr-300hp2:
    Buffalo WHR-300HP2
    Packages:
-and many many more-
```

After you find the appropriate profile pass it to the `make image` command:

For example, if we wanted to generate a default image for for Asus RT-N14u (from above).

```
$ make image PROFILE=rt-n14u
```

PACKAGES Variable

The `PACKAGES` variable allows to include and/or exclude packages in the firmware image. By default (empty PACKAGES variable) the Image Generator will create a minimal image with device-specific kernel and drivers, uci, ssh, switch, firewall, ppp and ipv6 support.

Syntax:

```
$ make image PACKAGES="pkg1 pkg2 pkg3 -pkg4 -pkg5 -pkg6"
```

The example above will include pkg1, pkg2, pkg3, and exclude pkg4, pkg5, pkg6, note the “-” before each excluded package.

You don't need to list all dependencies of the packages you need in this list, the Image Generator uses `opkg` to resolve automatically the package dependencies and install other required packages.

Tip: The list of currently installed packages on your device can be obtained with the command below:

```
echo $(opkg list_installed | awk '{ print $1 }')
```

> Many devices are limited in storage capacity and there is no guarantee that the build system will detect when you have added too many packages to fit into the device storage space, which may render the device unbootable if installed. If in doubt, do not go overboard. Use what you had installed on the device last as a guide or create a minimal image first, install it to the device and test what you would like to add first.

FILES Variable

The `FILES` variable allows custom configuration files to be included in images built with Image Generator. This is especially useful if you need to change the network configuration from default before flashing, or if you are preparing an image for mass-flashing many devices.

Syntax:

```
$ make image FILES=files/
```

Note: The `files/` folder is best in the imagebuilder root folder (where you issue the make command) otherwise it is best to use an absolute (full) path.

Examples

The following example shows:

1. Creating the directory for the configuration files
2. Using `scp` to transfer `uci` configuration files from a WL500GP router to the `files/etc/config` directory
3. Generating an image for WL500GP with custom packages and `uci` configuration files

```
mkdir -p files/etc/config
scp root@192.168.1.1:/etc/config/network files/etc/config/
scp root@192.168.1.1:/etc/config/wireless files/etc/config/
scp root@192.168.1.1:/etc/config/firewall files/etc/config/
make image PROFILE=wl500gp PACKAGES="nano openvpn -ppp -ppp-mod-pppo
e" FILES=files/
```

Cleanup

To clean up temporary build files and generated images, use the **make clean** command.

Advanced Topics

The topics below go beyond simple usage and aimed at developers and advanced users.

Adding/Modifying Profiles

The image generation is tied to the profile names. If you add a new profile without also adding an appropriate macro to the image-generation Makefile, no suitable firmware file will get generated when using the custom profile. Make sure to remove the /tmp directory to get modified package selection from profiles to work.

The location of the profiles for the pre-compiled package for *brcm47xx-for-Linux-i686* was *target/linux/brcm47xx/profiles/*

Remarkably, all that needs to be done to add a new profile, is to add a new file to the *profiles* directory. *While this may have been the case in earlier releases, for 17.01, it appears that manual editing of `.targetinfo` is also required.*

Here is what the *profiles/100-Broadcom-b43.mk* profile file looks like:

```
define Profile/Broadcom-b43
  NAME:=Broadcom BCM43xx WiFi (default)
  PACKAGES:=kmod-b43 kmod-b43legacy
endef

define Profile/Broadcom-b43/Description
        Package set compatible with hardware using Broadcom BCM43xx c
ards
endef
$(eval $(call Profile,Broadcom-b43))
```

Alternately edit the hidden .profile.mk file at the top level directory of the image builder (e.g. me@mymachine:~/openwrt-imagebuilder-versxxx-at91-sam9x-.Linux-x86_64#) and manually add the names of the desired packages to be added to the output image. An “ls -a” will reveal the files hidden in the various directories.

Remove useless files from firmware

This is not a standard feature of the Image Builder.

It is highly recommended that you test file removal prior to incorporating such changes at the image builder level or that you have low level means to recover a device before attempting this type of mod. As bricking / non booting may result.

Note that it requires patching of the `Makefile`

1. Create file 'files_remove' with full filenames:

```
/lib/modules/3.10.49/ts_bm.ko
/lib/modules/3.10.49/nf_nat_ftp.ko
/lib/modules/3.10.49/nf_nat_irc.ko
/lib/modules/3.10.49/nf_nat_tftp.ko
```

2. Patch Makefile

```
 ifneq ($(USER_FILES),)
        $(MAKE) copy_files
 endif
+
+ifneq ($(FILES_REMOVE),)
+       @echo
+       @echo Remove useless files
+
+       while read filename; do                          \
+           rm -rfv "$(TARGET_DIR)$$filename";  \
+       done < $(FILES_REMOVE);
+endif
+
        $(MAKE) package_postinst
        $(MAKE) build_image
```

3. Rebuild firmware

```
# make image \
    PROFILE=tlwr841 \
    PACKAGES="igmpproxy ip iptraf kmod-ipt-nathelper-extra openvpn-po
larssl tcpdump-mini -firewall -ip6tables -kmod-ip6tables -kmod-ipv6 -
odhcp6c -ppp -ppp-mod-pppoe" \
    FILES_REMOVE="files_remove"
```

Building the Image Generator with all packages inside

It is possible to build the Image Generator and integrate in it all packages so it will be able to

generate images without downloading packages:

In the graphical configuration, select "**Build the OpenWrt Image Builder**" to build the image builder, then select **Global Build Settings → Select all packages by default**, save and exit. Then build the image, including `IGNORE_ERRORS=1` as there might be unmaintained packages that fail to compile.

Enabling `IGNORE_ERRORS=1` should only be done **once the kernel and required packages are known to compile successfully.**

```
make IGNORE_ERRORS=1
```

Note: Don't call `make defconfig` or leave an old `.config` file in the path as `Select all packages by default` will only set the package selection to `[m]` for packages that are not already configured otherwise!(`make defconfig` will set most packages to `[n]` , i.e. *do not build*.)

Last modified: 2021/01/05 20:34 by pmelange