

Package SRC

Version 4.0.0-trunk-rpi-r60747

Christoph Schulz
email: fli4l@kristov.de

The fli4l-Team
email: team@fli4l.de

September 15, 2022

Contents

1. Documentation For Package SRC	3
1.1. SRC - The fli4l Buildroot	3
1.1.1. The Sources - An Overview	3
1.1.2. Compile A Program For fli4l	4
1.1.3. Testing Of A Compiled Program	7
1.1.4. Debugging Of A Compiled Program	7
1.1.5. Informations On The FBR	10
1.1.6. Changing The FBR Configuration	11
1.1.7. Updating The FBR	12
1.1.8. Integrating Own Programs Into The FBR	13
A. Appendix For Package SRC	14

1. Documentation For Package SRC

marklabelbuildroot

1.1. SRC - The fli4l Buildroot

This chapter is mainly of interest for developers, who want to compile binaries or the Linux kernel for the fli4l. If you only want to use fli4l as a router and do not offer packages for the fli4l needing own binaries, you may skip this chapter completely.

In general, for compiling program packages for the fli4l a Linux system is required. Compilation under other operating systems (Microsoft Windows, OS X, FreeBSD, etc.) is *not* supported.

The requirements for a Linux system for fli4l development are as follows:

- GNU gcc and g++ in version 2.95 or higher
- GNU gcc-multilib (depending on your host system)
- GNU binutils (contains bind and other necessary programs)
- GNU make in version 3.81 or higher
- GNU bash
- libncurses5-dev for `fbr-make *-menuconfig` (depending on your host system)
- The programs sed, awk, which, flex, bison and patch
- The programs makeinfo (package texinfo) and msgfmt (package gettext)
- The programs tar, cpio, gzip, bzip2 and unzip
- The programs wget, rsync, svn and git
- The programs perl and python

In the following, characters printed **bold** represent keyboard input, the ↵-character stands for the Enter key on your keyboard and executes entered commands.

1.1.1. The Sources - An Overview

In the directory `src` you will find the following subdirectories:

1. Documentation For Package SRC

Directory	Content
fbr	In this folder there is a custom build system based on the buildroot for uClibc (currently in version 0.9.33.2). FBR stands for “fil4-Buildroot”. It is thus possible, to compile all programs used on fli4l (kernel, libraries and applications) anew.
fli4l	This directory contains the fli4l specific sources for packages sorted by their names. All sources that are included in this subdirectory, were either written specifically for use with fli4l or are at least strongly adapted.
cross	This directory contains scripts that create the cross-compilers necessary for compiling mkfli4l for various Platforms.

1.1.2. Compile A Program For fli4l

In the subdirectory “fbr” a script **fbr-make** exists which controls the compilation of all the programs from the basic packages for fli4l. This script takes care of downloading and compiling all required binaries for fli4l. The script will save files in the directory `~/fbr`, if it does not yet exist, it will be created. (The directory may be changed by using the environment variable `FBR_BASEDIR`, see below.)

Note: During the compilation process much space is needed (currently around 900 MiB for the downloaded archives and almost 30 GiB for the intermediate results and the resulting compiled files). Hence, make sure to have enough space under `~/fbr`! (Alternatively, you may also use the `FBR_TIDY` option, see below.)

The directory structure below `~/fbr` is as follows:

Directory	Content
fbr-<branch>-<arch>	The uClibc buildroot is unpacked here. <code><branch></code> stands for the development branch the FBR is derived from, in this case (i.e. <code>trunk</code>). If the origin of the FBR is an unpacked <code>src</code> package, <code>fbr-custom</code> will be used. <code><arch></code> reflects the processor architecture in use (i.e. <code>x86</code> or <code>x86_64</code>). More concerning this directory can be found below.
dl	Here the downloaded archives are stored.
own	Here own FBR packages which should be compiled can be stored.

Under the Buildroot directory `~/fbr/fbr-<branch>-<arch>/buildroot` the following directories are of interest:

Directory	Content
output/sandbox	In this directory a subdirectory exists for each FBR package that holds the files of the package after being compiled. In the directory <code>output/sandbox/<package>/target</code> the files for the fli4l router can be found in this case. In the directory <code>output/sandbox/<package>/staging</code> files needed for building <i>other</i> FBR packages requiring this FBR package can be found.
output/target	In this directory <i>all</i> compiled programs for the fli4l-router are stored. This directory mirrors the directory structure on the fli4l router. By the help of <code>chroot</code> you can change to this directory and try the programs. ¹

General Settings

The operation of `fbr-make` can be influenced by various environment variables:

Variable	Description
FBR	Specifies the path to the FBR explicitly. Per default the path <code>~/.fbr/fbr-<branch>-<arch></code> (see above) is used.
FBR_BASEDIR	explicitly specifies the base path to the FBR. As a default the <code>~/.fbr</code> (see above) will be used. This variable will be ignored if the environment variable FBR is set.
FBR_DLDIR	Specifies the directory containing the source archives. Per default the path <code>\${FBR}/../dl</code> (i.e. <code>~/.fbr/dl</code>) will be used.
FBR_BRANCH	Specifies explicitly the name of the branch under which the packages below <code>~/.fbr</code> (see above) are built. This variable will be ignored if the environment variable FBR is set.
FBR_CATEGORY	Specifies explicitly the name of the category under which the packages below <code>~/.fbr</code> (see above) are built. This variable will be ignored if the environment variable FBR is set.
FBR_OWNDIR	Specifies the directory holding the own packages. Per default the path <code>\${FBR}/../own</code> (i.e. <code>~/.fbr/own</code>) will be used.
FBR_TIDY	if this variable is set to “y” intermediate results while compiling the FBR packages will be deleted immediately after their installation to the directory <code>output/target</code> . This saves a lot of space and is always recommended if you do not feel the urge to have a look at <code>output/build/...</code> after the build process. If this variable contains the value “k”, only the intermediate results in the various Linux kernel directories are removed, because this saves a lot of space without losing any functionality. All other assignments (or if the variable is missing entirely) ensure that all intermediate results are kept.

¹This is bound to some preconditions, see the paragraph “Testing Of A Compiled Program” (Page 7).

1. Documentation For Package SRC

Variable	Description
FBR_ARCH	This variable specifies the processor architecture for which the FBR (or FBR packages) should be built. If it is missing, x86 will be used. The supported architectures can be found below.

The FBR currently supports the following architectures:

Architecture	Description
x86	Intel x86-Architecture (32-Bit), also known as IA-32.
x86_64	AMD x86-64-Architecture (64-Bit), also called Intel 64 or EM64T by Intel.

Compilation Of All FBR Packages

If executing `fbr-make` with the argument `world`, it may last several hours to download and compile all source archives, depending on the computer and Internet connections used. ²

Compiling The Toolchain

If executing `fbr-make` with the argument `toolchain`, all FBR packages needed for building the `fli4l` binary programs will be downloaded and compiled (Compiler, Binder, `uClibc-library` etc.). Normally this command is not needed, because all FBR packages depend on the toolchain and thus it has to be downloaded and build anyway.

Compiling A Single FBR Package

If you only want to compile a certain FBR package (i.e. the programs of an OPT developed yourself), you may transfer the name of one or more FBR packages to the `fbr-make` program (for example `fbr-make openvpn` to download and compile the OpenVPN programs). All dependencies will also be downloaded and compiled.

Recompilation Of A Single FBR Package

If you like to compile a FBR package again (for whatever reason), you first need to remove the information on the FBR about the previous compilation process. For this purpose use the command `fbr-make <package>-clean` (eg `fbr-make openvpn-clean`). In this case informations about all dependencies for the package are also reset causing their recompilation with the next `fbr-make world` as well.

Recompiling All FBR Packages

If you like to recompile the entire FBR (eg because you want to use it as a benchmark program for your new high-end developer system ;-), you can use the command `fbr-make clean` and

²Downloading of source archives is of course done only once as long as you don't update the FBR and thus need new package versions and source archives.

remove all artifacts that have been generated during the last FBR build. You will have to confirm this action. ³ This is also useful to free used disk space.

1.1.3. Testing Of A Compiled Program

If a program has been compiled with `fbr-make` it may also be tested on the development machine. Such a test will of course only work if the processor architecture of the developer machine matches the processor architecture the fli4l programs were compiled for. (It is not possible, for example, to run `x86_64` fli4l programs on a `x86` operating system.) If this condition is met, change to the fli4l target directory with

```
chroot ~/.fbr/fbr-<branch>-<arch>/buildroot/output/target /bin/sh
```

and test the compiled program(s) there immediately. Please note that executing `chroot` needs administrator rights and thus you have to use `sudo` or `su`, depending on preference and system configuration! In addition you will have to compile the FBR package `busybox` (via `fbr-make busybox`), to have a working shell in the `chroot` environment. A small example:

```
$ sudo chroot ~/.fbr/fbr-trunk-x86/buildroot/output/target /bin/sh
Password:(Your password)
```

```
BusyBox v1.22.1 (fli4l) built-in shell (ash)
Enter 'help' for a list of built-in commands.
```

```
# ls
THIS_IS_NOT_YOUR_ROOT_FILESYSTEM  mnt
bin                               opt
dev                               proc
etc                               root
home                             run
img                              sbin
include                          share
lib                              sys
lib32                           tmp
libexec                         usr
man                             var
media                           windows
# bc --version
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
# echo "42 - 23" | bc
19
#
```

1.1.4. Debugging Of A Compiled Program

In case of problems with compiled fli4l programs (crashes) you have the option to analyze the state of the program immediately before the crash (also called “post-mortem debugging”). To do so, activate `DEBUG_ENABLE_CORE='yes'` in the configuration of the `base` package. If case of

³The whole directory `~/.fbr/fbr-<branch>-<arch>/buildroot/output` will be removed.

1. Documentation For Package SRC

a crash a memory dump is generated in `/var/log/dumps/core.<PID>`. “PID” is the process ID of the crashed process. You may analyze the state of the program on a Linux machine with a fully compiled FBR as described below. The following example to be analyzed is the program `/usr/sbin/collectd`, which was terminated with a SIGBUS. The dump was stored in `/tmp/core.collectd`.

```
fli4l@eisler:~$ .fbr/fbr-trunk-x86/buildroot/output/host/usr/bin/i586-linux-gdb
GNU gdb (GDB) 7.5.1
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=i586-buildroot-linux-uclibc".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) set sysroot /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target
(gdb) set debug-file-directory /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/debug
(gdb) file /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/usr/sbin/collectd
Reading symbols from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/usr/sbin/collectd...Reading symbols from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/debug/.build-id/8b/28ab573be4a2302e1117964edede2e54ebdbf.debug...done.
done.
(gdb) core /tmp/core.collectd
[New LWP 2250]
[New LWP 2252]
[New LWP 2259]
[New LWP 2257]
[New LWP 2255]
[New LWP 2232]
[New LWP 2235]
[New LWP 2238]
[New LWP 2242]
[New LWP 2244]
[New LWP 2245]
[New LWP 2231]
[New LWP 2243]
[New LWP 2251]
[New LWP 2248]
[New LWP 2239]
[New LWP 2229]
[New LWP 2249]
[New LWP 2230]
[New LWP 2247]
[New LWP 2233]
[New LWP 2256]
[New LWP 2236]
[New LWP 2246]
[New LWP 2240]
```


1. Documentation For Package SRC

```
[New LWP 2241]
[New LWP 2237]
[New LWP 2234]
[New LWP 2253]
[New LWP 2254]
[New LWP 2258]
[New LWP 2260]
Failed to read a valid object file image from memory.
Core was generated by `collected -f'.
Program terminated with signal 7, Bus error.
#0  0xb7705f5d in memcpy ()
    from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/lib/libc.so.0
(gdb) backtrace
#0  0xb7705f5d in memcpy ()
    from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/lib/libc.so.0
#1  0xb768a251 in rrd_write (rrd_file=rrd_file@entry=0x808e930, buf=0x808e268,
    count=count@entry=112) at rrd_open.c:716
#2  0xb76834f3 in rrd_create_fn (
    file_name=file_name@entry=0x808d2f8 "/data/rrdtool/db/vm-fli4l-1/cpu-0/cpu-i
nterrupt.rrd.async", rrd=rrd@entry=0xacff2f4c) at rrd_create.c:727
#3  0xb7683d7b in rrd_create_r (
    filename=filename@entry=0x808d2f8 "/data/rrdtool/db/vm-fli4l-1/cpu-0/cpu-int
errupt.rrd.async", pdp_step=pdp_step@entry=10, last_up=last_up@entry=1386052459,
    argc=argc@entry=16, argv=argv@entry=0x808cf18) at rrd_create.c:580
#4  0xb76b77fd in srrd_create (
    filename=0xacff33f0 "/data/rrdtool/db/vm-fli4l-1/cpu-0/cpu-interrupt.rrd.asy
nc",
    pdp_step=10, last_up=1386052459, argc=16, argv=0x808cf18) at utils_rrdcreate
.c:377
#5  0xb76b78cb in srrd_create_thread (targs=targs@entry=0x808bab8)
    at utils_rrdcreate.c:559
#6  0xb76b7a8f in srrd_create_thread (targs=0x808bab8) at utils_rrdcreate.c:491
#7  0xb7763430 in ?? ()
    from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/lib/libpthread
.so.0
#8  0xb775e672 in clone ()
    from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/lib/libpthread
.so.0
(gdb) frame 1
#1  0xb768a251 in rrd_write (rrd_file=rrd_file@entry=0x808e930, buf=0x808e268,
    count=count@entry=112) at rrd_open.c:716
716     memcpy(rrd_simple_file->file_start + rrd_file->pos, buf, count);
(gdb) print (char*) buf
$1 = 0x808e268 "RRD"
(gdb) print rrd_simple_file->file_start
value has been optimized out
(gdb) list
711     if((rrd_file->pos + count) > old_size)
712     {
713         rrd_set_error("attempting to write beyond end of file");
714         return -1;
715     }
716     memcpy(rrd_simple_file->file_start + rrd_file->pos, buf, count);
```

1. Documentation For Package SRC

```
717         rrd_file->pos += count;
718         return count;          /* mimic write() semantics */
719     #else
720         ssize_t _sz = write(rrd_simple_file->fd, buf, count);
(gdb) list 700↵
695     * rrd_file->pos of rrd_simple_file->fd.
696     * Returns the number of bytes written or <0 on error.  */
697
698     ssize_t rrd_write(
699         rrd_file_t *rrd_file,
700         const void *buf,
701         size_t count)
702     {
703         rrd_simple_file_t *rrd_simple_file = (rrd_simple_file_t *)rrd_file->
pvt;
704     #ifdef HAVE_MMAP
(gdb) print *((rrd_simple_file_t *)rrd_file->pvt)↵
$2 = {fd = 9, file_start = 0xa67d0000 <Address 0xa67d0000 out of bounds>,
      mm_prot = 3, mm_flags = 1}
```

After some “digging”, you will see that an invalid pointer is contained in the `rrd_simple_file_t` object (“Address ... out of bounds”). In the further process of debugging, it became clear that a failed `posix_fallocate`-call was the culprit for the crash.

It is important to pass *all* paths fully qualified (`/project/...`) and to use no “shortcuts” (i.e. in `~/...`). If you don’t obey this it may happen that `gdb` will not find the debug informations for the application and/or for the libraries in use. Due to space reasons debug informations are not contained directly in the program to be investigated but are saved in separate files in the directory `~/fbr/fbr-<branch>-<arch>/buildroot/output/debug/`.

1.1.5. Informations On The FBR

Displaying Help

Use the command `fbr-make help` to see what `fbr-make` can do for you.

Displaying Program Informations

By using the command `fbr-make show-versions` you can review all FBR packages provided with version number:

```
$ fbr-make show-versions↵
Configured packages

acpid 2.0.20
actctrl 3.25+dfsg1
add-days undefined
[...]
```

Display Of Dependencies On Libraries

With `fbr-make links-against <soname>` all files in `~/fbr/fbr-<branch>-<arch>/buildroot/output/target` linked to a library named `soname` can be shown. If for example all programs

1. Documentation For Package SRC

and libraries should be identified that use `libm` (Library with mathematical functions) use `fbr-make links-against libm.so.0` because `libm.so.0` is the name of the `libm` library. A possible output would be:

```
$ fbr-make links-against librrd_th.so.4
Executing plugin links-against
Files linking against librrd_th.so.4
collectd usr/lib/collectd/rrdcached.so
collectd usr/lib/collectd/rrdtool.so
rrdtool usr/bin/rrdcached
```

In the first column is the package name and in the second the (relative) path to the file that is linked against the library in question.

To find the library name for a library, you can use `readelf` like this:

```
$ readelf -d ~/.fbr/fbr-trunk-x86/buildroot/output/target/lib/libm-0.9.33.2.so |
> grep SONAME
0x0000000e (SONAME)                Library soname: [libm.so.0]
```

Display Of Version Changes

The command `fbr-make version-changes` is interesting (only) for `fli4l`-team developers with write access to the `fli4l` SVN repository. It lists all FBR packages whose version has been modified locally, i.e. those where the version in the working copy differs from the repository version. This helps the developer to get an overview on updated FBR packages before writing the changes to the repo. A possible output is:

```
$ fbr-make version-changes
Executing plugin version-changes
Package version changes
KAMAILIO: 4.0.5 --> 4.1.1
```

Here you can see that the package `kamailio` in FBR was updated from version 4.0.5 to version 4.1.1.

1.1.6. Changing The FBR Configuration

Reconfiguration Of The FBR

By using `fbr-make buildroot-menuconfig` it is possible to select the FBR packages to be compiled. This is useful if you want to compile other FBR packages for the `fli4l` that are not enabled by default but are integrated in the `uClibc` buildroot, or if you want to activate own FBR packages. On the other hand global properties of FBR may be changed, such as the version of the used GCC compiler. On successful exit of the configuration menu, the new configuration is saved in the directory `src/fbr/buildroot/.config`.

Please note, however, that such changes of the toolchain configuration are *not* officially supported because the resulting binaries will be incompatible with the official `fli4l` distribution with a high probability. So if you need binaries for your own OPT and want to publish this OPT, you should not change the toolchain settings!

Reconfiguration Of The uClibc Library

With `fbr-make uclibc-menuconfig` the functionality of the uClibc library in use may be changed. On successful exit of the configuration menu, the new configuration is saved to `src/fbr/buildroot/package/uclibc/uclibc.config`.

Like in the last paragraph also here applies: A change is most likely not compatible with the official fli4l distribution and is thus not supported!

Reconfiguration Of Busybox

With `fbr-make busybox-menuconfig` the Busybox may be changed in its functionality. On successful exit of the configuration menu, the new configuration is saved to `src/fbr/buildroot/package/busybox/busybox-<Version>.config`.

Also here applies: A change is most likely not compatible with the official fli4l distribution and is thus not supported! Adding new Busybox applets causes no problems as long as you only use the modified Busybox on your own fli4l router (and not require the users of your OPT to use a Busybox modified in this way).

Reconfiguration Of The Linux Kernel Packages

With `fbr-make linux-menuconfig` resp. `fbr-make linux-<version>-menuconfig` the configuration of all activated Kernel packages resp. a specific Kernel package may be changed. On successful exit of the configuration menu, the new configuration is saved to `src/fbr/buildroot/linux/linux-<version>/dot-config-<arch>`.⁴

Like in the last paragraph also here applies: A change is most likely not compatible with the official fli4l distribution and is thus not supported! At most, adding of new modules to the Linux kernel is easy, as long you only use the modified kernel on your own fli4l router (and not require the users of your OPT to use a Kernel modified in this way).

1.1.7. Updating The FBR

Each of the commands outlined above is advanced by an examination of the actuality of the FBR. If a discrepancy between the sources in which `fbr-make` is located (unpacked `src-package` or SVN-working copy) and the FBR in `~/fbr/fbr-<branch>-<arch>/buildroot` is detected the latter will be updated. New FBR packages will be integrated and old, no longer contained FBR packages will be deleted. The configurations are compared: FBR packages with modified configuration and all dependent FBR packages will be rebuilt. This ensures that the FBR on your computer is always equal to the developer's one (except for your own FBR packages under `~/fbr/own/`). **However, This also means that changes to the official part of the Buildroot configuration will be lost with the next update!** Therefore it is not recommended to reconfigure FBR, at least not if you are using `src` packages instead of a SVN working copy. (When updating a SVN working copy your local configuration changes and

⁴This only applies to the standard Kernel. For variants of a Kernel package a `diff` file will be saved in `src/fbr/buildroot/linux/linux-<version>/linux-<version>_<variante>/dot-config-<arch>.diff` instead.

1. Documentation For Package SRC

the changes to the SVN repository will be merged and the problem of lost configuration does not occur.) However, your own FBR packages may be reconfigured easily, without data loss occurring on an update.

1.1.8. Integrating Own Programs Into The FBR

Compilation of the individual FBR packages is controlled by small Makefiles. If you want to develop your own FBR packages, you have to create a Makefile and a configuration description in `~/.fbr/own/<package>/`. How these are constructed and how to write own Makefiles derived from them is described in detail in the documentation for the uClibc Buildroot <http://buildroot.uclibc.org/downloads/manual/manual.html#adding-packages>.

A. Appendix For Package SRC