

# **Paket SRC**

## **Version 4.0.0-trunk-rpi-r60747**

Christoph Schulz                      Das fli4l-Team  
E-Mail: [fli4l@kristov.de](mailto:fli4l@kristov.de)      E-Mail: [team@fli4l.de](mailto:team@fli4l.de)

15. September 2022

# Inhaltsverzeichnis

<b>1. Dokumentation des Paketes SRC</b>	<b>3</b>
1.1. SRC - Das fli4l-Buildroot . . . . .	3
1.1.1. Eine Übersicht über die Quellen . . . . .	3
1.1.2. Übersetzen eines Programms für den fli4l . . . . .	4
1.1.3. Testen eines übersetzten Programms . . . . .	7
1.1.4. Entwanzen eines übersetzten Programms . . . . .	8
1.1.5. Informationen über das FBR . . . . .	11
1.1.6. Ändern der FBR-Konfiguration . . . . .	12
1.1.7. Aktualisierung des FBRs . . . . .	13
1.1.8. Eigene Programme ins FBR einbinden . . . . .	13
<b>A. Anhang zum Paket SRC</b>	<b>14</b>

# 1. Dokumentation des Paketes SRC

## 1.1. SRC - Das fli4l-Buildroot

Dieses Kapitel ist hauptsächlich für Entwickler interessant, die Binärprogramme oder Linux-Kernel für den fli4l übersetzen wollen. Wenn Sie fli4l nur als Router einsetzen und keine Pakete für den fli4l anbieten wollen, die eigene Binärprogramme benötigen, können Sie dieses Kapitel komplett überspringen.

Generell ist für die Übersetzung von Programmpaketen für den fli4l ein Linux-System erforderlich. Eine Übersetzung unter anderen Betriebssystemen (Microsoft Windows, OS X, FreeBSD etc.) wird *nicht* unterstützt.

Die Anforderungen an ein Linux-System zur fli4l-Entwicklung sind wie folgt:

- GNU gcc und g++ in der Version 2.95 oder neuer
- GNU gcc-multilib (je nach Hostsystem nötig)
- GNU binutils (enthält den Binder sowie andere, notwendige Programme)
- GNU make in der Version 3.81 oder neuer
- GNU bash
- libncurses5-dev für `fbr-make *-menuconfig` (je nach Hostsystem nötig)
- die Programme sed, awk, which, flex, bison und patch
- die Programme makeinfo (Paket texinfo) und msgfmt (Paket gettext)
- die Programme tar, cpio, gzip, bzip2 und unzip
- die Programme wget, rsync, svn und git
- die Programme perl und python

Im Folgenden repräsentieren **fett** gedruckte Zeichen zu tätigende Eingaben, das ↵-Zeichen steht für die Eingabetaste Ihrer Tastatur und schließt einzugebende Befehle ab.

### 1.1.1. Eine Übersicht über die Quellen

Im `src`-Verzeichnis finden Sie folgende Unterverzeichnisse:

Verzeichnis	Inhalt
<b>fbr</b>	In diesem Verzeichnis befindet sich ein angepasstes Buildsystem, das auf dem Buildroot zur uClibc (aktuell in der Version 0.9.33.2) basiert. FBR steht hierbei für “fli4l-Buildroot”. Damit ist es möglich, alle auf dem fli4l verwendeten Programme (Kernel, Anwendungen und Bibliotheken) neu zu übersetzen.
<b>fli4l</b>	Dieses Verzeichnis enthält die fli4l-spezifischen Quellen, nach Paketen geordnet. Alle Quellen, die in diesem Unterverzeichnis enthalten sind, wurden entweder speziell für die Verwendung mit fli4l geschrieben oder zumindest stark angepasst.
<b>cross</b>	In diesem Verzeichnis befinden sich Skripte, mit denen die Cross-Compiler erstellt werden können, die für das Übersetzen von mkfli4l für diverse Plattformen benötigt werden.

### 1.1.2. Übersetzen eines Programms für den fli4l

Im Unterverzeichnis “fbr” finden Sie das Skript **fbr-make**, das die Übersetzung aller Programme aus den Basispaketen für den fli4l-Router steuert. Dieses Skript kümmert sich um das Herunterladen und Übersetzen aller für den fli4l benötigten Binärdateien. Generell legt das Skript Dateien in dem Verzeichnis `~/fbr` ab; existiert dieses noch nicht, wird es angelegt. (Dieses Verzeichnis kann mit Hilfe der Umgebungsvariable `FBR_BASEDIR` verändert werden, siehe unten.)

*Hinweis:* Während des Übersetzungsvorgangs wird viel Platz benötigt (momentan etwa 900 MiB für die heruntergeladenen Archive und knapp 30 GiB für die Zwischenergebnisse und die resultierenden Kompilate). Stellen Sie somit sicher, dass Sie unterhalb von `~/fbr` über genug Platz verfügen! (Alternativ können Sie auch die `FBR_TIDY`-Option nutzen, siehe unten.)

Die Verzeichnisstruktur unterhalb von `~/fbr` ist wie folgt:

Verzeichnis	Inhalt
<b>fbr-&lt;branch&gt;-&lt;arch&gt;</b>	Hierhin wird das uClibc-Buildroot entpackt. <b>&lt;branch&gt;</b> steht hierbei für den Entwicklungszweig (z.B. <b>trunk</b> ), aus dem das FBR stammt. Ist der Ursprung des FBRs ein entpacktes <b>src</b> -Paket, wird <b>fbr-custom</b> benutzt. <b>&lt;arch&gt;</b> steht für die jeweilige Prozessorarchitektur (z.B. <b>x86</b> oder <b>x86_64</b> ). Mehr zu diesem Verzeichnis steht weiter unten.
<b>dl</b>	Hier werden die heruntergeladenen Archive gespeichert.
<b>own</b>	Hier können eigene FBR-Pakete abgelegt werden, die ebenfalls übersetzt werden sollen.

Unterhalb des Buildroot-Verzeichnisses `~/fbr/fbr-<branch>-<arch>/buildroot` sind die folgenden Verzeichnisse interessant:

Verzeichnis	Inhalt
output/sandbox	In diesem Verzeichnis gibt es für jedes FBR-Paket ein Unterverzeichnis, das die Dateien des FBR-Pakets nach dem Übersetzungsvorgang aufnimmt. In dem Verzeichnis <code>output/sandbox/&lt;paket&gt;/target</code> befinden sich dabei die Dateien, die für den fli4l-Router vorgesehen sind. In dem Verzeichnis <code>output/sandbox/&lt;paket&gt;/staging</code> hingegen befinden sich Dateien, die zum Übersetzen <i>anderer</i> FBR-Pakete, die dieses FBR-Paket benötigen, erforderlich sind.
output/target	In diesem Verzeichnis werden <i>alle</i> übersetzten Programme für den fli4l-Router abgelegt. Dieses Verzeichnis spiegelt somit die Verzeichnisstruktur auf dem fli4l-Router wider. Mit Hilfe von <code>chroot</code> kann man in dieses Verzeichnis wechseln und die übersetzten Programme ausprobieren. <sup>1</sup>

## Allgemeine Einstellungen

Die Arbeitsweise von `fbr-make` kann durch verschiedene Umgebungsvariablen beeinflusst werden:

Variable	Beschreibung
FBR	Gibt den Pfad zum FBR explizit an. Standardmäßig wird der Pfad <code>~/fbr/fbr-&lt;branch&gt;--&lt;arch&gt;</code> (s.o.) verwendet.
FBR_BASEDIR	Gibt den Basispfad zum FBR explizit an. Standardmäßig wird der Pfad <code>~/fbr</code> (s.o.) verwendet. Diese Variable wird ignoriert, falls die Umgebungsvariable <code>FBR</code> gesetzt wird.
FBR_DLDIR	Gibt das Verzeichnis an, das die Quellarchive enthält. Standardmäßig wird der Pfad <code>\${FBR}/../dl</code> (also z.B. <code>~/fbr/dl</code> ) verwendet.
FBR_BRANCH	Gibt explizit den Namen des Branches an, unter dem die Pakete unterhalb von <code>~/fbr</code> (s.o.) gebaut werden. Diese Variable wird ignoriert, falls die Umgebungsvariable <code>FBR</code> gesetzt wird.
FBR_CATEGORY	Gibt explizit den Namen der Category an, unter dem die Pakete unterhalb von <code>~/fbr</code> (s.o.) gebaut werden. Diese Variable wird ignoriert, falls die Umgebungsvariable <code>FBR</code> gesetzt wird.
FBR_OWNDIR	Gibt das Verzeichnis an, das die eigenen Pakete enthält. Standardmäßig wird der Pfad <code>\${FBR}/../own</code> (also z.B. <code>~/fbr/own</code> ) verwendet.

<sup>1</sup>Dies ist an gewisse Voraussetzungen geknüpft, siehe hierzu den Abschnitt [“Testen eines übersetzten Programms”](#) (Seite 7).

Variable	Beschreibung
FBR_TIDY	Wenn diese Variable den Wert “y” enthält, werden Zwischenergebnisse, die während des Bauens der FBR-Pakete entstehen, unmittelbar nach der Installation in das Verzeichnis <code>output/target</code> gelöscht. Das spart viel Speicherplatz und ist eigentlich immer empfehlenswert, wenn man nach dem Bauen von Paketen nicht den Drang verspürt, in <code>output/build/...</code> hineinzuschauen. Falls diese Variable den Wert “k” enthält, werden nur die Zwischenergebnisse in den diversen Linux-Kernel-Verzeichnissen entfernt, weil dies verhältnismäßig viel Platz spart, ohne dass dadurch Funktionalität verloren geht. Alle anderen Belegungen (oder wenn die Variable gänzlich fehlt) sorgen dafür, dass alle Zwischenergebnisse erhalten bleiben.
FBR_ARCH	Diese Variable gibt die Prozessorarchitektur an, für die das FBR (bzw. einzelne FBR-Pakete) gebaut werden sollen. Fehlt sie, wird <code>x86</code> angenommen. Die unterstützten Architekturen sind weiter unten zu finden.

Momentan unterstützt das FBR die folgenden Architekturen:

Architektur	Beschreibung
x86	Intel x86-Architektur (32-Bit), auch IA-32 genannt.
x86_64	AMD x86-64-Architektur (64-Bit), von Intel auch Intel 64 oder EM64T genannt.

### Übersetzen aller FBR-Pakete

Wenn Sie `fbr-make` mit dem Argument `world` ausführen, dauert es je nach verwendetem Rechner und verwendeter Internetanbindung mehrere Stunden, bis alle Quellarchive heruntergeladen und übersetzt worden sind.<sup>2</sup>

### Übersetzen des Toolchains

Wenn Sie `fbr-make` mit dem Argument `toolchain` ausführen, werden alle FBR-Pakete heruntergeladen und übersetzt, die für das Bauen der eigentlichen fli4l-Binärprogramme benötigt werden (also Übersetzer, Binder, uClibc-Bibliothek etc.). Normalerweise wird dieses Kommando nicht benötigt, da alle FBR-Pakete vom Toolchain abhängig sind und somit diese Toolchain-Programme ohnehin heruntergeladen und gebaut werden.

### Übersetzen eines einzelnen FBR-Pakets

Wollen Sie hingegen nur ein bestimmtes FBR-Paket übersetzen (etwa die Programme für ein selbst entwickeltes OPT), können Sie den Namen des FBR-Pakets bzw. die Namen mehrerer

---

<sup>2</sup>Das Herunterladen der Quellarchive wird natürlich nur einmal durchgeführt, so lange Sie das FBR nicht aktualisieren und dadurch neuere Paketversionen andere Quellarchive benötigen.

FBR-Pakete dem **fbr-make**-Programm mitgeben (etwa **fbr-make openvpn** zum Herunterladen und Übersetzen der OpenVPN-Programme). Dabei werden alle nötigen Abhängigkeiten ebenfalls heruntergeladen und übersetzt.

### Erneutes Übersetzen eines einzelnen FBR-Pakets

Möchten Sie ein bestimmtes FBR-Paket erneut übersetzen (warum auch immer), müssen Sie zuerst die Informationen im FBR über den vorher stattgefundenen Übersetzungsvorgang entfernen. Dazu können Sie den Befehl **fbr-make <paket>-clean** (z.B. **fbr-make openvpn-clean**) verwenden. Dabei werden die Informationen all jener FBR-Pakete, die von dem angegebenen FBR-Paket abhängig sind, ebenfalls zurückgesetzt, so dass sie beim nächsten **fbr-make world** ebenfalls neu übersetzt werden.

### Erneutes Übersetzen aller FBR-Pakete

Möchten Sie das komplette FBR neu übersetzen (z.B. weil Sie es als Benchmark-Programm für Ihr neues High-End-Entwicklersystem nutzen wollen ;-), können Sie mit Hilfe des Kommandos **fbr-make clean** nach der Bestätigung einer Sicherheitsabfrage alle Artefakte entfernen, die während des letzten FBR-Baus erzeugt worden sind.<sup>3</sup> Dies ist auch nützlich, um belegten Plattenspeicher freizugeben.

#### 1.1.3. Testen eines übersetzten Programms

Ist ein Programm mit **fbr-make** übersetzt worden, kann es auf dem Entwicklungsrechner auch getestet werden. Ein solcher Test funktioniert natürlich nur, wenn die Prozessorarchitektur des Entwicklerrechners mit der Prozessorarchitektur des fli4ls, für welchen die Programme übersetzt werden sollen, übereinstimmt. (Es ist z.B. nicht möglich, **x86\_64-fli4l**-Programme auf einem **x86**-Betriebssystem auszuführen.) Ist diese Voraussetzung erfüllt, kann man mit

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

in das fli4l-Zielverzeichnis wechseln und dort das/die übersetzte(n) Programm(e) direkt ausprobieren. Beachten Sie jedoch bitte, dass Sie für **chroot** Administrator-Rechte benötigen und daher je nach Vorliebe und Systemkonfiguration die Dienste von **sudo** oder **su** in Anspruch nehmen müssen! Auch müssen Sie zumindest das FBR-Paket **busybox** übersetzt haben (via **fbr-make busybox**), damit Sie in der **chroot**-Umgebung eine Shell zur Verfügung haben. Ein kleines Beispiel:

```
$ sudo chroot ~/.fbr/fbr-trunk-x86/buildroot/output/target /bin/sh
Passwort:(Ihr Passwort)
```

```
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
```

---

<sup>3</sup>Es wird das gesamte Verzeichnis `~/.fbr/fbr-<branch>-<arch>/buildroot/output` entfernt.

```
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. Entwanzen eines übersetzten Programms

Macht ein Programm auf dem fli4l Probleme, mit anderen Worten: es stürzt ab, dann hat man die Möglichkeit, den Programmzustand unmittelbar vor dem Absturz nachträglich zu analysieren (auch “Post-Mortem Debugging” genannt). Hierzu muss man zuerst in der Konfiguration des base-Pakets `DEBUG_ENABLE_CORE='yes'` aktivieren. Wird dann beim Absturz ein Speicherabbild in `/var/log/dumps/core.<PID>` generiert, wobei “PID” die Prozessnummer des abgestürzten Prozesses ist, dann kann man den Zustand des Programms auf einem Linux-Rechner mit einem voll übersetzten FBR folgendermaßen analysieren. Im folgenden Beispiel ist das zu analysierende Programm `/usr/sbin/collectd`, das sich mit einem SIGBUS verabschiedet hatte. Das Speicherabbild liegt dabei 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-buildr
oot-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/ouput
/debug↵
(gdb) file /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/usr/sbin/co
llectd↵
Reading symbols from /project/fli4l/.fbr/fbr-trunk-x86/buildroot/output/target/u
sr/sbin/collectd...Reading symbols from /project/fli4l/.fbr/fbr-trunk-x86/buildr
oot/output/debug/.build-id/8b/28ab573be4a2302e1117964edede2e54ebdbbf.debug...don
e.
done.
(gdb) core /tmp/core.collectd↵
[New LWP 2250]
[New LWP 2252]
```



## 1. Dokumentation des Paketes SRC

```
[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]
[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);
717     rrd_file->pos += count;
718     return count;        /* mimmic 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}

```

Hier sieht man nach etwas “Wühlen”, dass sich in dem `rrd_simple_file_t`-Objekt ein ungültiger Zeiger befindet (“Address ... out of bounds”). Im weiteren Debugging-Verlauf wurde deutlich, dass ein gescheiterter `posix_fallocate`-Aufruf die Ursache für den Programmabsturz war.

Wichtig hierbei ist, dass *alle* anzugebenden Pfade voll qualifiziert sind (`/project/...`) und dass man auch keine “Abkürzungen” (etwa `~/...`) verwendet. Wenn man dies nicht beachtet, kann es passieren, dass `gdb` die Debug-Informationen zur Anwendung und/oder zu den verwendeten Bibliotheken nicht findet. Die Debug-Informationen sind nämlich aus Platzgründen nicht direkt in dem untersuchten Programm enthalten, sondern in einer separaten Datei unterhalb

des Verzeichnisses `~/.fbr/fbr-<branch>-<arch>/buildroot/output/debug/` gespeichert.

### 1.1.5. Informationen über das FBR

#### Anzeige der Hilfe

Was `fbr-make` alles für Sie tun kann, können Sie sich mit dem Kommando `fbr-make help` ausgeben lassen.

#### Anzeige von Programminformationen

Sie können sich alle verfügbaren FBR-Pakete sowie deren Versionen anschauen, indem Sie das Kommando `fbr-make show-versions` nutzen:

```
$ fbr-make show-versions␣  
Configured packages  
  
acpid 2.0.20  
actctrl 3.25+dfsg1  
add-days undefined  
[...]
```

#### Anzeige von Bibliotheksabhängigkeiten

Mit Hilfe des Kommandos `fbr-make links-against <soname>` können Sie sich alle Dateien in `~/.fbr/fbr-<branch>-<arch>/buildroot/output/target` anzeigen lassen, die gegen eine Bibliothek mit dem Bibliotheksnamen `soname` gebunden sind. Um beispielsweise alle Programme und Bibliotheken zu identifizieren, welche die `libm` (Bibliothek mit mathematischen Funktionen) verwenden, nutzen Sie das Kommando `fbr-make links-against libm.so.0`, da `libm.so.0` der Bibliotheksname der `libm`-Bibliothek ist. Eine mögliche Ausgabe ist:

```
$ 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
```

Dabei steht in der ersten Spalte der Paketname und in der zweiten der (relative) Pfad zu der Datei, die gegen die angegebene Bibliothek gebunden ist.

Um den Bibliotheksnamen für eine Bibliothek herauszufinden, können Sie `readelf` wie folgt nutzen:

```
$ 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]
```

#### Anzeige von Versionsänderungen

(Nur) für fli4l-Team-Entwickler mit Schreibzugriff auf das fli4l-SVN-Repository ist das Kommando `fbr-make version-changes` interessant. Es listet alle FBR-Pakete auf, deren Version lokal modifiziert wurde, deren Version in der Arbeitskopie also von der Repository-Version

abweicht. Damit kann der Entwickler sich einen Überblick über aktualisierte FBR-Pakete verschaffen, etwa bevor er die Änderungen ins Repo schreibt. Eine mögliche Ausgabe ist:

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

Hier sieht man sofort, dass das kamailio-FBR-Paket von der Version 4.0.5 auf die Version 4.1.1 aktualisiert worden ist.

### 1.1.6. Ändern der FBR-Konfiguration

#### Rekonfiguration des FBRs

Mit Hilfe des Kommandos `fbr-make buildroot-menuconfig` ist es zum einen möglich, die zu übersetzenden FBR-Pakete auszuwählen. Das ist nützlich, wenn Sie andere FBR-Pakete für den fli4l übersetzen möchten, die standardmäßig nicht aktiviert sind, aber im uClibc-Buildroot integriert sind, oder wenn Sie eigene FBR-Pakete aktivieren wollen. Zum anderen können andere, globale Eigenschaften des FBRs verändert werden, etwa die Version des verwendeten GCC-Compilers. Beim erfolgreichen Beenden des Konfigurationsmenüs wird die neue Konfiguration im Verzeichnis `src/fbr/buildroot/.config` gespeichert.

**Beachten Sie jedoch bitte, dass solche Änderungen der Toolchain-Konfiguration offiziell *nicht* unterstützt werden, weil die resultierenden Binärprogramme mit hoher Wahrscheinlichkeit inkompatibel mit der offiziellen fli4l-Distribution werden. Wenn Sie also Binärprogramme für Ihr eigenes OPT benötigen und dieses OPT veröffentlichen wollen, sollten Sie keine Toolchain-Einstellung verändern!**

#### Rekonfiguration der uClibc-Bibliothek

Mit dem Kommando `fbr-make uclibc-menuconfig` kann die Funktionalität der verwendeten uClibc-Bibliothek verändert werden. Beim erfolgreichen Beenden des Konfigurationsmenüs wird die neue Konfiguration in `src/fbr/buildroot/package/uclibc/uclibc.config` gespeichert.

**Wie im letzten Abschnitt gilt auch hier: Eine Änderung ist mit hoher Wahrscheinlichkeit nicht kompatibel mit der offiziellen fli4l-Distribution und wird somit nicht unterstützt!**

#### Rekonfiguration der Busybox

Mit Hilfe des Kommandos `fbr-make busybox-menuconfig` kann die Busybox in ihrer Funktionalität angepasst werden. Beim erfolgreichen Beenden des Konfigurationsmenüs wird die neue Konfiguration in `src/fbr/buildroot/package/busybox/busybox-<Version>.config` gespeichert.

**Auch hier gilt: Eine Änderung ist höchstwahrscheinlich nicht kompatibel mit der offiziellen fli4l-Distribution und wird somit nicht unterstützt! Allenfalls das**

Ergänzen der Busybox um neue Applets ist problemlos, solange Sie die so modifizierte Busybox nur auf Ihren fli4l-Routern einsetzen (und nicht vom Nutzer Ihres OPTs den Einsatz einer derart modifizierten Busybox verlangen).

### Rekonfiguration der Linux-Kernelpakete

Mit `fbr-make linux-menuconfig` bzw. `fbr-make linux-<version>-menuconfig` kann die Konfiguration aller aktivierten Kernel-Pakete bzw. eines bestimmten Kernel-Pakets vorgenommen werden. Beim erfolgreichen Beenden des Konfigurationsmenüs wird die neue Konfiguration in `src/fbr/buildroot/linux/linux-<version>/dot-config-<arch>` gespeichert.<sup>4</sup>

Wie im letzten Abschnitt gilt auch hier: Eine Änderung ist höchstwahrscheinlich nicht kompatibel mit der offiziellen fli4l-Distribution und wird somit nicht unterstützt! Allenfalls das Ergänzen des Linux-Kernels um neue Module ist problemlos, solange Sie den so modifizierten Kernel nur auf Ihren fli4l-Routern einsetzen (und nicht vom Nutzer Ihres OPTs den Einsatz eines derart modifizierten Kernels verlangen).

#### 1.1.7. Aktualisierung des FBRs

Jedem der beschriebenen Kommandos geht eine Prüfung des FBRs auf Aktualität voraus. Wird eine Diskrepanz zwischen den Quellen, in denen sich `fbr-make` befindet (also entpacktes `src`-Paket oder SVN-Arbeitskopie) und dem FBR in `~/.fbr/fbr-<branch>-<arch>/buildroot` entdeckt, wird letzteres auf den neuesten Stand gebracht. Dabei werden neu dazugekommene FBR-Pakete integriert sowie alte, nicht mehr enthaltene FBR-Pakete gelöscht. Auch die Konfigurationen werden verglichen: FBR-Pakete mit veränderter Konfiguration sowie alle davon abhängigen FBR-Pakete werden neu gebaut. Dies stellt sicher, dass das FBR auf Ihrem Computer immer dem der fli4l-Entwickler entspricht (mit Ausnahme Ihrer eigenen FBR-Pakete unterhalb von `~/.fbr/own/`). **Das bedeutet jedoch auch, dass Änderungen am offiziellen Teil der Buildroot-Konfiguration bei der nächsten Aktualisierung verloren gehen!** Auch deshalb ist eine Rekonfiguration des FBRs (s.o.) somit nicht zu empfehlen, zumindest nicht, wenn Sie mit `src`-Paketen anstatt mit SVN-Arbeitskopien arbeiten. (Bei der Aktualisierung einer SVN-Arbeitskopie werden Ihre lokalen Konfigurationsänderungen und die Änderungen im SVN-Repository zusammengeführt, so dass das Problem der verlorenen Konfiguration hier nicht auftritt.) Hingegen können Sie Ihre eigenen FBR-Pakete problemlos umkonfigurieren, ohne dass es bei einer Aktualisierung zu Datenverlusten kommt.

#### 1.1.8. Eigene Programme ins FBR einbinden

Die Übersetzung der einzelnen FBR-Pakete wird über kleine Makefiles gesteuert. Will man eigene FBR-Pakete entwickeln, so muss man ein Makefile sowie eine Konfigurationsbeschreibung unter `~/.fbr/own/<paket>/` ablegen. Wie diese aufgebaut sind und wie man daraus abgeleitet eigene Makefiles schreiben kann, wird in der Dokumentation des uClibc-Buildroots unter <http://buildroot.uclibc.org/downloads/manual/manual.html#adding-packages> ausführlich beschrieben.

---

<sup>4</sup>Dies gilt nur für den Standard-Kernel. Für Varianten eines Kernelpakets wird stattdessen eine `diff`-Datei in `src/fbr/buildroot/linux/linux-<version>/linux-<version>_<variante>/dot-config-<arch>.diff` abgelegt.

## **A. Anhang zum Paket SRC**