

Paquetage SRC

Version 4.0.0-testing-x86-r60742

Christoph Schulz
courriel: fli4l@kristov.de

L'équipe fli4l
courriel: team@fli4l.de

7 septembre 2022

Table des matières

1. Documentation du packaging SRC	3
1.1. SRC - Le Buildroot fli4l	3
1.1.1. Vue d'ensemble des répertoires sources	3
1.1.2. Compiler un programme pour fli4l	4
1.1.3. Tester d'un programme compilé	7
1.1.4. Déboguer un programme compilé	8
1.1.5. Information sur le FBR	11
1.1.6. Modification de la configuration du FBR	12
1.1.7. Mise à jour des FBRs	13
1.1.8. Intégrer vos propres programmes dans le FBR	13
A. Annexe du packaging SRC	14
A.1. SRC - Développement de son propre packaging	14
Table des figures	15
Liste des tableaux	16

1. Documentation du paquetage SRC

marklabelbuildroot

1.1. SRC - Le Buildroot fli4l

Ce chapitre est intéressant, uniquement pour les développeurs qui veulent compiler des programmes binaires ou le Kernel Linux pour fli4l. Si vous utilisez fli4l seulement comme routeur et si vous ne voulez pas donner au routeur fli4l des opt-packages (ou paquetage optionnel) ou créer vos propre programmes binaires, vous pouvez ignorer complètement ce chapitre.

En général, un système Linux est nécessaire pour la création de progiciels pour fli4l. La création sur d'autres systèmes d'exploitation (Microsoft Windows, OS X, FreeBSD etc.) ne sera *pas* support.

Les exigences du système Linux pour le développement de fli4l sont les suivantes :

- GNU gcc et g++ dans la version 2.95 ou supérieur
- GNU gcc-multilib (en fonction de votre système)
- GNU binutils (y compris Binder ainsi que d'autres programmes sont nécessaires)
- GNU make dans la Version 3.81 ou supérieur
- GNU bash
- libncurses5-dev pour **fbr-make *-menuconfig** (en fonction de votre système)
- Les programmes sed, awk, which, flex, bison et patch
- Les programmes makeinfo (paquet texinfo) et msgfmt (paquet gettext)
- Les programmes tar, cpio, gzip, bzip2 et unzip
- Les programmes wget, rsync, svn et git
- Les programmes perl et python

Dans la documentation ci-dessous les caractères en **gras** indiquent une commande, le caractère ↵ représente la touche Entrée de votre clavier et ferme la commande.

1.1.1. Vue d'ensemble des répertoires sources

Dans le répertoire **src**, vous trouverez les sous-répertoires suivants :

Répertoire	Contenu
fbr	Ce répertoire contient le système de construction personnalisable qui est basé sur le buildroot uClibc (avec actuellement la version 0.9.33.2). Le FBR signifie "fli4l Buildroot". Il est ainsi possible de recompiler tous les programmes fli4l (le Kernel, les bibliothèques et les applications).
fli4l	Ce répertoire contient les sources spécifiques à fli4l, triées par paquets. Toutes les sources incluses dans les sous-répertoires ont été écrites spécifiquement pour l'utilisation de fli4l ou du moins fortement personnalisées.

Répertoire	Contenu
cross	Ce répertoire contient les scripts nécessaires pour la compilation croisée, ils permettent de créer et de compiler le mkfli4l avec les différentes plates-formes.

1.1.2. Compiler un programme pour fli4l

Dans le répertoire "fbr" vous avez le script **fbr-make** qui contrôle la compilation de tous les programmes du paquet base du routeur fli4l. Ce script se charge de télécharger et de compiler tous les fichiers binaires requis pour fli4l. En règle générale, les fichiers script définient sont placés dans le répertoire `~/fbr` s'il n'existe pas encore, il sera créé. (Ce répertoire peut être modifié à l'aide de la variable d'environnement `FBR_BASEDIR`, voir ci-dessous.)

Précision : la quantité d'espace nécessaire pour le processus de compilation est (actuellement environ de 900 Mio pour les archives téléchargées et à peu près de 30 Gio avec les résultats intermédiaires en réalisant une compilation). Assurez-vous que dans le répertoire `~/fbr` vous avez suffisamment d'espace! (Sinon, vous pouvez également utiliser l'option `FBR_TIDY`, voir ci-dessous).

La structure du répertoire `~/fbr` est la suivante :

Répertoire	Contenu
fbr-<code><branch></code>-<code><arch></code>	Dans ce répertoire, le buildroot uClibc sera décompacté. <code><branch></code> est la branche de développement (par exemple <code>trunk</code>), à partir du quelle le FBR provient. A l'origine le FBR est un paquet <code>src</code> décompacté, qui était utilisé pour personnalisé le fbr . <code><arch></code> est l'architecture de processeur (par exemple <code>x86</code> ou <code>x86_64</code>). En plus de ce répertoire.
dl	Les archives téléchargées sont stockées ici.
own	Les paquets FBR peuvent être stockées ici, ils seront également compilés.

Ci-dessous le répertoire Buildroot `~/fbr/fbr-<branch>-<arch>/buildroot` les répertoires suivants sont intéressants :

Répertoire	Contenu
output/sandbox	Dans ce répertoire il y aura un sous-répertoire pour chaque paquet FBR, celui-ci contiendra les fichiers du paquet FBR après le processus de compilation. Dans le répertoire <code>output/sandbox/<code><paquet></code>/target</code> seront placé les fichiers qui sont prévus pour le routeur fli4l. Dans le répertoire <code>output/sandbox/<code><paquet></code>/staging</code> seront placé les fichiers qui sont nécessaires pour convertir <i>d'autre</i> paquet FBR qui ont besoin du paquet FBR principale.

Répertoire	Contenu
output/target	Dans ce répertoire, <i>tous</i> les programmes stockés seront compilés pour le routeur fli4l. Cette liste reflète la structure des répertoires sur le routeur fli4l. Avec l'aide de la commande chroot vous pouvez changer ce répertoire et tester les programmes compilés ¹

Paramètres généraux

Pour l'utilisation de la commande **fbr-make** vous pouvez affecter plusieurs variables d'environnement :

Variable	Description
FBR	Indique explicitement le chemin d'accès au FBR. L'utilisation du chemin par défaut est <code>~/fbr/fbr-<branch>-<arch></code> (voir ci-dessus).
FBR_BASEDIR	Indique explicitement le chemin d'accès au FBR. Par défaut, le chemin <code>~/fbr</code> est utilisé (voir ci-dessus). Cette variable est ignorée si la variable d'environnement FBR est définie.
FBR_DLDIR	Indique le répertoire qui contient les archives sources. L'utilisation du chemin par défaut est <code>\${FBR}/../dl</code> (par exemple <code>~/fbr/dl</code>).
FBR_BRANCH	Spécifie explicitement le nom de la branche sous laquelle les paquets sont construits en dessous de <code>~/fbr</code> (voir plus haut). Cette variable est ignorée si la variable d'environnement FBR est définie.
FBR_CATEGORY	Spécifie explicitement le nom de la catégorie sous laquelle les paquets sont construits en dessous de <code>~/fbr</code> (voir plus haut). Cette variable est ignorée si la variable d'environnement FBR est définie.
FBR_OWNDIR	Indique le répertoire qui contient les paquets spécifiques. L'utilisation du chemin par défaut est <code>\${FBR}/../own</code> (par exemple <code>~/fbr/own</code>).

1. cette fonction est soumise à certaines conditions, se référer au paragraphe ["tester un programme compilé"](#) (Page 7).

Variable	Description
FBR_TIDY	Si cette variable contient la valeur "y", les résultats intermédiaires qui apparaissent pendant la compilation des paquets FBR seront effacés directement après l'installation du répertoire <code>output/target</code> . Cela permet d'économiser beaucoup d'espace, en faite cette valeur est toujours recommandé, si vous ne vous sentez pas l'envie de vérifier les répertoires <code>output/build/...</code> après la construction des paquetages. Si cette variable contient la valeur "k", c'est seulement les résultats intermédiaires dans les différents répertoires du Kernel de Linux qui seront effacés, cela permet d'économiser relativement beaucoup d'espace sans perdre les fonctionnalités. Tous les autres déplacement (ou si la variable est manquante) sont effectuées et tous les résultats intermédiaires seront conservés.
FBR_ARCH	Cette variable spécifie l'architecture du processeur pour lequel le FBR (ou les paquets FBR individuels) doit être construit. Si elle est absente le <code>x86</code> sera utilisé pour la construction. Voir ci-dessous les architectures supportées.

Le FBR prend actuellement en charge les architectures suivantes :

Architecture	Description
<code>x86</code>	Intel Architecture x86 (32-Bit), aussi connu sous le nom IA-32.
<code>x86_64</code>	AMD Architecture x86-64 (64-Bit), appelé aussi Intel 64 ou EM64T.

Compiler tous les paquets FBR

Lorsque vous exécutez la commande `fbr-make` avec l'argument `world`, pour que toutes les archives sources soit téléchargés et compilées, l'ordinateur devra être utilisé pendant plusieurs heures et ceci en fonction de l'ordinateur et le type de connexion Internet.²

Compiler avec Toolchain

Lorsque vous exécutez la commande `fbr-make` avec l'argument `toolchain`, tous les paquets FBR sont téléchargés et convertis, ils sont nécessaires pour construire les fichiers binaires réels pour fli4l (c-à-d construire Binder, la bibliothèque uClibc etc.). Normalement, cette commande n'est pas nécessaire car tous les paquets FBR dépendante du toolchain, les programmes toolchain téléchargés et sont de toute façon compilés.

2. Le téléchargement des archives source sera bien entendu effectuée qu'une seule fois, tant que vous ne mettez pas à jour le FBR, si vous le mettez à jour vous aurez besoin de nouvelles versions de paquets avec d'autres archives source.

Compiler un unique paquet FBR

Si vous voulez compiler seulement un certain paquet FBR (pour auto-développé un programme OPT), vous pouvez indiquer le nom du paquet FBR ou le nom de plusieurs paquets FBR avec la commande **fbr-make**, (vous pouvez indiquer **fbr-make openvpn** pour télécharger et compiler le programme openVPN). Toutes les fichiers nécessaires qui dépende du programme seront également téléchargés et compilés.

Recompiler un unique paquet FBR

Si vous voulez recompiler un certain paquet FBR (pour une raison quelconque), vous devez d'abord supprimer les informations du processus de compilation dans le FBR avant de recommencer. Vous pouvez utiliser la commande **fbr-make <paquet>-clean** (par ex. **fbr-make openvpn-clean**). Les informations de tous les paquets FBR qui dépendent du paquet FBR spécifié seront également réinitialisées, de sorte qu'ils pourront également être recompilés la prochaine fois avec **fbr-make world**.

Recompiler tous les paquets FBR

Si vous souhaitez recompiler complètement le FBR (par exemple parce que vous voulez l'utiliser un programme de référence pour développer votre nouveau système haut de gamme ;-). Vous pouvez tous supprimer en utilisant la commande **fbr-make clean** après avoir été invité à confirmé la commande, tous les artefacts qui ont été générés au cours de la dernière construction FBR seront supprimés.³ Cela est également utile pour libérer de l'espace disque.

1.1.3. Tester d'un programme compilé

Si un programme a été compilé avec **fbr-make**, il peut également être testé sur l'ordinateur de développement. Un tel test ne fonctionne que lorsque l'architecture du processeur de l'ordinateur de développement et l'architecture du processeur pour fli4l pour laquelle les programmes ont être compilés, correspondent. (Par exemple, il n'est pas possible d'exécuter les programmes fli4l x86_64 sur un système d'exploitation x86). Si la condition est remplie, vous pouvez faire un teste,

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

allez dans le répertoire cible fli4l et essayez directement avec le/les programme(s) compilé(s). S'il vous plaît faites attention, vous avez besoin pour utiliser **chroot** des droits administrateur et en fonction de vos préférences et de la configuration du système vous devez utiliser le service **sudo** ou **su** c'est une exigence! Vous devez aussi, avoir compilé dans le paquet FBR **busybox** (via **fbr-make busybox**) de sorte de pouvoir disposer d'un environnement shell dans le répertoire **chroot**. Voici un petit exemple :

```
$ 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.
```

3. Tout les répertoires `~/.fbr/fbr-<branch>-<arch>/buildroot/output` seront supprimés.

```
# 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. Déboguer un programme compilé

Si vous avez un problème sur un programme `fli4l`, en d'autres termes : s'il se bloque, vous avez la possibilité d'analyser l'état du programme plus tard, juste avant l'accident (aussi nommé "débogage post-mortem"). Pour cela il est d'abord nécessaire de configurer le paquet `base` et d'activer `DEBUG_ENABLE_CORE='yes'`. Lors du crash un vidage de la mémoire est généré dans `/var/log/dumps/core.<PID>`, le "PID" est le numéro de processus accidenté, on peut analyser de la manière suivante l'état du programme sur un ordinateur Linux avec un FBR complètement compilé. Dans l'exemple suivant, le programme à analyser est `/usr/sbin/collectd`, avec le SIGBUS accepté. Le vidage de la mémoire est placé dans le fichier `/tmp/core.collectd`.

```
fli4l@eisler:~$ .fbr/fbr-trunk-86/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-86/buildroot/output/target
(gdb) set debug-file-directory /project/fli4l/.fbr/fbr-trunk-86/buildroot/output/debug
(gdb) file /project/fli4l/.fbr/fbr-trunk-86/buildroot/output/target/usr/sbin/collectd
Reading symbols from /project/fli4l/.fbr/fbr-trunk-86/buildroot/output/target/usr/sbin/collectd...Reading symbols from /project/fli4l/.fbr/fbr-trunk-86/buildroot/output/debug/.build-id/8b/28ab573be4a2302e1117964edede2e54ebdbdf.debug...done.
done.
```


1. Documentation du packaging SRC

```
(gdb) core /tmp/core.collected
[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]
[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-86/buildroot/output/target/lib/libc.so.0
(gdb) backtrace
#0  0xb7705f5d in memcpy ()
    from /project/fli4l/.fbr/fbr-trunk-86/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-interrupt.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-interrupt.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.async",
```

1. Documentation du packaging SRC

```
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-86/buildroot/output/target/lib/libpthread.so.
0
#8 0xb775e672 in clone ()
    from /project/fli4l/.fbr/fbr-trunk-86/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}
```

Vous pouvez voir ci-dessus, il faut un peu "fouiller" que dans le répertoire d'objet `rrd_simple_file_t` le pointeur est invalide ("Address ... out of bounds") dans cette suite de débogage, il est clair que l'échec de `posix_fallocate` est la cause de l'interruption du programme.

Ce qui est important ici, c'est que *tous* les chemins indiquer sont pleinement qualifié (`/project/...`) et qu'il n'y a aucun "raccourcis" utilisé (par exemple `~/...`). Si cela n'est pas respecté, il peut arriver que les informations de débogage `gdb` ne trouve pas l'application et/ou n'utilise pas

les bibliothèques. Les informations de débogage ne sont pas directement incluses dans le programme testé, mais stocké dans le répertoire `~/fbr/fbr-<branch>-<arch>/buildroot/output/debug/` sur fichier séparé.

1.1.5. Information sur le FBR

Accéder à l'aide

Que peut faire `fbr-make` pour vous, vous pouvez peut-être utiliser la commande `fbr-make help`.

Affichage des informations sur le programme

Vous pouvez voir tous les paquets FBR disponibles et leurs versions, en utilisant la commande `fbr-make show-versions` :

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

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

Affichage des associations de bibliothèques

Avec la commande `fbr-make links-against <soname>` et en indiquant le nom de la bibliothèque à la place de `soname`, vous pourrez voir tous les fichiers dans le répertoire `~/fbr/fbr-<branch>-<arch>` qui sont associés à cette bibliothèque. Par exemple pour identifier tous les programmes et bibliothèques qui utilisent `libm` (bibliothèque de fonction mathématique), vous indiquez la commande `fbr-make links-against libm.so.0` car le nom de la bibliothèque `libm.so.0` est `libm-bibliothèque`. Un autre exemple :

```
$ 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
```

Dans la première colonne le nom du paquet est indiqué et dans la seconde le chemin (relatif) du fichier qui est associé à la bibliothèque.

Pour trouver le nom d'une bibliothèque, vous pouvez utiliser `readelf`, comme ceci :

```
$ 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]
```

Affichage des changements de version

(Uniquement) intéressant pour les développeurs de l'équipe `fli4l` avec un accès en écriture au répertoire `fli4l-SVN-Repository`, utilisez la commande `fbr-make version-changes`. Vous

pourrez voir la liste de tous les paquets dont la version FBR a été modifié localement, la version différente de la copie de travail et la version du référentiel. Ainsi, le développeur peut voir un aperçu des paquets FBR mis à jour, avant d'écrire les changements dans le repo. Voici une entrée possible :

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

Ici vous pouvez voir immédiatement que le paquet FBR `kamailio` avait la version 4.0.5 et a été mise à jour avec la version 4.1.1.

1.1.6. Modification de la configuration du FBR

Reconfiguration des FBRs

D'une part, à l'aide de la commande `fbr-make buildroot-menuconfig`, il est possible de choisir les paquets FBR à compiler. Ceci est utile si vous voulez compiler d'autres paquets FBR pour `fi4l` et qui ne sont pas activés par défaut, mais qui sont intégrés dans le buildroot `uClibc`, ou si vous souhaitez activer vos propres paquets FBR. D'autre part, des propriétés globales du FBRs peuvent être modifiées, telles que la version du compilateur GCC utilisée. Pour couronnée de succès la configuration du menu, la nouvelle configuration doit être sauvegardée dans le répertoire `src/fbr/buildroot/.config`.

S'il vous plaît noter, que des modifications dans la configuration du toolchain ne sont officiellement *pas* prisent en charge, car les fichiers binaires auront une forte probabilité d'incompatibilités avec la distribution officielle de `fi4l`. Donc, si vous avez besoin de fichiers binaires pour votre propre OPT et que vous souhaitez publier cette OPT, vous ne devez pas modifier un paramètre de chaîne de compilation !

Reconfiguration de la bibliothèque `uClibc`

Si vous utilisez de commande `fbr-make uclibc-menuconfig` vous pouvez modifier la fonctionnalité de la bibliothèque `uClibc`. Pour couronnée de succès la configuration du menu, la nouvelle configuration doit être enregistrée dans le répertoire `fbr-make uclibc-menuconfig`.

Il faut prendre également en compte comme dans le dernier paragraphe : une modifications sera hautement probable qu'elle ne soit pas compatible avec la distribution officielle de `fi4l` et ne sera donc pas pris en charge !

Reconfiguration de `Busybox`

A l'aide de la commande `fbr-make busybox-menuconfig`, vous pouvez régler le fonctionnement de `Busybox`. Pour couronnée de succès la configuration du menu, la nouvelle configuration doit être enregistrée dans le répertoire `src/fbr/buildroot/package/busybox/busybox-<Version>.config`.

Ici aussi, un changement n'est probablement pas compatible avec la distribution officielle de `fi4l` et n'est donc pas pris en charge ! Tout au plus le fait de compléter

le Busybox autour de nouvelles applets est sans problèmes, tant que vous utilisez le Busybox ainsi modifié seulement sur votre routeur fli4l, (il ne permet pas l'utilisation d'un tel Busybox modifié, pour vos propres OPTs).

Reconfiguration du paquet Kernel-Linux

A l'aide de la commande `fbr-make linux-menuconfig` ou `fbr-make linux-<version>-menuconfig` la configuration de tous les paquets Kernel ou d'un paquet Kernel spécifique est permis. Pour couronnée de succès la configuration du menu, la nouvelle configuration doit être enregistrée dans le répertoire `src/fbr/buildroot/linux/linux-<version>/dot-config-<arch>`. Ceci est valable seulement pour un Kernel standard. Pour une variante du paquet Kernel, au lieu que le fichier soit indiqué `diff`, il sera indiqué `src/fbr/buildroot/linux/linux-<version>/linux-<version>_<`

Il faut prendre également en compte comme dans le dernier paragraphe : un changement n'est probablement pas compatible avec la distribution officielle de fli4l et n'est donc pas pris en charge ! Tout au plus le fait de compléter le Kernel Linux autour de nouveaux modules c'est sans problèmes (il ne permet pas l'utilisation d'un tel Kernel modifié, pour vos propres OPTs).

1.1.7. Mise à jour des FBRs

Pour chacune des commandes décrites ci-dessus, un examen des FBRs est effectué pour une évolution de la mise à jour. En cas de divergence entre les sources, après avoir utilisé le `fbr-make` pour (décompacter le paquet `src` ou pour la copie de travail SVN) et le FBR dans `~/fbr/fbr-<branch>-<arch>/buildroot`, les différentes sources trouvées seront mise à jour. Les nouveaux paquets FBR ainsi que les anciens seront intégrés, les paquets FBR existants seront effacés. Les configurations sont comparées : les paquets FBR dont la configuration a été récemment modifiée ainsi que tous les paquets FBR qui en découle seront construits. Cela garantit que le FBR sur votre ordinateur est toujours conforme pour le développement de fli4l (excepté vos propres paquets FBR qui sont dans `~/fbr/own/`). **Cela signifie aussi que des modifications de la partie officielle et de la configuration du Buildroot seront perdues lors de la prochaine de la mise à jour !** C'est pourquoi, une reconfiguration des FBRs (voir ci-dessus) n'est pas recommandé, du moins pas si à la place des paquets `src` vous travaillez avec une copie de travail `src`. (Lors de la mise à jour d'une copie de travail SVN, vos changements de configuration locaux et les modifications apportées dans le SVN-Repository sont fusionnées, si bien que le problème de la perte de la configuration ne se produira pas ici). En revanche, vous pouvez reconfigurer vos propres paquets FBR facilement, sans provoquer une mise à jour avec la perte de données.

1.1.8. Intégrer vos propres programmes dans le FBR

La compilation des paquets FBR individuels est contrôlé par un petit fichier `make`. Si vous voulez développer vos propres paquets FBR, et si vous utilisez le fichier `make` une description de la configuration est dans `~/fbr/own/<paquet>/`. Pour comprendre comment ceux-ci sont construits et comment écrire son propre fichier `Make`, une documentation décrit en détail le uClibc-Buildroots dans <http://buildroot.uclibc.org/downloads/manual/manual.html#adding-packages>.

A. Annexe du paquetage SRC

A.1. SRC - Développement de son propre paquetage

Si vous voulez développer votre propre paquetage, on pourrait appeler (paquetage) un programme, qui n'est pas intégré dans le FBR, qui sera compilé pour fli4l, vous devez écrire au moins deux fichiers : un fichier make et un fichier pour la description de la configuration.

(TODO)

Comment ces fichiers sont construits et comment peut on écrire un dérivé de son propre Makefiles, vous trouverez une description dans la documentation de uClibc-Buildroots dans <http://buildroot.uclibc.org/downloads/manual/manual.html#adding-packages>

Table des figures

Liste des tableaux