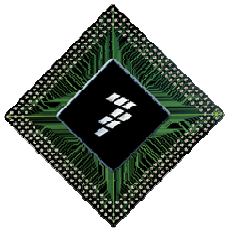
Linux Target Image Builder

Based on ltib 6.4.1 ($Revision: 1.8 $ sv)



Peter Van Ackeren – *peter.vanackeren@freescale.com*
Sr. Software FAE – Freescale Semiconductor Inc.

freescale™ semiconductor

LTIB
Topics

- ... Philosophy
- ... on the Intranet/Internet
- ... Package pools
- ... Policies
- ... Host Support
- ... Installation
- ... Directory Structure
- ... Commands
- ... Command Line Options
- ... BSP Configuration / Build
- ... Configurations and Build Commands
- ... Working with Individual Packages
- ... Patch Generation
- ... Publishing a BSP
- ... Tips and Tricks
- ... References

indicates advanced material or links for power users

LTIB
Philosophy

- Freescale GNU/Linux Target Image Builder is a tool created by Freescale, that is used to build Linux target images, composed of a set of packages
- LTIB has been released under the terms of the GNU General Public License (GPL)
- LTIB BSPs draw packages from a common pool.
  All that needs to be provided for an LTIB BSP is:

  1. cross compiler
  2. boot loader sources
  3. kernel sources
  4. kernel configuration
  5. top level config file ... **`main.lkc`**
  6. BSP config file ... **`defconfig`**

LTIB
Philosophy (cont.)

- A lightweight command line interface controls scripts and configuration menus to perform the following functions :
  - Build kernel, boot loader and application packages from source
  - Deploy built packages to a root file system (RFS) tree
  - Prepare appropriate kernel or RFS image files ready for network or flash based use on the embedded target board
  - Manage target image files using a private rpm database per LTIB instance on the host
  - Capture source modifications into patches and auto update .spec files
  - Interface directly to the network / Internet for package download and update from public CVS site
  - All package building is done as regular user (i.e. non-root)

LTIB
Philosophy (cont.)

- LTIB performs all package configuration, build and installation tasks, that would normally take place on a self hosted Linux target platform
- Conceptually running LTIB means updating the RFS tree according to the desired configuration, including the boot loader and kernel, relying on a private per-project host based RPM management for the specific target platform
- LTIB manages changes to a package by transparently working with released or user generated .patch files

LTIB on the Internet
http://savannah.nongnu.org/projects/ltib

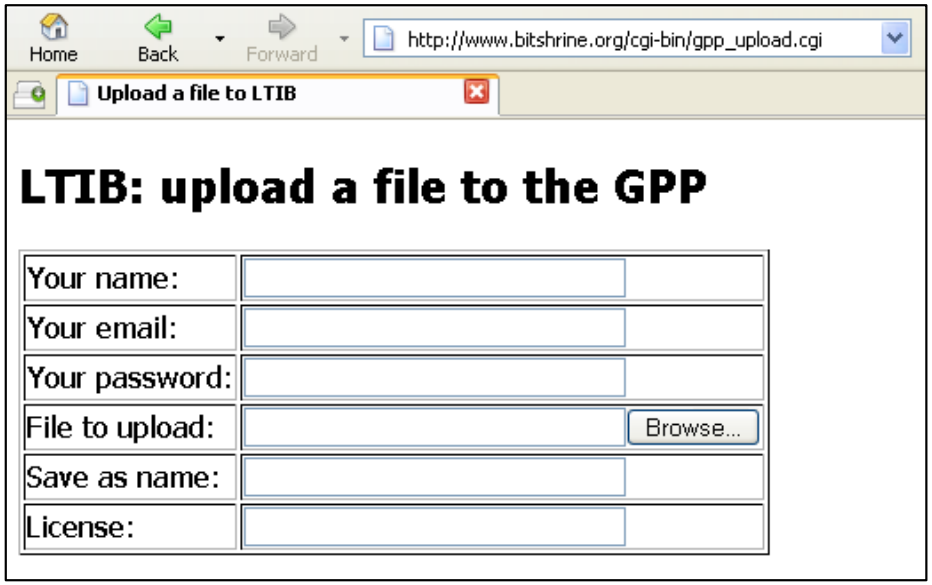
- Target audience are LTIB developers

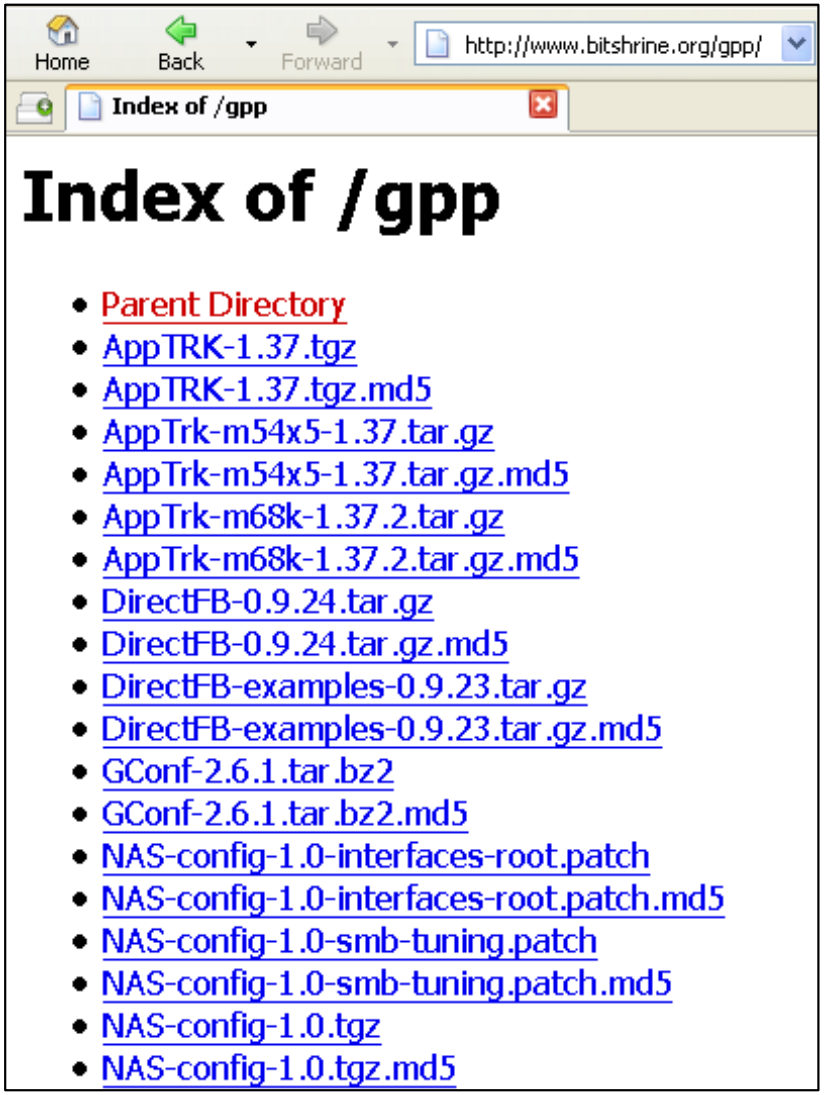
- Regular LTIB users should go to http://www.bitshrine.org

freescale™ semiconductor

The GPP on the Internet

http://www.bitshrine.org



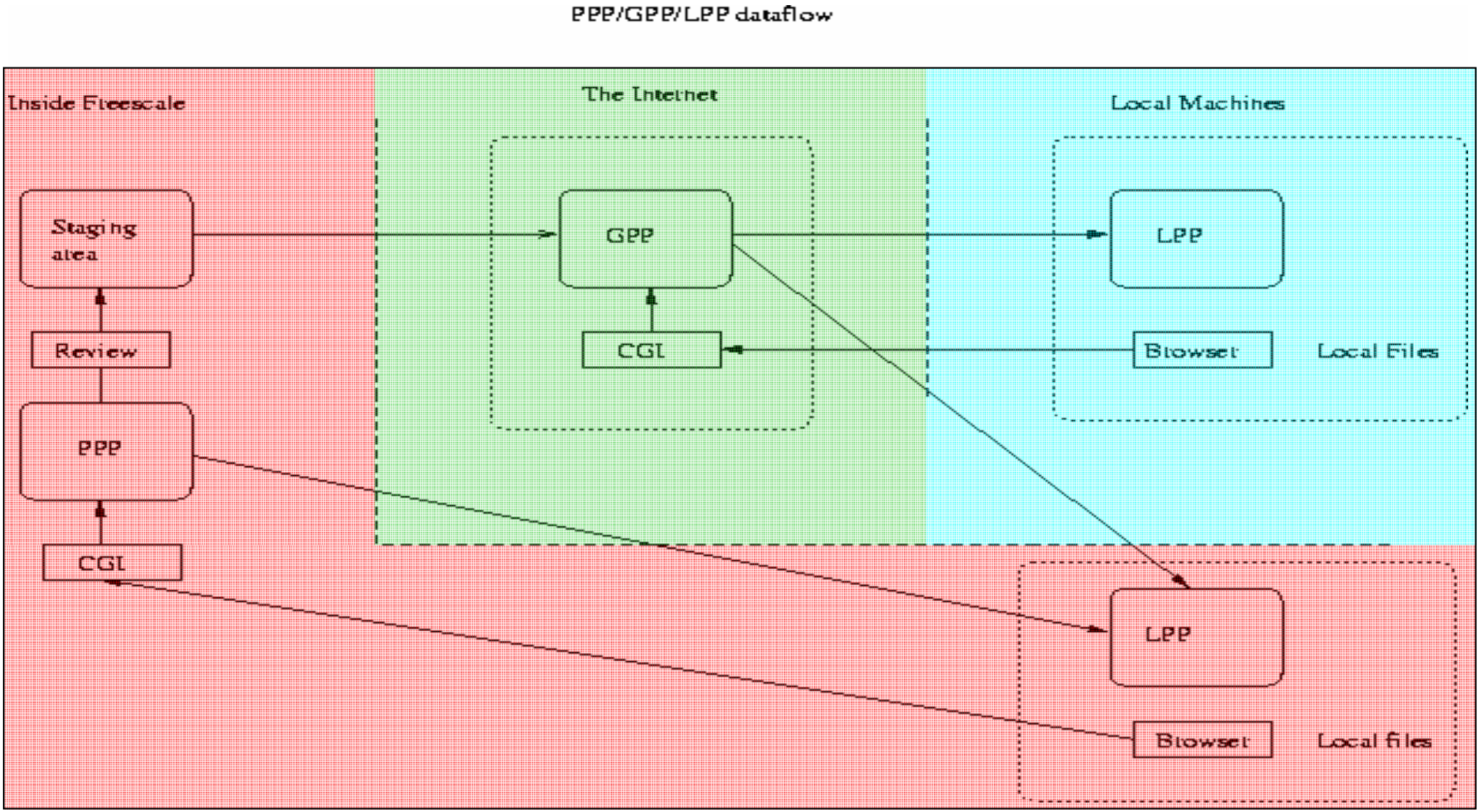


freescale™ semiconductor

LTIB Package Pools
Overview

- Each package normally consists of a main archive + patches (`.tar.gz/.tgz/.bz2 +.patch`) located in one of the 3 types of package pools
  - **PPP (Private Package Pool) :** inside the Freescale network...
    - ...with private contents
  - **GPP (Global Package Pool) :** external to Freescale, but mostly a sub-set to PPP...
    - ...with public contents
    - Files will have a suitable license for copying, or have no legal copy restrictions (public domain)
  - **LPP (Local Package Pool) :** a local directory where ...
    - LTIB will cache packages/patches that it downloads from the GPP
    - local users can put their own packages / patches during development
    - local users can share the same open source packages

LTIB Package Pools
Dataflow



freescale™ semiconductor

LTIB Policies
Moving files from the PPP to the GPP

- For a file to be publicly accessible, it has to be published on the GPP.
- All files from Freescale must initially be uploaded to the PPP and reviewed before they can be copied to the Freescale GPP.
- Once published on Freescale's GPP, these files will be mirrored to the GPP on the internet : http://www.bitshrine.org
- LTIB accesses the GPP directly over the Internet to retrieve all needed components for a BSP
- Trusted external developers (with an account/password) can directly upload to the external GPP

LTIB Host Support

- LTIB will run on the following supported Linux-only hosts :

  - **x86 Linux**
    - Redhat: 7.3, 8.0, 9.0
    - Fedora Core: FC1/2/3/4/5/6
    - Debian: 3.1r0 (stable), unstable (*)
    - Suse: 8.2, 9.1, 9.2, 9.3, 10.0, 10.1
    - RedHat Enterprise: TBD

  - **PPC Linux**
    - Debian: 3.1r0 (stable), unstable (*)

(*) *stable: “sarge”, latest officially released distribution of Debian*
unstable: “sid”, where active development of Debian occurs

LTIB Installation
Latest Public Version from www.bitshrine.org

- Follow the Quick Install instructions on www.bitshrine.org :
  - download the netinstall Perl script
  - run the script on your Linux development workstation

```
$ perl netinstall.pl

You are about to install the LTIB (GNU/Linux Target Image Builder)
Do you want to continue ? Y|n
Y
Where do you want to install LTIB ? (/mnt/tmp/more_ltib_bsps/bitshrine_savannah/ltib)

Installing LTIB to /mnt/tmp/more_ltib_bsps/bitshrine_savannah/ltib
+ cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/ltib co -d ltib -P ltib
cvs checkout: warning: failed to open /home/fsl/.cvspass for reading: No such file or directory
cvs checkout: Updating ltib
[...]cvs checkout: Updating ltib/bin
[...]
cvs checkout: Updating ltib/config
cvs checkout: Updating ltib/config/defaults
[...]
cvs checkout: Updating ltib/internal
   [...]
LTIB download complete, your ltib installation has been placed in
/mnt/tmp/more_ltib_bsps/bitshrine_savannah/ltib, to complete the installation, run the following
   commands:

$ cd /mnt/tmp/more_ltib_bsps/bitshrine_savannah/ltib
$ ./ltib
```

LTIB Installation
From a Binary Release (.ISO Image or CD)

- Mount the image and run the BSP's install script :

```
$ sh <CD mount point>/install
```

  Enter your chosen installation directory...
  An **`ltib`** or **`<bsp-name>`** sub-directory will be created in that location.

- ... or if you got LTIB as a tar file from a internal Freescale or public download location, unpack it in a new directory

- Run the LTIB install script as a regular user (not as root) :

```
$ cd <install path>/ltib
$ ./ltib
```

LTIB Installation
General

- The first time LTIB will build and install the host side packages (mostly **`rpm-fs`**), which will take quite a long time.
- Some host packages are unlikely to be on the host, so they are built from sources provided with LTIB (e.g. lkc, genext2fs, mtd-utils)
- These host packages are shared across LTIB installs
- If installation fails with an error, check the log... a missing package in your host environment is a common occurrence

```
$ tail -n 50 host_config.log
```

  Report problems with ISO releases ..... to Freescale support
  Reporting problems with Bitshrine LTIB ... to the public LTIB mailing list

LTIB Installation
Host Dependencies

package	version	comment
perl	>= 5.6.1	ltib script
sudo	any	to run the 'rpm install' phase on each package
wget	any	to download packages/patches on demand
rpm-build	any	need by rpm to do actual building of packages
rpm	any	to build initial rpm-fs host package
glibc	>= 2.2.x	to build/run host packages
libstdc++-devel	any?	to build rpm-fs host package
binutils	>= 2.11.93	to build host packages
gcc	>= 2.96	to build host packages
gcc-c++	>= 2.26	to build rpm-fs host package
zlib-devel	any	to build rpm-fs and mtd-utils host packages
ncurses	>= 5.1	to build lkc (config language) host package
ncurses-devel	>= 5.1	to build lkc (config language) host package
m4	any?	may be needed by bison
bison	any	to build lkc (config language) host package
flex	any	to build lkc (config language) host package
texinfo	any	to build genext2fs host package
libtool	>= 1.4.2	to build libusb target package
gettext	any	to build genext2fs target package

freescale™ semiconductor

LTIB Installation Gotchas
sudo permissions

- PROBLEM

```
$ ./ltib
I ran the command: sudo -S -l which returned:

<SNIP>

To configure this, as root using the command "/usr/sbin/visudo",
and add the following line in the User privilege section:

<username> ALL = NOPASSWD: /bin/rpm, /opt/freescale/ltib/usr/bin/rpm

<SNIP>
```

- SOLUTION
  - Do as indicated

LTIB Installation Gotchas
rpmpopt Error

- PROBLEM (LTIB in older BSPs only)

```
$ ./ltib
ERROR: link target doesn't exist (neither in build root nor in installed
system):
/var/tmp/freescale/usr/lib/rpmpopt -
  /var/tmp/freescale/usr/lib/rpm/rpmpopt
```

- SOLUTION
  - This is a problem related to SuSE's host side rpm implementation
  - Edit the file: **`./dist/lfs-5.1/rpm/rpm-fs.spec`** (line 70)

```
%Install
export NO_BRP_STALE_LINK_ERROR=yes
rm -rf $RPM_BUILD_ROOT
```

  - Once you've done that, you'll need to remove the failed rpm build and then re-run ltib e.g:

```
$ rm -rf /tmp/rpm-"login"
$ ./ltib
```

LTIB Installation Gotchas
loading shared libraries Error

- PROBLEM (some older BSPs on some distro's)

```
sed: error while loading shared libraries: libc.so.6: cannot open shared
object file: No such file or directory
```

- SOLUTION

Edit **dist/lfs-5.1/rpm/rpm-fs.spec** and comment out the following lines (put a **#** in front of each line):

```
# if [ "`uname -m`" != "x86_64" ]
# then
# export LD_ASSUME_KERNEL=2.2.5
# fi
```

LTIB Un-installation

- First do ...

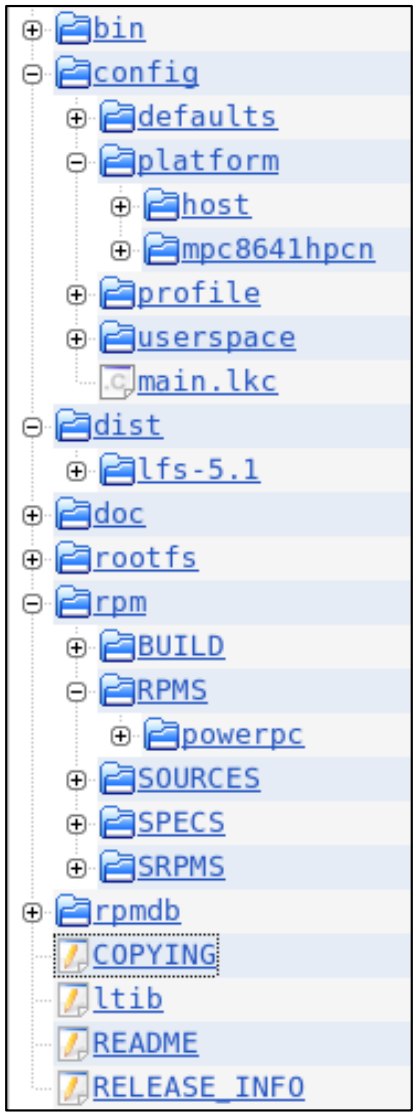
```
$ ./ltib -m distclean
```

- Then as root (and when no other users on this machine will be needing it the common files) ...

```
rm -rf /opt/freescale/pkgs
rm -rf /opt/freescale/ltib
rm <install_path>/ltib
```

LTIB Directory Structure
Per LTIB Instance

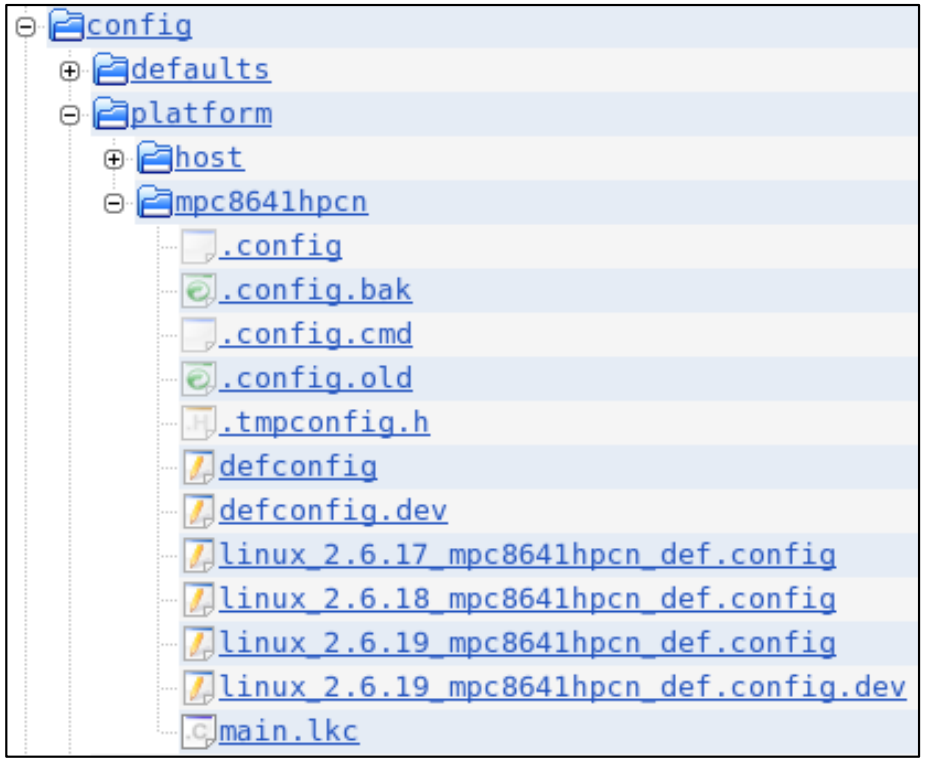
- **`./rpmdb`**
  - RPM database for this LTIB instance
- **`./rpm`**
  - Where sources, source RPMs and binary RPMs live
  - Packages are built in ./**`BUILD/<pkg>`**
  - Binary RPMs will be created in **`./RPMS/<arch>`** for all up-to-date packages
- **`./rootfs`**
  - root file system tree for the target, created by installation of binary RPMs
- **`./dist/lfs-5.1`**
  - package spec files



freescale™ semiconductor

LTIB Directory Structure
Per LTIB Instance (cont.)

- **`./config/platform/<platform_name>`**
  - **`main.lkc`** defines the BSP top level menu
  - default configuration files :
    - **`devconfig`** ... for the top level menu
    - **`.config`** ... for the Linux kernel
    - **`busybox.config`** ... for busybox
    - current (i.e. modified from default) configuration values are stored in **`<filename>.dev`**



- **`./config/platform/host`**
  - Controls the host side package that gets installed during pre-configuration to support the LTIB tool
  - **`main.lkc`** describes BSP's top level configuration

freescale™ semiconductor

LTIB Directory Structure
Common Directories

- **`/opt/freescale/ltib`**
  - Contains LTIB's common host side files, in part built during installation process
  - LTIB rpm binaries are in **`./usr/bin`**
  - LTIB rpm support files are in **`./var/lib/rpm`**

- **`/opt/freescale/pkgs`**
  - The Local Package Pool (LPP):
    global storage area for downloaded source packages and patches

- **`/var/tmp/pkgs`** (also for old LTIB version LPP compatibility)
  - Local directories, searched before the Local Package Pool for files

- **`./.ltibrc`** specifies these locations :

```
%lpp
/opt/freescale/pkgs
```

LTIB
`.config` Files

- **`./.config`**

  Selects the chosen platform (fixed for .ISO releases)

- **`./config/platform/<platform_name>/defconfig`**

  LTIB top level default configuration, for the current target platform
  Can be used with **`--preconfig`**

- **`./config/platform/<platform>/.config`**

  LTIB top level active configuration, for the current target platform

- **`./config/platform/<platform>/ \`**
  `linux-2.6.<version>-<platform>-<identifier>.config`

  Pre-defined kernel configurations for **`make menuconfig`**

LTIB
`.config` Files (cont.)

- **`./config/profiles/<profile_name>.config`**

  Defined set of user space packages, excluding kernel, bootloader and deployment choices, i.e. largely target independent
  Can be used with **`--profile`**

- **`./config/defaults/busybox.config`**

  Default configuration file for busybox

Issuing LTIB Commands

- There is an instance of LTIB per installed BSP
- To issue an LTIB command for a BSP, always put yourself in the directory where the BSP is installed

```
$ cd <install_dir>
$ ./ltib <some command>
```

- LTIB evolves continually, and the most recent stable version is normally included when a new BSP is released.

  It always makes sense to check, which commands are available for your BSP using **`./ltib --help`**

LTIB Command Line Options
./ltib --help

```
ltib [-m <mode>] [options....]
    Where:
        --mode|m
          Where mode is either:
            prep          just prep the package
            scbuild       rpmbuild -bc --short-circuit
            scinstall     rpmbuild -bi --short-circuit
            scdeploy      does an scinstall followed by an install to the rootfs
            patchmerge    generate and merge a patch (requires -p <pkg>)
            clean         clean/uninstall target packages
            distclean     full cleanup, removes nearly everything
            listpkgs      list packages (alphanumeric)
            listpkgseula  list package names and licenses
            listpkgstw    list packages in twiki format
            release       make a binary release iso image
            config        use with --configure to do configuration only
            shell         enter ltib shell mode (sets up spoofing etc)
        --pkg|p         :  operate on this package only
        --configure|c   :  run the interactive configuration
        --preconfig     :  configuration file to build from (defaults to .config)
```

[These commands may be useful later on]

More LTIB Command Line Options

./ltib --help

```
--profile      :  profile file.  This is used to select an alternate
                  set of userspace packages, this is saved and used
                  on later runs of ltib (e.g config/profiles/max.config)
--rcfile|r <f>:  use this resource file
--batch|b      :  batch mode, assume yes to all questions
--force|f      :  force rebuilds even if they are up to date
--reinstall|e  :  re-install rpms (but don't force rebuild)
--nodeps|n     :  turn off install/uninstall dependency checks
--conflicts|k  :  don't force install rpms that have file conflicts
--keepsrpms|s  :  keep the srpms after the build (deleted by default)
--verbose|v    :  more output
--dry-run|d    :  mostly a dry run (calls to system are just echos)
--continue|C   :  try to continue on package build errors (autobuilds)
--version|V    :  print the application version and quit
--noredir|N    :  do not redirect any output
--deploy|D     :  run the deploy scripts even if build is up to date
--dlonly       :  just download the packages only
--dltest       :  test that the BSP's packages are available
--leavesrc|l   :  leave the sources unpacked (only valid for pkg mode)
--hostcf       :  (re)configure/build/install the host support package set
--help|h       :  help on usage
```

[You should master these commands first]

[These commands may be useful later on]

freescale™ semiconductor

LTIB BSP Configuration
General

- To configure and build Linux OS for the target :

```
$ ./ltib --configure
```

  This allows to change the configuration of the BSP options in the top level menu.

  Upon exit LTIB will execute everything needed to regenerate the image files in accordance with the desired configuration

- LTIB will show the Platform Selection menu if needed, and then continues on to the Top Level menu

  Note: ISO releases will typically only offer a single selectable platform

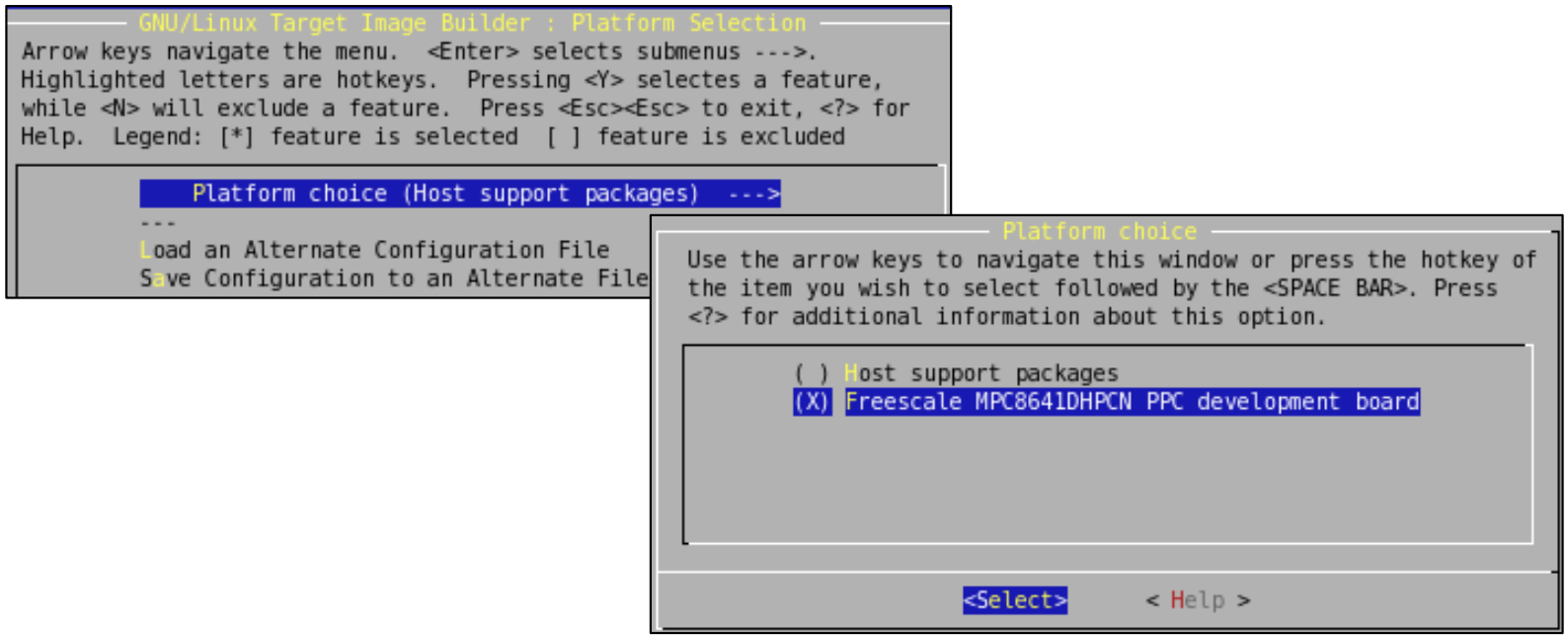
- Most users initially will not have to make any changes to the BSP configuration and can directly choose Exit to let LTIB run to completion

LTIB Platform Selection Menu
Bitshrine / CVS Installs

- Shows a list of available target platforms as defined in

  `./config/platform` ...

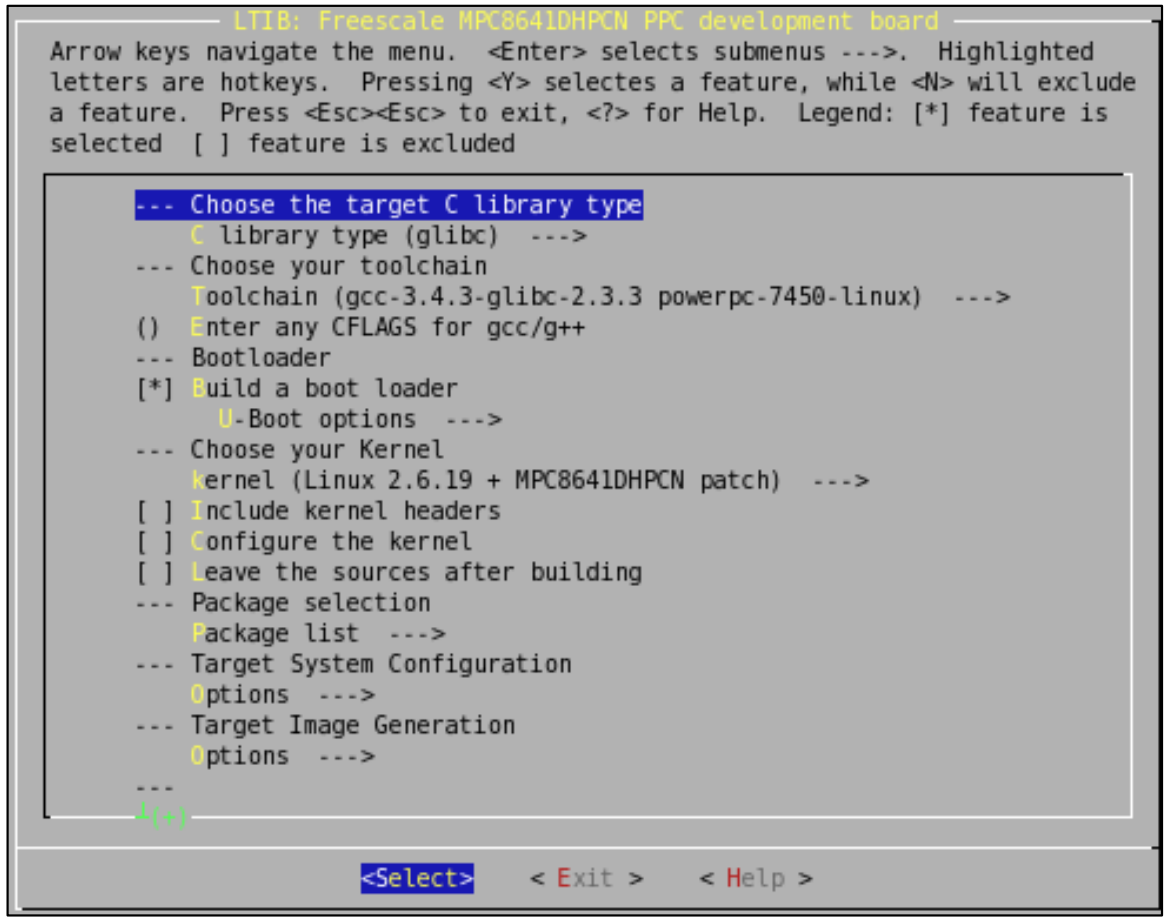
  - ... when working with BSPs from the Web based GPP
  - ... also when running **`./ltib`** for the first time after cleaning out a previous build with **`-m distclean`**



freescale™ semiconductor

LTIB Top Level Configuration Menu
Keyboard Navigation

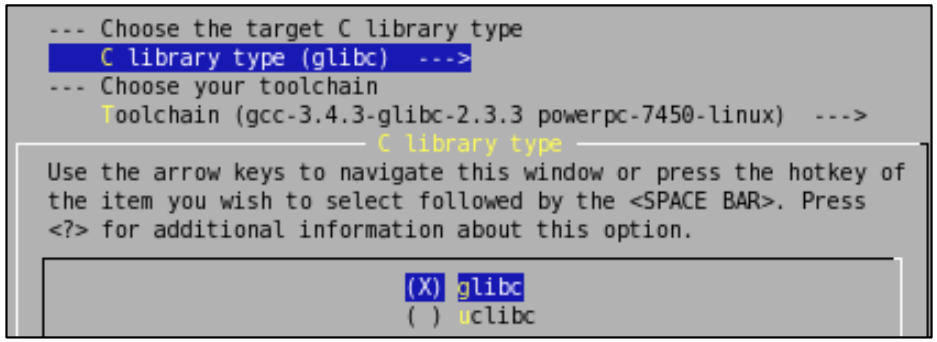
Up, Down, PgUp, PgDn, Home, End
..... move to a menu item
..... (shortcut = enter highlighted capital letter)
..... ***Left*** ..... choose from
..... ***Right*** ... *Select/Exit / Help*
... ***ENTER*** .. select item under cursor / enter sub-menu ("- - - >")
... ***SPACE*** .. enable an *[ ]* item
..... ***Esc*** ..... return from the current menu level (also *Exit*)
..... ***S*** ..... search for a text string in the menu system
..... ***/*** ..... search for configuration keyword (e.g. PKG_BUSYBOX)

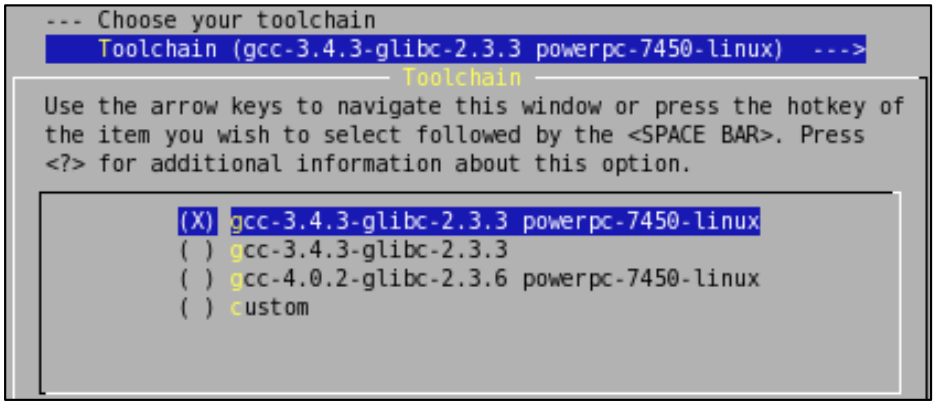


freescale™ semiconductor

LTIB BSP Configuration
Selecting the `libc` C-library and `gcc` Tool Chain

- Currently selected **`libc`** and GCC toolchain are shown, with alternative choices available as per the BSP





- Toolchain choice drives the CFLAGS entry, which can be manually overridden

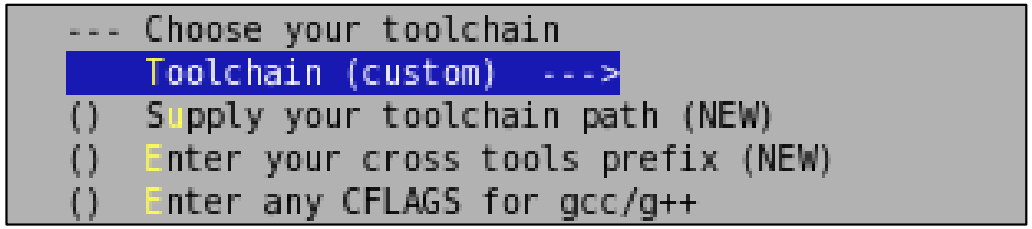
Note: the libc for the target platform is part of the GCC binary RPM that comes with the BSP, so the cross compiler and libc are always in sync

freescale™ semiconductor

LTIB BSP Configuration
Selecting the `gcc` Tool Chain (cont.)

- A single custom tool chain can be specified by selecting the **`(custom)`** item ...

  The user must provide the fully qualified path to the custom tool chain and the cross tools prefix (e.g. **`powerpc-linux-`**):



- To specify more than 1 additional tool chain, extend the file **`config/platform/<platform_name>/main.lkc`** with additional entries (example for advance users follows)

freescale™ semiconductor

Adding a Tool Chain to an LTIB BSP
in config/platform/<platform>/main.lkc

```
comment "Choose your toolchain"
choice
    prompt "Toolchain"
    default TOOLCHAIN1
    help
        This menu will help you choose the cross toolchain to use to build
        your packages with.  If you choose none, you can enter the
        path to your toolchain by hand.
        Note: gcc-3.4 cannot be used to compile linux-2.4.x

    config TOOLCHAIN1
        bool "gcc-3.4.3-glibc-2.3.3 powerpc-7450-linux" if GLIBC
    config TOOLCHAIN2
        bool "gcc-3.4.3-glibc-2.3.3" if GLIBC
    config TOOLCHAIN3
        bool "gcc-3.4.3/uclibc-0.9.28" if UCLIBC
    config TOOLCHAIN4
        bool "gcc-4.0.2-glibc-2.3.6 powerpc-7450-linux" if GLIBC
        help
            gcc-4.0.2 glibc-2.3.6 NPTL thread library toolchain.

    config TOOLCHAIN_CUSTOM
        bool "custom"
endchoice
```

Adding a Tool Chain to an LTIB BSP
in config/platform/<platform>/main.lkc

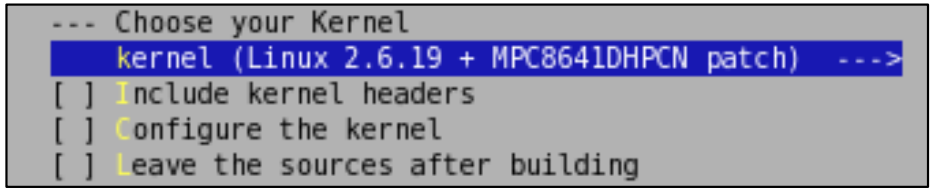
```
config TOOLCHAIN
    string
    default tc-mtwk-lnx-7450-3.4.3-1.i686.rpm if TOOLCHAIN1
    default mtwk-lnx-powerpc-gcc-3.4.3-glibc-2.3.3-0.28-1.i686.rpm if TOOLCHAIN2
    default tc-fsl-x86lnx-ppc-uclibc-3.4.3-1.i386.rpm if TOOLCHAIN3
    default tc-fsl-x86lnx-7450-nptl-4.0.2-2.i386.rpm if TOOLCHAIN4
config TOOLCHAIN_PATH
    string
    default "/opt/mtwk/usr/local/gcc-3.4.3-glibc-2.3.3/powerpc-7450-linux" if TOOLCHAIN1
    default "/opt/mtwk/usr/local/powerpc-linux/gcc-3.4.3-glibc-2.3.3" if TOOLCHAIN2
    default "/opt/freescale/usr/local/gcc-3.4.3-uClibc-0.9.28-nfp-1/powerpc-linux" if TOOLCHAIN3
    default "/opt/freescale/usr/local/gcc-4.0.2-glibc-2.3.6-nptl-2/powerpc-7450-linux" if TOOLCHAIN4
    string "Supply your toolchain path" if TOOLCHAIN_CUSTOM

config CUSTOM_TOOLCHAIN_PREFIX
    depends TOOLCHAIN_CUSTOM
    string  "Enter your cross tools prefix"
    help
        For example arm-linux- or powerpc-linux-

config TOOLCHAIN_PREFIX
    string
    default "powerpc-7450-linux-" if TOOLCHAIN1 ||  TOOLCHAIN4
    default "powerpc-linux-" if TOOLCHAIN2
    default "powerpc-linux-uclibc-" if TOOLCHAIN3
    default CUSTOM_TOOLCHAIN_PREFIX if TOOLCHAIN_CUSTOM
```

LTIB BSP Configuration
Selecting the Linux Kernel

- Currently selected kernel is shown, with alternative choices available as per the BSP



- Let LTIB configure the kernel by running make menuconfig with :

  Configure the kernel

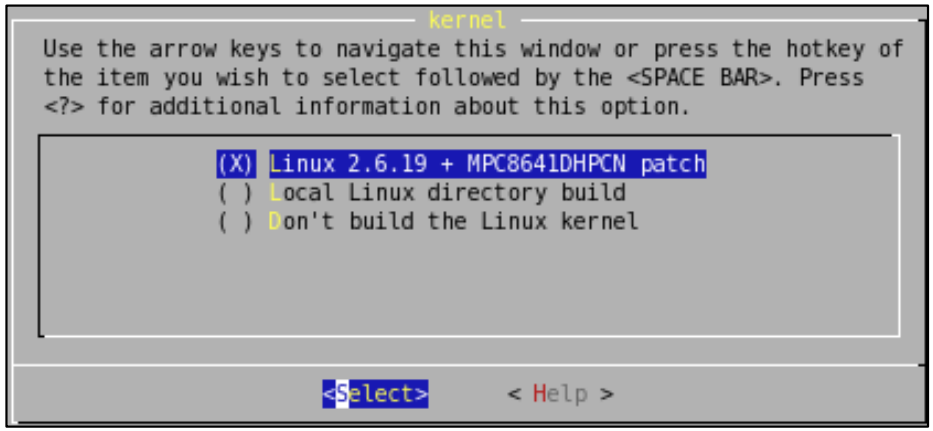
- To suppress rebuilding the kernel :

  Don't build the Linux kernel

- To build another kernel tree :

  Local Linux directory build

  A fully qualified path to the kernel tree to use must be provided



freescale™ semiconductor

LTIB BSP Configuration
Selecting the Linux Kernel (cont.)

- Linux code that is to be built separately from the BSP, like custom drivers, will need to reference the kernel header files.

  To install kernel headers in **`./rootfs/usr/include`** (RFS tree) :

  Include kernel headers

- To allow the Linux kernel source debugging, the kernel source tree will have to be retained ... select :

  Leave the sources after building

LTIB BSP Configuration
Common Package Selection

- **Package Selection → Package List** opens the Common Package Selection screen
- All packages selected will be built if needed and installed into the RFS
- A package required another package, will be auto-selected (auto-dependency resolution)
- To run the **busybox** configuration menu prior to building/installating the package :

  Configure busybox at build time

  Note:

  busybox.config preconfig file is located in **config/platform/<platform_name>**

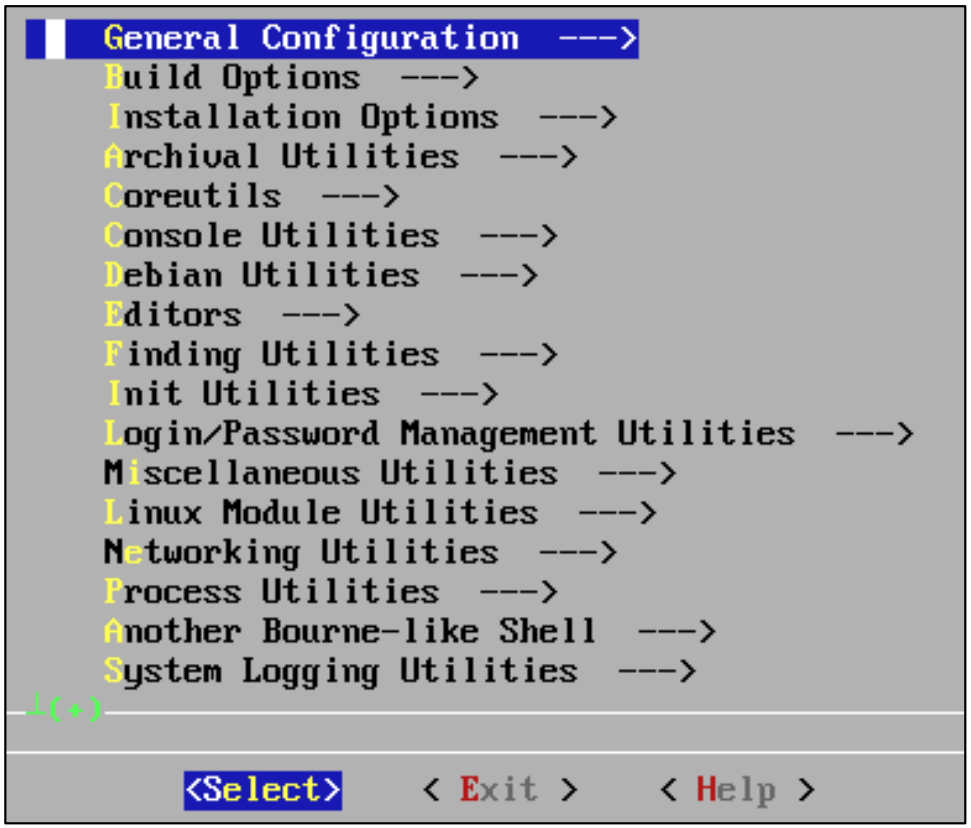
```
[ ] apptrk binary package for powerpc
[ ] autoconf
[ ] automake
[*] Include C library
(base_libs) C library package
[ ]   Include libc locale files ?
[ ]   Include header files from toolchain ?
[ ]   Include static libc libraries ?
[ ] alsa-lib
[ ] alsa-utils
[ ] bash
[ ] bind
[ ] binutils
[ ] bison
[ ] boa
[ ] bonnie++
[ ] bridge-utils
[*] busybox
(busybox.config) busybox preconfig filename
[ ]   Configure busybox at build time
[ ] bzip2
[ ] can4linux
[ ] clamav
[ ] coreutils
[ ] cpio
[ ] cracklib
-(+)
```

LTIB BSP Configuration
Common Package Selection (cont.)

- Dependency checking :
  - A configured independent package will take precedence over its **`busybox`** equivalent in the RFS, without any upstream conflict checking, nor removal from the busybox configuration.
  - If an independent package is removed, there are triggers to re-install busybox. This trigger mechanism is general and works for other packages too.
- Typical package selections :
  - Default : (most BSPs) **`baselibs, busybox, skeleton base files, ntpclient +`** always auto-selected : **`fake-provides, merge`**
  - Useful : **`usbutils, pciutils, ethtool`**
  - CodeWarrior : **`apptrk binary package`**
  - GDB : **`gdb, gdbserver`**

LTIB Package Configuration
Configuring *busybox*

- If **`busybox`** configuration is specified in the top level menu, LTIB will install the **`busybox`** source tree and run its **`make menuconfig`** prior to building the package



freescale™ semiconductor

LTIB Common Target System
Configuration Options

- Use **Target System Configuration → Options** to define how the target system should start up and configure itself at run time (services, ...)

```
(mpc8641hpcn) target hostname
[*] boot up with a tty and login
(::respawn:/sbin/getty -L ttyS0 115200 VT100) Enter your inittab startup line
()  load these modules at boot
[ ] start devfsd
[*] start networking
      Network setup  --->
[*] set the system time at startup
[*] start syslogd/klogd
[*] start inetd
()    Enter command line arguments for inetd startup
```

- Config files and scripts for the RFS are created/modified accordingly

  ... including Network setup with configuration per enabled interface

```
[*] Enable interface 0 (NEW)
(eth0) interface (NEW)
[ ]   get network parameters using dhcp (NEW)
(192.168.1.152) ip address
(255.255.255.0) netmask (NEW)
(192.168.1.255) broadcast address
(192.168.1.20) gateway address
(192.168.1.20) nameserver ip address
---
[ ] Enable interface 1 (NEW)
---
[ ] Enable interface 2 (NEW)
---
[ ] Enable interface 3 (NEW)
---
[ ] Enable interface 4 (NEW)
```

LTIB Common Target Image
Generation Options

- Use **Target Image Configuration → Options** to ...
  - ... select the type of the target image RFS : **`jffs2, ext2`** or **`NFS`**
    (do not select NFS for a kernel debug setup nor for production deployment)
  - ... reduce the footprint of the target image RFS type

```
--- Choose your root filesystem image type
    Target image: (ext2.gz ramdisk)  --->
[ ] read-only root filesystem
[*] create a ramdisk that can be used by u-boot
()  rootfs target directory
[ ] Keep temporary staging directory
[*] remove man pages etc from the target image
[*] remove the /boot directory
[*] remove the /usr/src/ directory
[*] remove the /usr/include directory
()  remove these directories
()  remove these files
[*] remove the static libraries
[*] strip any remaining binaries or libraries in the target image
(0) Allocate extra space (Kbytes)
```

```
( ) jffs2
(X) ext2.gz ramdisk
( ) cramfs(TODO)
( ) NFS only
```

Assures the rootfs image will have a suitable Magic word for U-boot

- You can provide a space separated list of target directories and files to be removed from the RFS

LTIB Common Target Image
Removing files from the RFS

- Most embedded boot loaders don't use the **`/boot`** directory.
  Also header files in places like **`/usr/src/linux, /usr/include`** serve no run-time purpose in the RFS of a deployed embedded target system.

  Use the **`[*] remove...`** options to exclude these files from the RFS

- Likewise it is unlikely symbol information will be required on the target system itself, since debugging normally takes place in a cross development environment.

  Use the **`[*] strip...`** options remove symbol information from libraries and executable files

LTIB BSP Configuration
... Ready to Build

- Exit from the top level menu when configuration is done

- LTIB will then...

1. ... gather BSP **configuration** changes from the user
2. ... create and/or update all needed **binary RPMs** for the target platform, as required by the BSP configuration, by building from original sources and patches
3. ... create a **root file system file** (RFS) tree, by installing the binary RPMs of the packages needed for the configuration
4. ... generate all **image files** needed for deploying Linux OS to the target board : boot loader, Linux kernel, file system(s)
5. ... leave **image deployment** to the target hardware as a **manual operation** for the user, as documented for the BSP

freescale™ semiconductor

LTIB BSP Configuration
... Ready to Build (cont.)

- The top level configuration will be stored in :

  `./config/platform/<platform_name>/[defconfig.dev|.config]`

  ... and used to update the entire RFS ...

  Each package will be configured, built and deployed as needed :

  - package sources are unpacked into **`./rpm/BUILD/<pkg>`**
  - package configuration may be run first (e.g. for **`kernel`** or **`busybox`**).
  - **`.config`** files for kernel and busybox are copied to **`.dev`** versions
  - the package build order is defined in **`./dist/lfs5.1/common/pkg_map`**
  - **`<pkg>.rpm`** is created in **`./BUILD/RPMS/<arch>`** and marked as up-to-date
  - rpm removes the package sources (THIS IS THE DEFAULT BEHAVIOUR !!!)
  - rpm installs the **`<pkg>.rpm`** into the RFS

LTIB Image Generation
Final Stages

```
Processing deployment operations
==================================
making filesystem image file
staging directory is /mnt/tmp/more_ltib_bsps/ltib-mpc8641hpcn-
20070118/rootfs.tmp
removing the boot directory and files
removing man files and directories
removing info files
removing /usr/src directory
removing /usr/include directory
removing static libraries
stripping binaries and libraries
```

```
Filesystem stats, including padding:

    Total size            = 10495k
    Total number of files = 495

Your ramdisk exceeds the old default size of 4096k, you may need to
set the command line argument for ramdisk_size in your bootloader
allowing 10% free this gives 11544k .  For instance, for u-boot:

setenv bootargs root=/dev/ram rw ramdisk_size=11544
```

← Apply prior to booting

LTIB Image Generation
Final Stages (cont.)

```
creating an ext2 compressed filesystem image: rootfs.ext2.gz
creating a uboot ramdisk image: rootfs.ext2.gz.uboot
Image Name:   uboot ext2 ramdisk rootfs
Created:      Wed Apr 18 13:00:43 2007
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
Data Size:    3097764 Bytes = 3025.16 kB = 2.95 MB
Load Address: 0x00000000
Entry Point:  0x00000000
```

```
Started: Wed Apr 18 12:55:42 2007
Ended:   Wed Apr 18 13:00:43 2007
Elapsed: 301 seconds

Build Succeeded
```

More on LTIB Configuration and Building
Changing only the Top Level Configuration

- To change the top level BSP configuration only

```
$ ./ltib -m config
```

  This only changes the configuration of the BSP's top level screen, defined by **`config/platform/<platform>/main.lkc`**

  Upon exit the configuration state is saved, then LTIB stops.

- To bring all BSP image files up to date, according to the saved configuration state

```
$ ./ltib
```

More on LTIB Configuration and Building
Configuration State Management

- When you run top level configuration, the following file manipulation takes place in **`./config/platform/<platform_name>`** :
  - restore the **`.config`** file from the saved copy
    - if **`defconfig.dev`** exists
      copy **`defconfig.dev`** → **`.config`**
    - else
      copy **`defconfig`** → **`.config`**
  - run the configuration screen, using the configuration stored in **`.config`**
  - upon exit, save the current configuration state :
    - **`copy .config → defconfig.dev`**
- The same approach is used to save and restore the **`.config`** configuration state for packages like the kernel, the source of which is by default not retained between builds

More on LTIB Configuration and Building
Regenerating All Packages and RFS from Scratch

- Purge all built packages, reconfigure and recompile all images :

```
$ ./ltib -m distclean

<install_dir>/rpm
<install_dir>/rpmdb
<install_dir>/tmp

To continue type in 'yes':yes
[...]

$ ./ltib --configure
```

- To force a rebuild of all packages without reconfigure :

```
$ ./ltib -f
```

Working with Individual Packages

- List available packages : **`$ ./ltib -m listpkgs`**

```
---------------------- ------------- ------- -------     ----------------------------------------
Package                   Spec file   Enabled License     Summary
---------------------- ------------- ------- -------     ----------------------------------------
DirectFB-0.9.24-1      DirectFB          n    LGPL        DirectFB is a graphics library for embedded syst
NAS-config-1.0-1       NAS-config        n    GPL         NAS setup scripts and instructions
alsa-lib-1.0.10-0      alsa-lib          n    distributab A libraries for ALSA (Advanced Linux Sound Archi
alsa-utils-1.0.10-0    alsa-utils        n    distributab Utilities for ALSA (Advanced Linux Sound Archite
autoconf-2.57-1        autoconf          n    GPL         A GNU tool for automatically configuring source
automake-1.7.6-1       automake          n    GPL         A GNU tool for automatically creating Makefiles
base_libs_mv-1.0-1     base_libs_mv      y    LGPL        Base Libraries (from toolchain).
bash-2.05b-1           bash              y    GPL         bash - GNU Bourne-Again SHell
bind-9.3.2-1           bind              n    Internet Sy Internet Systems Consortium BIND DNS server, res
binutils-2.15-1        binutils          y    GPL         A GNU collection of binary utilities.
bison-1.875-1          bison             n    GPL         A GNU general-purpose parser generator
boa-0.94.13-1          boa               n    GPL         Lightweight http server for embedded systems
bonnie++-1.93c-1       bonnie++          n    GPL         Benchmark suite for hard drive and file system p
busybox-1.01-1         busybox           y    GPL         A small executable that replaces many UNIX utili
bzip2-1.0.2-1          bzip2             n    GPL         The GNU libtool, shared libraries management too
can4linux-3.3.3-1      can4linux         n    GPL         Linux CAN/CANopen driver
clamav-0.88-1          clamav            n    GPL         Clam AntiVirus is a GPL anti-virus toolkit for U
coreutils-5.0-1        coreutils         n    GPL         coreutils - GNU core utilities commonly used in
daemonizer-1.0-0       daemonizer        n    GPL         Used to start apptrk as a deamon
db1-1.85-8             db1               n    BSD         The BSD database library for C (version 1).
dev-1.1-1              dev               y    GPL         Device files for a small embedded system
[...]
```

Working with Individual Packages (cont.)

- Running **`./ltib`** with a **`-p`** context, implies you only want to work with a specific package, irrespective of the RFS configuration.

- To work with a single package :

```
$ ./ltib -m <ltib command> -p <pkg>
```

- The **`<package>`** argument can be :
  - The package's **`.spec`** or **`.spec.in`** file
  - The package name ...
    - ... as listed with **`-m listpkgs`**
    - ... as found among **`.spec`** files in **`dist/lfs5.1`** (less the file extension)

Working an Individual Packages
Example : `.spec`-file lookup

- Specifying a package as listed in **`dist/lfs-5.1/common/pkg_map`** (also as seen in the **Package List)**, will make LTIB perform a lookup for the associated **`.spec`**-file and perform the action on the package, e.g. :

```
./ltib -p kernel

Processing platform: Freescale MPC8641DHPCN PPC development board
================================================================
using config/platform/mpc8641hpcn/.config

Processing: kernel-2.6.19-mpc8641
=================================

Started: Wed Apr 18 13:16:01 2007
Ended:   Wed Apr 18 13:16:02 2007
Elapsed: 1 seconds

Build Succeeded
```

Working an Individual Packages
Example : Alternate or New Packages

- If a package is not included in the configuration system, but it has a **`.spec`**-file, then LTIB will still perform the action on the package, e.g. :

```
$ ./ltib -m listpkgs | grep busybox
busybox-1.1.3-1  busybox  y  GPL  A small executable that replaces many UNIX utili

$ ls dist/lfs-5.1/busybox/
busybox-1.00.spec  busybox.spec

./ltib -p busybox-1.00.spec

Processing platform: Freescale MPC8641DHPCN PPC development board
================================================================
using config/platform/mpc8641hpcn/.config

Processing: busybox-1.00
========================
```

Use cases :

- you are developing a new package and you want to use LTIB on it before it has been fully integrated into the configuration system
- it is different package version from the default configured into the BSP

Working an Individual Packages
Example : Automatic Package Download

- If a package is not selectable through the configuration system, but it has a `.spec`-file and it is available in the LPP or the GPP, then LTIB will still perform the action on the package :

```
Processing platform: Freescale MPC8641DHPCN PPC development board
================================================================
using config/platform/mpc8641hpcn/.config

Processing: busybox-1.00
========================
Testing ppp_url connectivity: FAIL, disabling
Testing gpp_url connectivity: OKAY
Getting busybox-1.00.tar.bz2.md5 from the Global Package Pool
14:01:42 URL:http://www.bitshrine.org/gpp/busybox-1.00.tar.bz2.md5 [54/54] -> "busybox-
  1.00.tar.bz2.md5" [1]
Getting busybox-1.00-mmuless.patch from the Global Package Pool
14:01:47 URL:http://www.bitshrine.org/gpp/busybox-1.00-mmuless.patch [52737/52737] ->
  "busybox-1.00-mmuless.patch" [1]
Getting busybox-1.00-insmod.patch from the Global Package Pool
[...]
```

Working an Individual Packages (cont.)

- Taking a package from source to deployment in the RFS

```
$ ./ltib -m prep -p <spec file basename>
```

… installs source for the package in **`./rpm/BUILD/<pkg>`**

... you can now make your modifications and ...

```
$ ./ltib -m scbuild -p <spec file basename> [-c]
```

… re-build the package from the **`prep`**-ed source tree
Package reconfiguration may be run first ...

... if **`-c`** switch is passed
... if the saved top level menu configuration explicitly or implicitly requires it

Working with Individual Packages (cont.)

```
$ ./ltib -m scinstall -p <pkg>
```

… installs the binary and related files in **`./rpm/BuildRoot`** and marks the package as being up-to-date.

```
$ ./ltib -m scdeploy -p <pkg>
```

… creates the **`<pkg>.rpm`**
… creates a new RFS image file from the **`./rootfs`** dir

Patch Generation
Capturing Package Source Changes

```
$ ./ltib -p <pkg> -m patchmerge
```

… creates a patch file called :

```
/opt/freescale/pkgs/<pkg>-<timestamp>.patch
```

that includes any changes you have made to the sources relative to the source referenced in the packages spec file

… updates the **`<pkg>.spec`** file

Patch Generation
Capturing Package Source Changes (cont.)

- The new patch is generated under a unique filename :

```
A patch has been generated and placed in:

  /opt/freescale/pkgs/<pkg>-<10-digit suffix>.patch

You need to check this and removed any bogus entries that may exist
due to an incomplete "make distclean"

In addition, the specfile:

      <install_dir>/dist/lfs-5.1/[<subdir>/]<pkg>.spec

had been edited, and an entry for the new patch has been put in there, a
backup of the original specfile is in

      <install_dir>/dist/lfs-5.1/[<subdir>/]<pkg>.spec.bak
```

- The suffix is the time since the epoch, decoded thusly :

```
$ perl -e 'print scalar localtime(1145390352)'
Fri Apr 14 21:59:12 2006
```

Patch Generation
Capturing Package Source Changes (cont.)

- The package source tree with changes will be relocated to :

  `./rpm/BUILD/<pkg>.modified`

- The unchanged source tree is reinstalled and a **`diff`** is run against the **`<pkg>.modified`** tree, thereby generating the patch contents

- When a patch is good to go:
  - rename the patch filename to something meaningful (thereby removing the timestamp from the filename)
  - add comments to the **`<pkg>_<meaningful>.patch`** file body
  - also change the **`<pkg>.spec`** file to reference the new patch filename

- When rebuilding later, the package source tree will be recreated from the original package sources and patches + the modifications contained in the new patch

Patch Generation

Example : Adding a Patch to the *u-boot-1.1.3.spec* file

- When a change to u-boot sources is captured in a patch, the u-boot-1.1.3.spec file is automatically edited as follows :

```
%define pfx /opt/freescale/rootfs/%{_target_cpu}

Summary         : Universal Bootloader firmware
Name            : u-boot
Version         : 1.1.3
Release         : 1
License         : GPL
Vendor          : Freescale
Packager        : Olivia Yin
Group           : Applications/System
Source          : %{name}-%{version}.tar.bz2
Patch0          : u-boot-1.1.3-mpc83xx.patch
Patch1          : u-boot-1.1.3-1145390352.patch
BuildRoot       : %{_tmppath}/%{name}
Prefix          : %{pfx}

%Description
%{summary}

All source and patches from Freescale.
```

Patch Generation

Example : Adding a Patch to the *u-boot-1.1.3.spec* file (cont.)

```
%Prep
%setup -n %{name}-%{version}
%patch0 -p1
%patch1 -p1
%Build

PKG_U_BOOT_CONFIG_TYPE=${PKG_U_BOOT_CONFIG_TYPE:-MPC8349ADS_config}
make HOSTCC="$BUILDCC" CROSS_COMPILE=$TOOLCHAIN_PREFIX $PKG_U_BOOT_CONFIG_TYPE
make HOSTCC="$BUILDCC" HOSTSTRIP="$BUILDSTRIP" \
     CROSS_COMPILE=$TOOLCHAIN_PREFIX $PKG_U_BOOT_BUILD_ARGS all

%Install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%{pfx}/boot
for i in u-boot.bin u-boot
do
    cp $i $RPM_BUILD_ROOT/%{pfx}/boot
done

%Clean
rm -rf $RPM_BUILD_ROOT

%Files
%defattr(-,root,root)
%{pfx}/*
```

LTIB RFS Operations
Repopulating All or Part of the RFS

- Remove all installed packages from **`./rootfs`**, then invoke **`./ltib`** to deploy all configured packages to the RFS

```
$ ./ltib -m clean
$ ./ltib
```

- To remove a single package (but careful, as other packages may depend on it) :

```
$ ./ltib -m clean -p <pkg>
```

- To redeploy just one package to the RFS

```
$ ./ltib --reinstall -p <pkg>
```

LTIB Special Use Cases
Full BSP Builds

- How do I autobuild a complete BSP?

  To do an **unattended** build of the **default configuration** :

```
./ltib --preconfig config/platform/<platform>/defconfig \
   --batch
```

- How do I autobuild a BSP but with a **full package list** (for testing) ?

```
./ltib --preconfig config/platform/<platform>/defconfig \
   --profile max.config --batch
```

LTIB Special Use Cases
Publishing a New BSP

- Remove any development config options and non-default packages
  - Configure LTIB to the default state you want to adopt for the release. Make sure you turn off any development configuration options (such as **CONFIG_PKG_KERNEL_WANT_CF**)
- Force re-build the image
  - Make sure you force re-build at least all packages that are turned on by the default configuration.

```
$ ./ltib -f
```

LTIB Use Case
Publishing a New BSP (cont.)

- Make sure your image boots and runs normally on the target

- Save your work (config files, ...).
  All **`.dev`** files should be moved to non-**`.dev`** versions

```
cp config/platform/<platform_name>/defconfig.dev \
   config/platform/<platform_name>/defconfig
                                  . "

cp config/platform/<platform_name>/\
  linux-2.6.<version>-<platform_name>-<config>.dev> \
  config/platform/<platform_name>/linux-2.6.<version>-\
  <platform>-<config>.config
```

LTIB Tips and Tricks
Discarding Package Changes

- Earlier versions of LTIB, by default LTIB will refuse to overwrite (clobber) existing package sources in **`./rpm/BUILD`**, and terminate with an error :

```
Processing: pcre
==================
Cowardly refusing to clobber existing directory:
              /opt/ltib-bsps/mxc/rpm/BUILD/pcre-6.3
Remove this by hand if you really want to rebuild this package from
scratch
Died at ./ltib line 565.
Started: Fri Sep 22 10:35:14 2006
  Ended:   Fri Sep 22 10:35:19 2006
  Elapsed: 5 seconds
Build Failed
Exiting on error or interrupt
```

- Very recent versions of LTIB likewise will not overwrite existing sources, but gracefully continue building the already “prep-ed” package: **`scbuild/scdeploy already unpacked package`**

LTIB Tips and Tricks
Discarding Package Changes (cont.)

- The very existence of a package source tree prior to running LTIB, means uncaptured source modifications *might* remain.
  Either ...
  - ... remove the source tree with **`rm -rf`** (dangerous, your decision), or rename it with **`mv`** (prudent), then force rebuild to verify the build is still OK
  - ... capture your modifications in a patch, then manually delete the directory.
    Your modifications will be applied automatically the next time the package source is re-installed and re-built.
  - ... **`-m scinstall`** the package first, to assure the package is up-to-date, so it does not need to be rebuilt and therefore the check for existing sources does not need to happen

LTIB Tips and Tricks
Skipping Errors

- How do I get LTIB to carry on if there are errors?
  - Add **`-C`** or **`--continue`** option to the LTIB command line.

    Note the build may fail later on, because the package that caused the error in the first place may be needed by another package

LTIB Tips and Tricks

Adding a Package or Externally Built Files to the RFS

- How to deploy an additional package to the RFS ?

  `$ ./ltib -p <pkg> [--leavesrc]`

  E.g. to add ***`strace`*** to a currently NFS-mounted RFS

  `$ ./ltib -p strace`

  The package will be installed from the binary RPM if available.
  If not, the package will be first rebuilt from source.

- Add externally built files to LTIB, by putting them into the **`./config/platform/<platform>/merge`** directory.

  The merge package will detect these additions and adds them to the **`merge`** package rpm, to bring the merge payload under package management

Adding Packages to a BSP
Updating to the Configuration System

- Edit **`./config/userspace/packages.lkc`**, (alphabetical order) e.g. :

```
config PKG_STRACE
        bool "strace"
```

- Edit **`./dist/lfs/common/pkg_map`** (in build order).
  Put your package where it should go in the build order, and add an entry that ties the config key to the directory containing the **`.spec`**-file for the package, e.g. :

```
PKG_GDB                 =    gdb
PKG_STRACE              =    strace
```

Adding Packages to a BSP

Setting Up the New Package for the Build System

- Many Open Source packages use the **`./configure`** script to configure the package source tree on the host
- Create a **`.spec`** file for such a package from the template in **`dist/lfs-5.1/template/template.spec`**

  This will cause **`.configure`** and **`make`** to be invoked with appropriate options for cross compilation and installation into the RFS

```
%Build
./configure --prefix=%{_prefix} --host=$CFGHOST --build=%{_build}
make

%Install rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT/%{pfx}
```

- Do a test **`-m scbuild`** ... the package should now be correctly compiled with the cross-compiler for the BSP

Adding Packages to a BSP
When You Have Just a Source Tree

1. Clean your sources : remove any generated files [`.o,.a,.so`]
2. Make a 'tarball', e.g. :

```
cd <my_new_package>-x.y
make clean
cd ..
tar zcvf <my_new_package>-x.y.tar.gz <my_new_package>-x.y
```

3. Move this tarball to the LPP so LTIB can find it

```
mv <my_new_package>-x.y.tar.gz /opt/freescale/pkgs/
```

4. Create a **`.spec`**-file using the existing template

```
mkdir dist/lfs-5.1/<my_new_package>
cp ./dist/lfs-5.1/template/template.spec \
          ./dist/lfs-5.1/<my_new_package>/<my_new_package>.spec
```

5. Edit and fix-up the template to reflect your package :

```
| *Field*  | *Description* |
| Summary  | put in a summary of what the package is/does
| Name     | put in the name of the packge (usually from the tarball name)
| Version  | put in the version (usually from the tarball/directory
| Release  | start at 1 and rev each time you change the spec file
| License  | e.g GPL/LGPL/BSD, look this up in the package's files
| Group    | If this exists on an rpm based machine, copy from rpm -qi <package>
           If not, choose something from /usr/share/
| %Build   | you may need to add *--host=$CFGHOST --build=%{_build}* to the configure clause
```

Cross Compiling An External Driver Module
… Using LTIB

- Background :
  - On module programming : http://tldp.org/LDP/lkmpg/2.6/html/index.html
  - On build mechanics : `Documentation/kbuild/modules.txt`
- Building the example module using LTIB :
  - Source of kernel package must be installed
  - External driver package must have a `.spec` file
  - Module can be build using `ltib` script

```
./ltib -p hello_mod.spec
```

Cross Compiling An External Driver Module
… Standalone (for background)

- Building the example module manually :
  - Source of kernel package must be installed (passed with **`KERNELDIR=<kerneltree>`**)
  - Source of module must be installed (passed with **`M=<moduletree>`**)
  - Cross compiler must be included in PATH envvar

```
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- KERNELDIR=../linux
make ARCH=ppc -C ../linux M=/ltib/rpm/BUILD/hello_mod-1.0
make[1]: Entering directory \
                 `/home/seh/ltib_bsps/savannah_ltib/rpm/BUILD/linux-2.6.13'
  LD      /ltib/rpm/BUILD/hello_mod-1.0/built-in.o
  CC [M]  /ltib/rpm/BUILD/hello_mod-1.0/mod_main.o
  CC [M]  /ltib/rpm/BUILD/hello_mod-1.0/message_mod.o
  LD [M]  /ltib/rpm/BUILD/hello_mod-1.0/modexample.o  Building modules, stage 2.
  MODPOST
  CC      /ltib/rpm/BUILD/hello_mod-1.0/modexample.mod.o
  LD [M]  /ltib/rpm/BUILD/hello_mod-1.0/modexample.ko
make[1]: Leaving directory `/ltib/rpm/BUILD/linux-2.6.13'
$
```

LTIB Tips and Tricks
Getting Additional Output

- To see all the compile arguments, export the following environment variable prior to running LTIB :

```
$ export FS_DEBUG=1
```

- To log the build messages to a file

```
$ ./ltib > log_name 2>&1
```

- To log the output and to be able to see it on the screen:

```
$ ./ltib 2>&1 | tee logfile
```

LTIB Tips and Tricks
Run a Shell with Spoofing Set Up

- To easily work with the cross compilation tool chain from a shell prompt, start LTIB in shell mode :

```
$./ltib -m shell
```

In this spoofed shell environment :

- tool chain components are aliased to the cross compilation tools
- the project interface area `./rootfs/usr/{lib,include}` is wired into the compiler

```
$ ./ltib -m shell
Entering ltib shell mode, type 'exit' to quit
LTIB> gcc --version
powerpc-linux-gcc (GCC) 3.4.3
Copyright (C) 2004 Free Software Foundation, Inc.
[...]
LTIB> exit
exit
```

LTIB Tips and Tricks
RPM and the Root File System

- LTIB creates binary RPMs that are subsequently used to install packages into the target's RFS
- You can use the **rpm** utility to query the packages installed in the RFS.
  Note this is different from requesting the package list from LTIB

```
$ /opt/freescale/ltib/usr/bin/rpm -dbpath ./rpmdb -qa
```

  The exact equivalent command in an LTIB shell :

```
LTIB> rpm -qa
```

References

- LTIB Twiki Pages : *<BSP_ISO>/docs/LtibHome*

LtibPrerequisites	Host dependencies
LtibInstall	Installation Guide
LtibFaq	Frequently Asked Questions
LtibFeatures	Feature comparison (vs. Debian/ELDK/Buildroot/PCS)
LtibPkgInfo	Referenced package information
LtibConvertPcsBsp	HOWTO: Convert a PCS BSP
LtibPackagePool	LTIB package pool (PPP/GPP) info and policies
LtibReleaseProcess	HOWTO: make a BSP release

- Public mailing list

  http://lists.nongnu.org/mailman/listinfo/ltib

- LTIB will be publicly accessible on

  http://www.bitshrine.org

freescale™
semiconductor