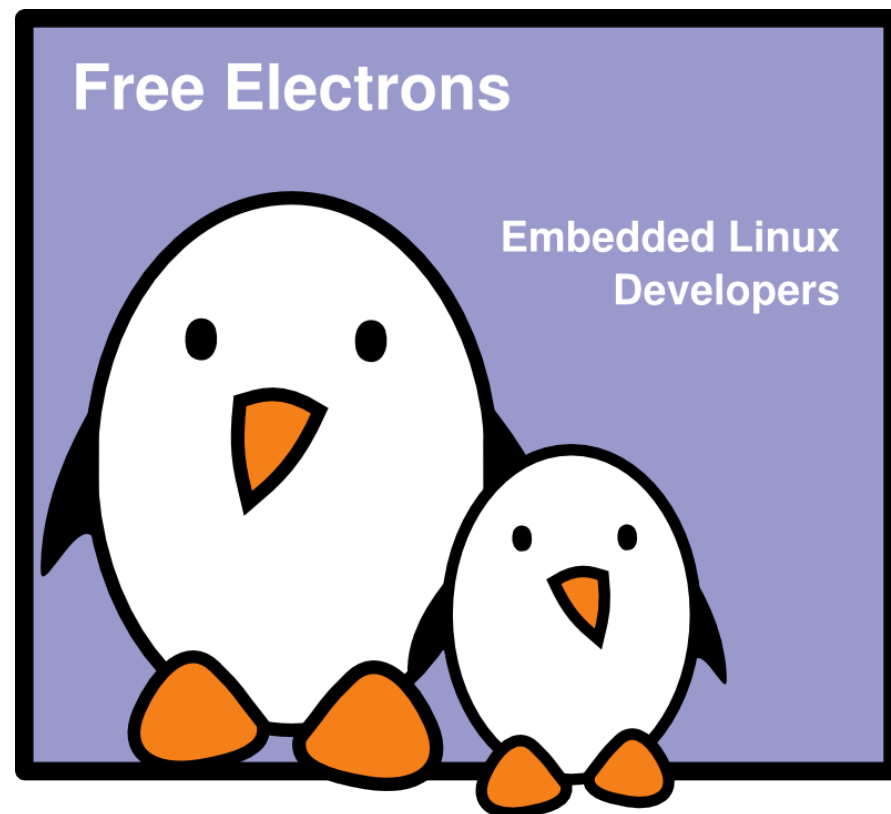
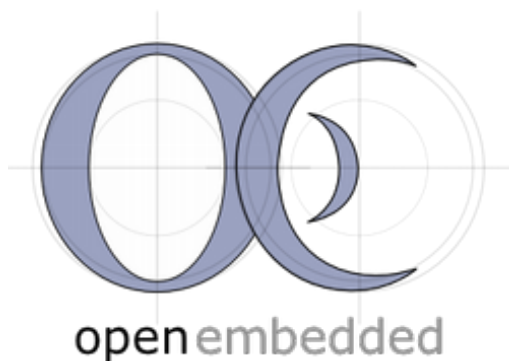




OpenEmbedded

Michael Opdenacker
Thomas Petazzoni
Free Electrons





Rights to copy

© Copyright 2004-2009, Free Electrons
feedback@free-electrons.com

Document sources, updates and translations:
<http://free-electrons.com/docs/openembedded>

Corrections, suggestions, contributions and
translations are welcome!

Latest update: Sep 15, 2009



Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions



Attribution. You must give the original author credit.

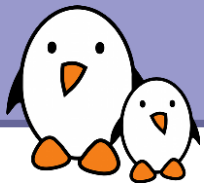


Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



OpenEmbedded history

- ▶ Started as the build system for the OpenZaurus distribution for Sharp Zaurus PDAs.
- ▶ Then developers of the Familiar distribution for PDAs, decided to share their development efforts with OpenZaurus. This increased the project momentum and made it more generic.
- ▶ Then, other embedded distributions started to adopt OE too: Unslug, OpenSimpad, GPE Phone Edition, Ångström, OpenMoko...

See <http://oe.linuxtogo.org/wiki/SuccesStories> for an impressive list of projects relying on OE.



OpenEmbedded details (1)

- ▶ **bitbake**: self contained cross-compiling and building environment for embedded devices.
- ▶ **OpenEmbedded**: collection of bitbake recipes (metadata) describing how to build:
 - ▶ Packages, for thousands of tools (applications, libraries, kernels, bootloaders...). In May 2008:
1600 packages in `packages/<tool>/`
5100 package versions in `packages/<tool>/*.bb`
 - ▶ Target machines. May 2008:
161 machines defined in `conf/machine/`
 - ▶ Distributions: machine and package configurations. May 2008:
34 machines defined in `conf/distro`



BitBake features (1)

Generates **everything** from scratch, using package descriptions.

- ▶ Implemented in Python.
Created from [Gentoo's emerge](#) tool. Still many similarities.
- ▶ Fetches sources from the Internet, either from tarballs or from source control repositories ([svn](#), [cvs](#), [git](#)...). It can also use source mirrors.
- ▶ Applies patches, contained in package descriptions.
- ▶ By default, builds the latest version of all components (unless specific versions are chosen by the distribution).
- ▶ Even builds the compiler and cross-compiler versions you specified, as well as configuration tools ([autoconf](#)...).

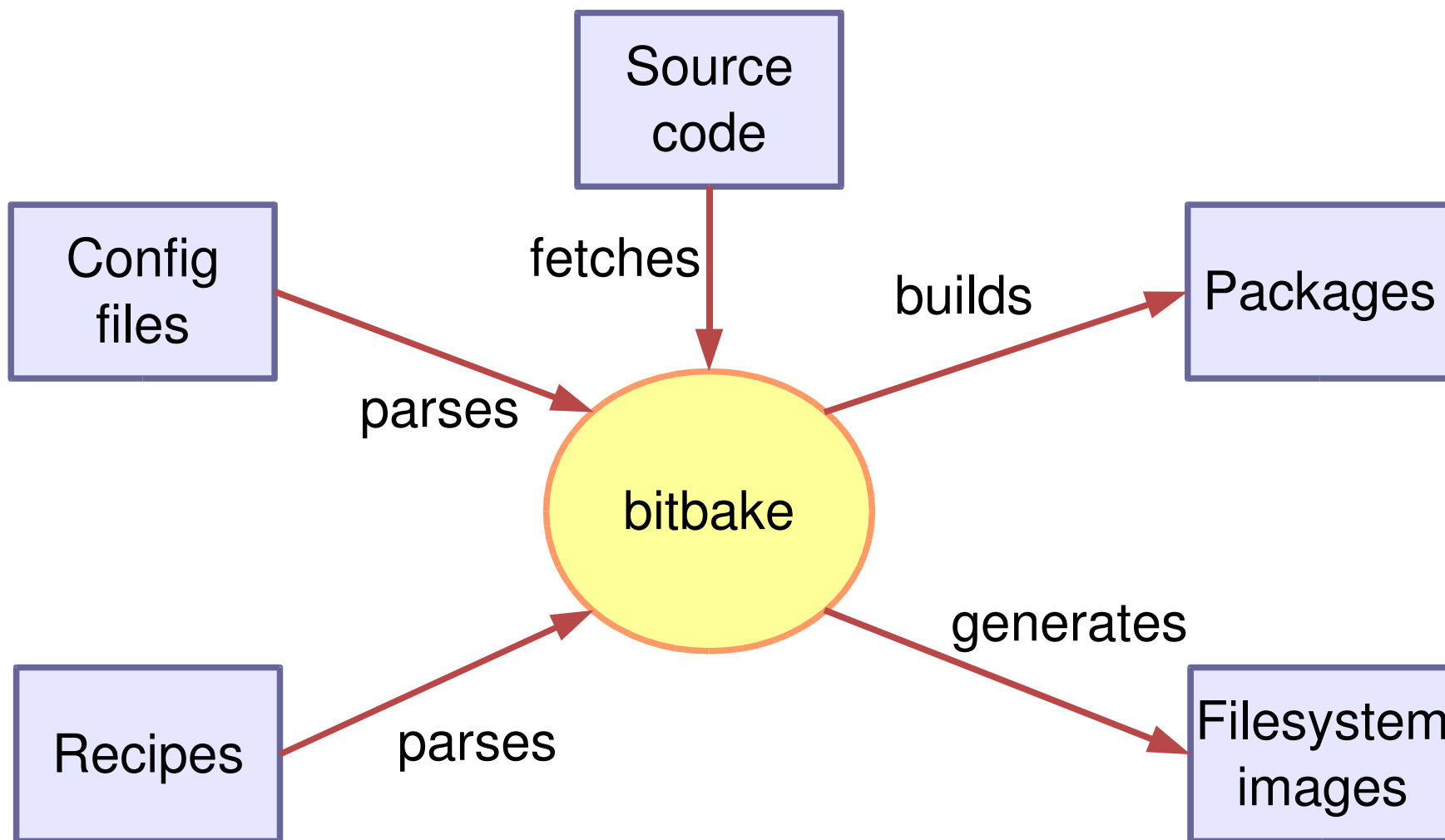


BitBake features (2)

- ▶ Configures, compiles and deploys (copies to the root filesystem and creates packages), including the C library.
- ▶ Supports both `glibc` or `uClibc`!
- ▶ Can be used to compile for several architectures in parallel. Need to duplicate build directories though.
- ▶ Builds filesystem images by creating and installing packages:
Supports several package formats: `.rpm`, `.ipk`, `.deb`
- ▶ Can be used to (cross)compile a single package:
`bitbake dropbear`



bitbake's operation



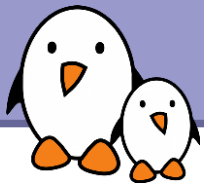


Using OpenEmbedded



Workspace requirements

- ▶ Use a GNU/Linux workstation with at least 512 MB of RAM (1 GB recommended). Otherwise, runs are extremely long (because of swapping).
- ▶ Choose a partition with at least 10 GB of free space (**bitbake** doesn't clean up sources after compiling)
- ▶ Create a working directory with no symbolic link above:
> `mkdir $HOME/oe/`
- ▶ You are also going to need a lot of time:
OE takes hours to run!



Installing required software

Instructions tested on Ubuntu 8.04

► Tools to fetch code from source repositories:

```
> apt-get install wget curl  
> apt-get install cvs subversion monotone git
```

► Compiler related tools:

```
> apt-get install m4 make ccache sed bison
```

► Python Just-In-Time compiler (accelerates bitbake):

```
> apt-get install python-psyco
```

► Documentation related tools:

```
> apt-get install diffstat gawk help2man texi2html texinfo
```

See <http://oe.linuxtogo.org/wiki/RequiredSoftware>
if anything changes in the future.



Getting bitbake

Get the latest stable version of bitbake:

```
> cd $HOME/oe/  
> svn co svn://svn.berlios.de/bitbake/branches/bitbake-1.8/ bitbake
```

Keep bitbake updated:

```
> cd $HOME/oe/bitbake  
> svn update
```

Add bitbake to the Unix PATH:

```
export PATH=$HOME/oe/bitbake/bin:$PATH
```

You may check whether a new stable branch is available:
see <http://oe.linuxtogo.org/wiki/GettingStarted> for details.



Getting the OpenEmbedded database (1)

The OE database is available through a Monotone repository:

- ▶ Get the latest snapshot of the database

```
> cd $HOME/oe  
> wget http://www.openembedded.org/snapshots/OE.mtn.bz2  
> bunzip2 -d OE.mtn.bz2
```

Warning: this file is big (170 MB as of May 23, 2008)

- ▶ Choose a development branch from the list on <http://oe.linuxtogo.org/wiki/DevelopmentBranches>
- ▶ Let's assume that you chose the `org.openembedded.stable` branch (recommended).



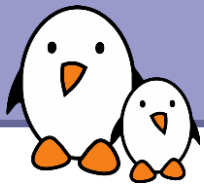
Getting the OpenEmbedded database (2)

- ▶ Update your local copy of the OE database:

```
> mtn --db=OE.mtn pull monotone.openembedded.org  
org.openembedded.stable
```
- ▶ Now checkout your local copy of the OE tree:

```
> mtn --db=OE.mtn checkout --branch=org.openembedded.stable
```
- ▶ It is recommended to keep your database updated frequently
(as it gets multiple updates per day):

```
> cd $HOME/oe/org.openembedded.stable  
> mtn update
```



Create a local configuration

Create a local configuration by starting from the supplied template:

```
> cd $HOME/oe/  
> mkdir -p build/conf  
> cp org.openembedded.stable/conf/local.conf.sample \  
build/conf/local.conf  
> vi build/conf/local.conf
```



Local configuration

Read comments in the `build/conf/local.conf` file carefully!
You should have at least the following 3 settings...

- ▶ Define where `.bb` files are available:

```
BBFILES = "${HOME}/oe/org.openembedded.stable/packages/*/*.bb"
```

- ▶ Choose a machine from
`org.openembedded.stable/conf/machine/`
or create a new one. For example:

```
MACHINE = "at91sam9263ek"
```

- ▶ Choose a distribution from
`org.openembedded.stable/conf/distro/`
or create a new one. For example:

```
DISTRO = "angstrom-2008.1"
```



Extra useful configuration settings

- ▶ Specify a project-wide download directory or mirror (to save downloading time or to work behind a firewall):

```
DL_DIR = "/work/project/oe/sources"
```

- ▶ Enable parallel compiling and processing:

```
PARALLEL_MAKE = "-j 4"
```

```
BB_NUMBER_THREADS = "4"
```

- ▶ Specify filesystem images to build:

```
IMAGE_FSTYPES = "ext2 tar"
```




Set up the environment

- ▶ Create and source an environment setting script (example)

```
export PATH=$HOME/oe/bitbake/bin:$PATH
```

```
export BBPATH=$HOME/oe/build:$HOME/org.openembedded.stable
```

- ▶ Tweak for Ubuntu Hardy:

```
> echo 0 > /proc/sys/vm/mmap_min_addr
```

- ▶ You are ready to start, at last!



Running bitbake

- ▶ Pick up a default system image in `org.openembedded.stable/packages/images/` or make your own. Several images, such as `opie-image` (Qt based) or `gpe-image` (GTK based) can be available for the same core distribution.
- ▶ Run `bitbake` to create your system image:
`bitbake <your-system-image>`
- ▶ Now, be patient!
It will take hours for your machine to complete the job.



Using bitbake behind a proxy

- ▶ **bitbake** retrieves sources for a significant number of packages directly through **CVS**.
- ▶ **CVS** port often blocked by proxies
- ▶ In this case, can use an **HTTP** mirror with tarballs generated every day. Just an example (the below URL doesn't seem to work any more):
`CVS_TARBALL_STASH = "http://oesources.org/source/current/"`
- ▶ Highly protected networks with no outside **HTTP** access can set up their own mirror.



Overlay directory for custom settings (1)

- ▶ **bitbake** parses all configuration and recipe files found in directories specified by **BBPATH** environment variable.
- ▶ Suggestion: make an overlay directory holding:
 - ▶ Specific configuration files
 - ▶ Project specific packages
 - ▶ Any redefinition of OE files: package (bb) files, configuration files, classes



Overlay directory for custom settings (2)

- ▶ You should set the BBPATH variable as follows:
`export BBPATH=$HOME/oe/overlay:
$HOME/oe/org.openembedded.stable`
- ▶ The `overlay/` directory should have the below structure:
 - ▶ `conf/`: custom and redefined configuration files
 - ▶ `packages/`: project specific and redefined bb files.



Configuration and package files



Distro configuration files

Highest level of configuration files. They define:

- ▶ Toolchain and package versions
- ▶ Package configuration.
Packages can be built in several ways. The distro chooses which one.
- ▶ Distribution information variables
- ▶ High level settings: filesystem formats, use udev, etc.



conf/distro/gmustix.conf

```
#@TYPE: Distribution
#@NAME: GMustix
#@DESCRIPTION: Gumstix distribution for GMU (George Mason University)

INHERIT += "package_tar package_ipk"
TARGET_OS = "linux-uclibc"
TARGET_FPU = "soft"
IMAGE_FSTYPES = "jffs2"

PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}gcc-initial:gcc-cross-initial"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}gcc:gcc-cross"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}g++:gcc-cross"
PREFERRED_PROVIDERS += " virtual/${TARGET_PREFIX}libc-for-gcc:uclibc"

PREFERRED_PROVIDER_classpath = "classpath-minimal"

PREFERRED_VERSION_gcc-cross-initial = "3.4.4"
PREFERRED_VERSION_gcc-cross = "3.4.4"
PREFERRED_VERSION_gcc-cross-sdk = "3.4.4"
PREFERRED_VERSION_gcc = "3.4.4"

PREFERRED_VERSION_ipkg-native = "0.99.160"
PREFERRED_VERSION_qemu-native = "0.8.0"

DISTRO_VERSION = "uno"

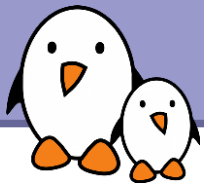
IPK_EXCLUDE_SOURCE = "1"
```




Machine configuration files

Defines board specific features, such as

- ▶ Architecture
- ▶ Compiler version and CPU instruction set / optimizations (such as `armv5te` and `i686`).
- ▶ Kernel version and package provider
- ▶ Board specific libraries, services and utilities.



conf/machine/at91sam9263ek.conf

```
#@TYPE: Machine
#@Name: Atmel AT91SAM9263EK Development Platform
#@DESCRIPTION: Machine configuration for the at91sam9263ek development
board with a at91sam9263 processor

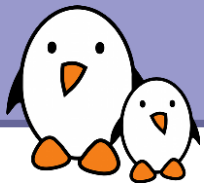
TARGET_ARCH = "arm"
PACKAGE_EXTRA_ARCHS = "armv4t armv5te"
PREFERRED_PROVIDER_virtual/kernel = "linux"
PREFERRED_PROVIDER_xserver = "xserver-kdrive"

#don't try to access tty1
USE_VT = "0"

MACHINE_FEATURES = "kernel26 alsa ext2 usbhost usb gadget screen"
# used by sysvinit_2
SERIAL_CONSOLE = "115200 ttyS0"
IMAGE_FSTYPES ?= "jffs2"
EXTRA_IMAGECMD_jffs2 = "--pad --little-endian --eraseblock=0x20000 -n"

require conf/machine/include/tune-arm926ejs.inc

KERNEL_IMAGETYPE = "uImage"
```



bitbake recipe files

.bb files:

- ▶ Declare environment variables needed by **bitbake** to build a package (mainly package related information).
- ▶ Declare methods to execute to build a package: fetching, configuring, compiling, installing...



4 types of bb files

- ▶ Classes

Common steps for a class of packages.

Example: all **Qtopia Core** packages are built in the same way (with **qmake**)

- ▶ Packages

Inherit classes and add or override specific settings or steps.
Also declare package dependencies.

- ▶ Tasks

Define collections of packages to be built

- ▶ Images

Create filesystem images from tasks.



classes/gnome.bbclass

```
def gnome_verdir(v):
    import re
    m = re.match("^[0-9]+\.[0-9]+", v)
    return "%s.%s" % (m.group(1), m.group(2))

SECTION ?= "x11/gnome"
SRC_URI = "${GNOME_MIRROR}/${PN}/${@gnome_verdir("${PV}")}/${PN}-${PV}.tar.bz2"

DEPENDS += "gnome-common"

FILES_${PN} += "${datadir}/application-registry ${datadir}/mime-info \
    ${datadir}/gnome-2.0"

inherit autotools pkgconfig gconf

gnome_stage_includes() {
    autotools_stage_includes
}
```



packages/dillo/dillo_0.8.6.bb

```
DESCRIPTION = "Lightweight gtk+ browser, enhanced version, with support for SSL,  
frames, tabs and much more..."  
HOMEPAGE = "http://www.dillo.org"  
SECTION = "x11/network"  
PRIORITY = "optional"  
LICENSE = "GPL"  
DEPENDS = "gtk+-1.2 libpng openssl"  
RCONFLICTS = "dillo2"  
PR = "r2"  
SRC_URI="http://www.dillo.org/download/dillo-${PV}.tar.bz2 \  
        file://dillo-il8n.diff;patch=1 \  
        file://dillo.desktop \  
        file://dillo.png"  
  
S = "${WORKDIR}/dillo-${PV}/"  
  
inherit autotools pkgconfig  
  
FILES_${PN} += " /usr/lib/dillo/ /usr/bin/dpid /usr/bin/dpidc "  
FILES_${PN}-dbg += " ${libdir}/dillo/dpi/*.debug/"  
  
export PNG_CONFIG = "${STAGING_BINDIR_CROSS}/libpng-config"  
  
EXTRA_OECONF = "--disable-dlgui --disable-anti-alias"  
  
do_install_append() {  
    install -d ${D}${datadir}/applications  
    install -d ${D}${datadir}/pixmaps  
    install -m 0644 ${WORKDIR}/dillo.desktop ${D}$  
{datadir}/applications/dillo.desktop  
    install -m 0644 ${WORKDIR}/dillo.png ${D}${datadir}/pixmaps/dillo.png  
}
```



OE tasks

Typical classes:

- ▶ Base: basic user space applications to boot to a command line prompt: C library, BusyBox, init scripts, sshd.
- ▶ Core: core tools and libraries needed by the applications.
- ▶ Apps / UI: class of applications corresponding to product applications.



packages/task/task-gpe-games.bb

```
DESCRIPTION = "Games task package for GPE Palmtop  
Environment"
```

```
PR = "r6"
```

```
LICENSE = "MIT"
```

```
inherit task
```

```
RDEPENDS_${PN} = "\  
    gpe-go \  
    gpe-lights \  
    gpe-othello \  
    gpe-tetris \  
    gsoko \  
    xdemineur"
```




OE images

- ▶ Specify what goes into the root filesystem
- ▶ Image types defined in the distro or local configuration
 - ▶ Flash file systems to store on flash storage: [jffs2](#)
 - ▶ Other filesystem images for block storage: [ext2](#), [ext3](#), [squashfs](#)
 - ▶ Tarballs for nfsroot, initramfs or init ramdisks.
- ▶ Remember that root filesystems are created from packages.
- ▶ Images files are created in `tmp/deploy/images/`



packages/images/openmoko-image.bb

```
#-----  
# OpenMoko Image Recipe  
#-----  
  
IMAGE_LINGUAS = ""  
  
IMAGE_INSTALL = "\n  
    ${MACHINE_TASK_PROVIDER} \n  
    task-openmoko-linux \n  
    task-openmoko-net \n  
    task-openmoko-ui \n  
    task-openmoko-base \n  
    task-openmoko-phone \n  
    task-openmoko-games \n  
    task-openmoko-pim \n  
"  
  
DEPENDS = "\n  
    ${MACHINE_TASK_PROVIDER} \n  
    task-openmoko \n  
"  
  
inherit image  
  
ROOTFS_POSTPROCESS_COMMAND += 'date "+%m%d%H%M%Y" >$  
{IMAGE_ROOTFS}/etc/timestamp'
```



How to develop applications with OE

- ▶ When you build a new application for an OE based system, OE is a too heavy environment to compile your application every time you update the sources.
- ▶ With OE, it's possible to export a standalone SDK which can be used to develop your application.
- ▶ The SDK can be shared by the whole team.
- ▶ Of course, at the end, you can add a recipe for your package and build your system with it.

See [packages/meta/meta-toolchain-gpe.bb](#) and [packages/meta/meta-toolchain.bb](#) for more information.



Conclusions and references



OpenEmbedded strengths

- ▶ Best tool for embedded distribution makers, targeting multiple devices of the same kind, or generic devices open to third-party / community applications.
- ▶ Allows developers to share their experience building common libraries and utilities, and compete only on the embedded system itself!
- ▶ Brings determinism and reproducibility in product development (provided you froze the versions of all the components!).
- ▶ Big user community. Look at the [packages/](#) directory and discover packages you didn't know about!



OpenEmbedded drawbacks

- ▶ Very long runs.!!!
- ▶ Too complex filesystems sometimes
You may want to have the applications, without the packages (to save space in very small systems).
- ▶ Builds generic filesystems by default. Good for PDAs and phones, or systems which are meant to be general purpose. A lot of overrides needed to generate a system with a very limited purpose, without creating components that you won't use.
- ▶ Builds all interface translations by default!
- ▶ Powerful but complex. Long learning curve.
Buildroot more difficult to customize but much easier to use.
- ▶ It relies on Monotone!



Weaknesses of generated systems

Too modular

- ▶ Great flexibility and great for development.
- ▶ But too many shell scripts duplicating the same tests (environment variables, existence of files or devices, device information) over and over again, unaware of what has already been tested.

Too generic

- ▶ Generic scripts testing things or features which are not present in the system.
- ▶ Still too close to a generic GNU/Linux system.

All this costs tens of seconds in boot time!
Execution performance is all right though.



Tips for improving system performance

At the end of development with OE, when everything works fine.

- ▶ Remove unneeded and duplicate tests in init scripts.
- ▶ Even better: replace individual init scripts by a single startup one.
- ▶ Remove any file or executable you do not need.
- ▶ Look for duplicate files.
- ▶ Block storage: put all your programs and libraries in a SquashFS partition! Reading files faster can boost startup time too.




BitBake / OE resources

- ▶ ELC 2008 presentation, by Matthew Locke:
http://www.celinux.org/elc08_presentations/mlocke-elc2008-oe.pdf
- ▶ First Project, a very nice testimonial and getting started instructions:
<http://oe.linuxtogo.org/node/65/diff/416/417>
- ▶ OpenEmbedded getting started guide:
<http://oe.linuxtogo.org/wiki/GettingStarted>
- ▶ BitBake user manual
<http://bitbake.berlios.de/manual/>
- ▶ OpenEmbedded Wiki:
<http://oe.linuxtogo.org/wiki>
- ▶ OE mailing lists
<http://www.openembedded.org/contact>



Related documents



Free Electrons

Embedded Freedom

HOME DEVELOPMENT SERVICES TRAINING DOCS COMMUNITY COMPANY BLOG

Recent blog posts

ELC Europe in Grenoble

Free Electrons at ELC

Linux kernel 2.6.29 - New features for embedded users

The Buildroot project begins a new life

FOSDEM 2009 videos

USB-Ethernet device for Linux

Program for Embedded Linux Conference 2009 announced

Public session changes


Real hardware in our training sessions

Call for presentations for the LSM embedded track

Docs

Most of the below documents are presentations used in our [training sessions](#), or in technical conferences.

License

 All our documents are available under the terms of the [Creative Commons Attribution-ShareAlike 3.0 license](#). This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

Linux kernel

- [Embedded Linux kernel and driver development](#)
- [New features in Linux 2.6](#) (since 2.6.10)
- [Kernel initialization](#)
- [Porting Linux to new hardware](#)
- [Power management in Linux](#)
- [Linux PCI drivers](#)
- [Block device drivers](#)
- [Linux USB drivers](#)
- [DMA](#)

Architecture specific documents

- [ARM Linux specifics](#)
- [Linux on TI OMAP processors](#)

Embedded Linux system development

- [Embedded Linux system development](#)
- [Real time in embedded Linux systems](#)
- [Block filesystems](#)
- [Flash filesystems](#)
- [Free software development tools](#)
- [The U-boot bootloader](#)
- [The GRUB bootloader](#)
- [The blob bootloader](#)
- [Hotplugging with udev](#)
- [Introduction to uClinux](#)
- [Java in embedded Linux](#)
- [Embedded Linux optimizations](#)
- [Audio in embedded Linux systems](#)
- [Multimedia in embedded Linux systems](#)
- [Embedded Linux From Scratch... in 40 minutes!](#)
- [Building embedded Linux systems with Buildroot](#)
- [Developing embedded distributions with OpenEmbedded](#)
- [The Scratchbox development environment](#)

Miscellaneous

- [Introduction to the Unix command line](#)
- [SSH](#)
- [Linux virtualization solutions](#) (with an embedded perspective)
- [Advantages of Free Software and Open Source in embedded systems](#)
- [Introduction to GNU/Linux and Free Software](#)

All our technical presentations
on <http://free-electrons.com/docs>

- ▶ Linux kernel
- ▶ Device drivers
- ▶ Architecture specifics
- ▶ Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- ▶ By sending corrections, suggestions, contributions and translations
- ▶ By asking your organization to order development, consulting and training services performed by the authors of these documents (see <http://free-electrons.com/>).
- ▶ By sharing this document with your friends, colleagues and with the local Free Software community.
- ▶ By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

- Linux device drivers
- Board support code
- Mainstreaming kernel code
- Kernel debugging

Embedded Linux Training

All materials released with a free license!

- Unix and GNU/Linux basics
- Linux kernel and drivers development
- Real-time Linux, uClinux
- Development and profiling tools
- Lightweight tools for embedded systems
- Root filesystem creation
- Audio and multimedia
- System optimization

Free Electrons

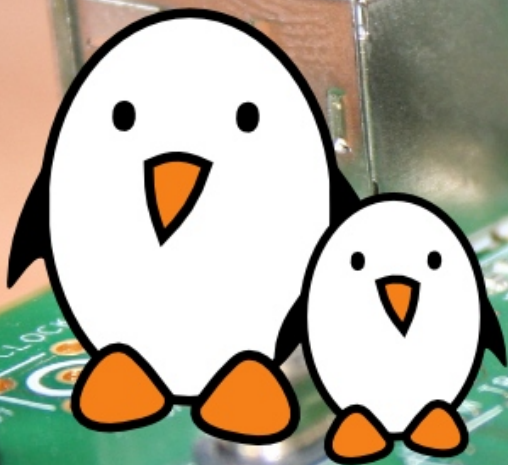
Our services

Custom Development

- System integration
- Embedded Linux demos and prototypes
- System optimization
- Application and interface development

Consulting and technical support

- Help in decision making
- System architecture
- System design and performance review
- Development tool and application support
- Investigating issues and fixing tool bugs



Free Electrons
Embedded Linux Experts

<http://free-electrons.com>