Using ELBE To Build
Debian Based Embedded Systems
Embedded Linux Conference Europe - Berlin, Germany

Manuel Traut
Linutronix GmbH

October 12, 2016
Agenda

1. Why should I use Debian / ELBE
2. ELBE Features
3. Example: Beaglebone Black SD Card Image
4. Own Software with ELBE pbuilder
5. Customize Image
BUILD YOUR OWN LINUX DISTRO
Build Your Own Distro

it might be harder than you think

YOU
- need to wait until every package and its dependencies are compiled
- are probably the only one testing the resulting binaries
- are alone with the bugs in your binary packages
- need to track security of your distro

...why not let OTHERS do the job and focus on your application?
Who are the OTHERS?
Why Debian based?

Manuel Traut
Linutronix GmbH
Why elbe?

ftp.debian.org

debootstrap

debian RFS

scripts / manual modifications: rm, cp, mv, ..

customized RFS

parted, mtd-utils,
qemu-img, e2-tools, ..

Firmware Image
Why elbe?

brilliant
Firmware
with no bugs

Why elbe?

is just python code that uses Debian infrastructure and tools

Manuel Traut
Linutronix GmbH
Why elbe?

elbe is just python code that uses Debian infrastructure and tools
Project description is in a single XML file
Integrate Debian binary packages
Build own software components
Customize default config files
Produce flashable images
Regenerate images and build environment
ELBE Highlights

ELBE-XML
Project description

Packages
Debian Repositories

Images
Rebuild-CD
Quellcode-CD
Licences

native ADK
Application Development Kit
ELBE Versioning

ELBE stable

The current elbe stable release is elbe 1.1
- target images for armel, armhf, i386, amd64 and powerpc
ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)

Manuel Traut
Linutronix GmbH
ELBE Versioning

ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)
- extract archive over target RFS (e.g. for config files)
ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, …)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
- CDROM with all used Debian source packages
ELBE Versioning

ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
- CDROM with all used Debian source packages
- CDROM with all used Debian binary packages (target and initvm)
ELBE stable

The current elbe stable release is elbe 1.1
- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
- CDROM with all used Debian source packages
- CDROM with all used Debian binary packages (target and initvm)
- licence.txt with all LICENCES used on the target
ELBE stable

The current elbe stable release is elbe 1.1

- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, …)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
- CDROM with all used Debian source packages
- CDROM with all used Debian binary packages (target and initvm)
- licence.txt with all LICENCES used on the target
- check for updates of a generated target
The current elbe stable release is elbe 1.1
- target images for armel, armhf, i386, amd64 and powerpc
- MSDOSHD, GPT, tar.gz, cpio, squashfs, UBI/UBIFS support
- finetune target RFS (cp, mv, command, ...)
- extract archive over target RFS (e.g. for config files)
- target related sysroots for cross-compile toolchains
- CDROM with all used Debian source packages
- CDROM with all used Debian binary packages (target and initvm)
- licence.txt with all LICENCES used on the target
- check for updates of a generated target
- different modes to shrink the target image
ELBE Versioning

ELBE testing

The next major release will be elbe 2.0
The current testing releases are called elbe 1.9.x
New features in ’testing’:

- elbe-pbuilder for building own debian packages
ELBE testing

The next major release will be elbe 2.0
The current testing releases are called elbe 1.9.x
New features in 'testing':
- elbe-pbuilder for building own debian packages
- aarch64 aka arm64 target support
ELBE testing

The next major release will be elbe 2.0
The current testing releases are called elbe 1.9.x
New features in ’testing’:

- elbe-pbuilder for building own debian packages
- aarch64 aka arm64 target support
- display build log, instead of ’wait project is busy’
ELBE testing

The next major release will be elbe 2.0
The current testing releases are called elbe 1.9.x

New features in ‘testing’:

- elbe-pbuilder for building own debian packages
- aarch64 aka arm64 target support
- display build log, instead of ‘wait project is busy’
- generate SPDX files
The next major release will be elbe 2.0
The current testing releases are called elbe 1.9.x
New features in ’testing’:  
- elbe-pbuilder for building own debian packages 
- aarch64 aka arm64 target support 
- display build log, instead of ’wait project is busy’ 
- generate SPDX files 
- extended partitions support for msdoshd
Install ELBE

Install Debian packages

**ELBE stable**

$ echo "deb http://debian.linutronix.de/elbe jessie main" > /etc/apt/sources.list.d/elbe.list

**ELBE testing**

$ echo "deb http://debian.linutronix.de/elbe-testing jessie main" > /etc/apt/sources.list.d/elbe-testing.list

**common for testing and stable**

$ apt-get update
[...]
$ apt-get install elbe
[...]
Install ELBE

**create an initvm**

```bash
$ elbe initvm create --directory $HOME/elbe-initvm
```

the initvm is

- a minimal Debian installation including 'elbe-daemon'
- used to build the embedded target (firmware) images
- reproducible by using 'bin-cdrom.iso'
  this iso is generated with each target image build
- not started after reboot
  (use 'elbe initvm start --directory $HOME/elbe-initvm')
simple XML for beaglebone black 1/4

```xml
<ns0:RootFileSystem
 xmlns:ns0="https://www.linutronix.de/projects/Elbe"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 created="2016-09-29T18:29:33"
 revision="6"
 xsi:schemaLocation="https://www.linutronix.de/projects/Elbe_dbsfed.xsd">
 <project>
   <name>beaglebone-black-simple</name>
   <version>1.0</version>
   <description>debian jessie rootfs for beaglebone black</description>
   <buildtype>armhf</buildtype>
   <mirror>
     <primary_host>ftp.de.debian.org</primary_host>
     <primary_path>/debian</primary_path>
     <primary_proto>http</primary_proto>
   </mirror>
   <suite>jessie</suite>
 </project>
```
<target>
  <hostname>lxbbbsimple</hostname>
  <domain>linutronix.de</domain>
  <passwd>foo</passwd>
  <console>ttyO0,115200</console>
  <images>
    <msdoshd>
      <name>sdcard.img</name>
      <size>1500MiB</size>
      <partition>
        <size>50MiB</size> <label>boot</label> <bootable/>
      </partition>
      <partition>
        <size>remain</size> <label>efs</label>
      </partition>
    </msdoshd>
  </images>
</target>
simple XML for beaglebone black 3/4

```xml
<fstab>
  <bylabel>
    <label>rfs</label>
    <mountpoint>/</mountpoint>
    <fs>
      <type>ext2</type>
      <tune2fs>-i 0</tune2fs>
    </fs>
  </bylabel>
  <bylabel>
    <label>boot</label>
    <mountpoint>/boot</mountpoint>
    <fs>
      <type>vfat</type>
    </fs>
  </bylabel>
</fstab>
```
simple XML for beaglebone black 4/4

```xml
<finetuning>
  <cp path="/usr/lib/linux-image-3.16.0-4-armmp/am335x-boneblack.dtb"/>
  /boot/dtb-3.16.0-4-armmp</cp>
  <cp path="/usr/lib/u-boot/am335x_boneblack/u-boot.img"/>
  /boot/u-boot.img</cp>
  <cp path="/usr/lib/u-boot/am335x_boneblack/MLO"/>
  /boot/MLO</cp>

  <command>echo "uenvcmd=setenv bootargs ’console=tty00,115200 root=/dev/... ...’" > /boot/uEnv.txt</command>
</finetuning>

<pkg-list>
  <pkg>u-boot-omap</pkg>
  <pkg>linux-image-3.16.0-4-armmp</pkg>
</pkg-list>
</target>
</ns0:RootFileSystem>
```
Build the SDCard Image

using high-level interface:

$ elbe initvm submit --keep-files examples/armhf-ti-beaglebone-black-simple.xml

build output:

$ cd elbe-build
$ ls -lh
-rw-r--r-- 1 manut lx 381M Sep 29 21:38 bin-cdrom.iso
-rw-r--r-- 1 manut lx 1014K Sep 29 21:37 elbe-report.txt
-rw-r--r-- 1 manut lx 1.3M Sep 29 21:37 licence.txt
-rw-r--r-- 1 manut lx 1.6M Sep 29 21:37 licence.xml
-rw-r--r-- 1 manut lx 863K Sep 29 21:37 log.txt
-rw-r--r-- 1 manut lx 1.5G Sep 29 21:37 sdcard.img
-rw-r--r-- 1 manut lx 73K Sep 29 21:37 source.xml
-rw-r--r-- 1 manut lx 1019M Sep 29 21:40 src-cdrom.iso
-rw-r--r-- 1 manut lx 116 Sep 29 21:37 validation.txt
Flash the image to a SD Card

$ zcat sdcard.img.gz | sudo dd of=/dev/mmcblk0 bs=5M
add users and groups

[...]  
<finetuning>
  <addgroup>debian</addgroup>
  <adduser passwd="elbe" shell="/bin/sh" groups="users,debian">
    elbe </adduser>
Customization

Add own software

```xml
[..]
pbuilder>
git revision='a15a83e2649765736aa6bfe9c490a39a417cf69b'>
https://github.com/Linutronix/libgpio.git
</git>
pbuilder>
pkg-list>
pkg>libgpio1 </pkg>
[..]

The project needs to be in a git or svn repository and the source needs to be debianized.
Customization

Add own software

[...]  
<pbuilder>
  <git revision='a15a83e2649765736aa6bfe9c490a39a417cf69b'>
    https://github.com/Linutronix/libgpio.git
  </git>
</pbuilder>
<pkg-list>
  <pkg>libgpio1</pkg>
[...]  

The project needs to be in a git or svn repository and the source needs to be debianized.
See the excellent ’Debians new Maintainers guide’ for details about debianizing your source:
https://www.debian.org/doc/devel-manuals#maint-guide
Use Debian and custom Bootloader / Kernel

- UI: z.B. Webapplication
- Backend: read/write Sensor/Actors sync Data with Cloud
- Userspace: z.B. Debian Libraries, Services
- stable API / HAL / Userspace - Interface
- Linux / Kernel
- Bootloader
- Hardware
Customization

about ELBE pbuilder

- pdebuild and qemu-user is used to build the packages
Customization

about ELBE pbuilder

- pdebuild and qemu-user is used to build the packages
- build-dependencies need to be specified correctly
Customization

about ELBE pbuilder

- pdebuild and qemu-user is used to build the packages
- build-dependencies need to be specified correctly
- build environment is created from the same debian mirrors/releases as the target image
shrink image

- remove man, unneeded locales, package lists in finetuning
- set the `<norecommend>` tag
- use ‘diet’ or ‘tighten’ mode to generate target RFS
Customization

copy mode basics

- the result of debootstrap + pkgs from pkg-list are in a directory called 'chroot'
- the target directory is a copy of the chroot directory
- archive extraction and finetuning runs on the target directory
copy modes

**default**  target is a 1:1 copy of chroot
copy modes

**default**  target is a 1:1 copy of chroot

**diet**  pkg-list = pkg-list + runtime_dependency of each pkg-list entry
Customization

copy modes

**default**  target is a 1:1 copy of chroot

**diet**  pkg-list = pkg-list +
runtime_dependency of each pkg-list entry
file-list = get_files referenced by each pkg-list entry
Customization

copy modes

**default**  target is a 1:1 copy of chroot

**diet**  pkg-list = pkg-list +
  runtime_dependency of each pkg-list entry
  file-list = get_files referenced by each pkg-list entry
  copy file-list to target
Customization

copy modes

**default**  target is a 1:1 copy of chroot

**diet**  pkg-list = pkg-list +
            runtime_dependency of each pkg-list entry
            file-list = get_files referenced by each pkg-list entry
            copy file-list to target

**tightly** same as diet but without resolving
            runtime_dependencies
Resources

elbe website  http://elbe-rfs.org
elbe source  http://github.com/linutronix/elbe
Conclusion

- let Debian do the maintainance work
- use ELBE to integrate and build Debian binary packages
- focus on application development
Thanks for your attention

Contact

Linutronix GmbH
Manuel Traut
Bahnhofstr. 3
88690 Uhldingen (Germany)
eMail manuel.traut@linutronix.de
fon +49 7556 255 99 16