# SBXG Documentation

Jean Guyomarc'h

Jun 24, 2019

# Contents:

SBXG is a build system that generates bootable images for embedded devices. The images generation is highly customizable, but is mainly composed of:

- a bootloader: U-Boot,
- a kernel: Linux,
- and a foreign root file system (e.g. generated with DFT or Debootstrap).

All components but the toolchain are built from source, with a configuration file enforced by version. This allows SBXG users to rely on the sources and their own (or pre-packaged) configurations, instead of a black box downloaded from untrusted sources.

SBXG provides default configurations for some boards, toolchains, kernels and u-boot, to demonstrate its capabilities, but one of its goal is to be able to use opaque (private) user configurations that can leave outside of SBXG (e.g. reside in a dedicated source control repository).

This guide explains in details how to hack SBXG to develop your own configurations, to forge system software for your embedded boards.

# CHAPTER 1

# SBXG Requirements

**Python** SBXG relies on third-party tools to fulfull its duty. Its core is written with Python. Python 2.7 will do, but it is advised to use Python 3.4 or later.

**Make** SBXG bootstraps its build system, by generating a Makefile. Therefore, `make` (only GNU make is tested) is a strong requirement for SBXG.

**Subcomponent** Subcomponent is used to fetch the components that SBXG depends on. It is packaged as a cargo crate, and therefore can be installed directly from cargo.

**mkfs (ext3, vfat)** To generate an image, mkfs (ext3 and vfat) will be required.

**autotools** SBXG will build from sources a package that uses the autotools. As such, the autotools programs needs to be installed (e.g. autoconf, automake, . . . )

**kernel build essentials** SBXG will compile the Linux Kernel and U-Boot. Hence, such a development environment shall be installed.

CHAPTER 2

# Packages Installation

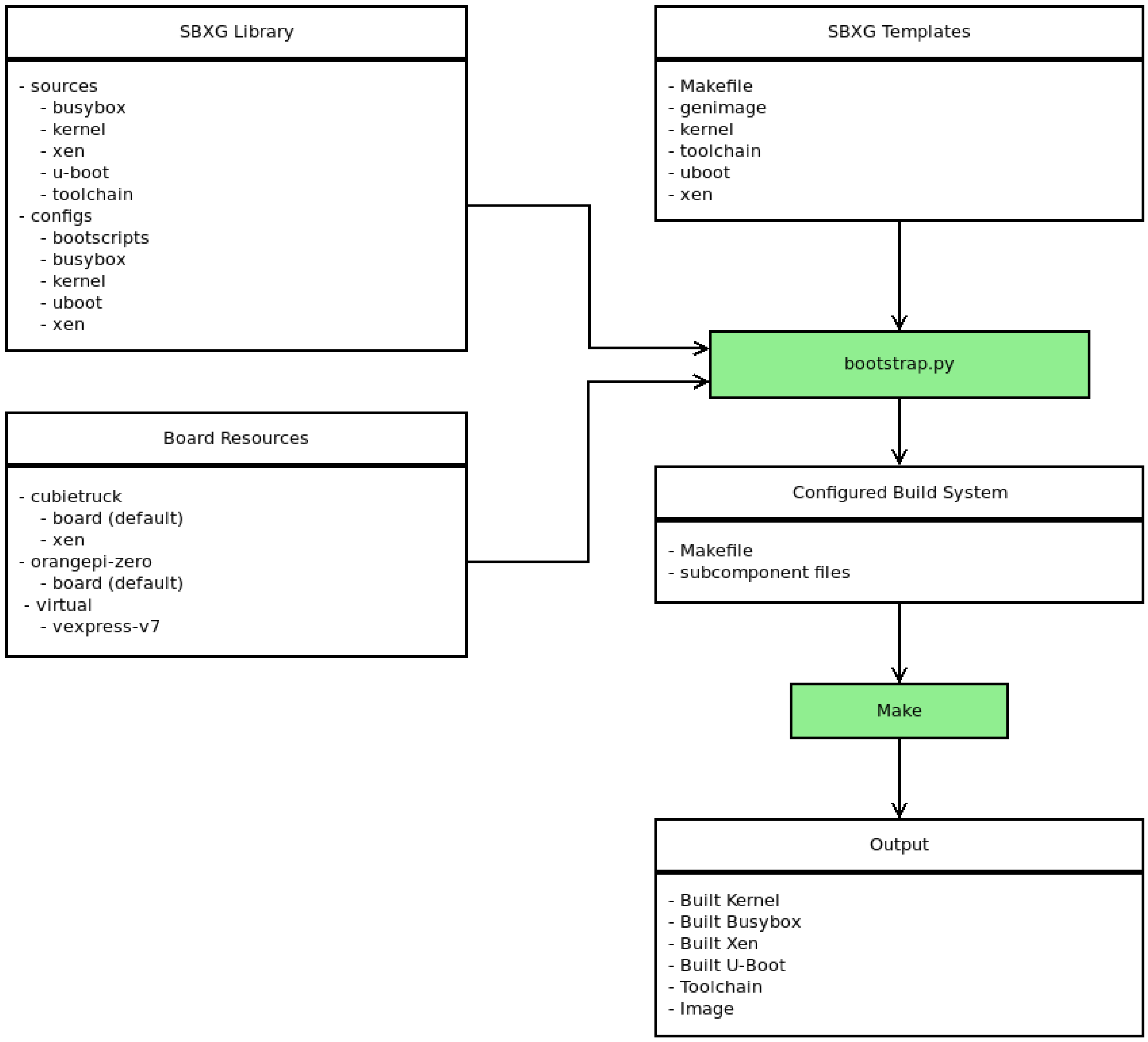
SBXG provides per GNU/Linux distribution scripts to install the necessary packages. They are contained within the `utils/` directory, in the top source directory. Run the script associated to your distribution.

CHAPTER 3

# Starting Up with SBXG

SBXG will retrieve and build bootloaders and kernels (e.g. U-Boot, Linux, Xen), but does not create nor configure a root file system! This task is up to you, or dedicated tools that you chose to use. However, SBXG proposes a “toy” script that creates a minimal Debian root file system from scratch so you can use SBXG, even when you don’t have a root file system.

SBXG proposes various built-in configurations. In function whether these offer support for Xen or not, different root file systems may be needed. This document proposes the command-line sequences to be paired with the default options proposed by SBXG. They all assume a POSIX shell is being used (e.g. bash), and are initiated from the top source directory of SBXG.



## 3.1 Cubietruck Default

```
mkdir build && cd build
../scripts/create-debootstrap.sh
../bootstrap.py --board cubietruck --toolchain armv7-eabihf
make
```

## 3.2 Cubietruck with Xen (1 guest)

```
mkdir build && cd build
sudo ../scripts/create-debootstrap.sh -x
sudo ../scripts/create-debootstrap.sh -o guest0_rootfs.ext3
```

(continues on next page)

(continued from previous page)

```
../bootstrap.py --board cubietruck --board-variant xen --toolchain armv7-eabihf
make
```

# CHAPTER 4

# How Do I Do That?

## 4.1 I want to compile a single component

You first need to take a look at the files known to SBXG, by running the `bootstrap.py` script with the `--show-library` option:

```
$ ./bootstrap.py --show-library

List of available boards (with variants):
  - cubietruck ( xen )
  - virtual ( vexpress-v7 )
  - orangepi-zero

List of sources:
  - uboot: 2017.07
  - xen: 4.8.0
  - toolchain: local
  - toolchain: armv7-eabihf
  - kernel: linux-4.12.0
  - busybox: 1.27.1

List of configurations:
  - bootscript: boot-sunxi-default
  - bootscript: boot-sunxi-xen
  - uboot: 2017.07-minimal
  - xen: 4.8-sunxi
  - kernel: linux-4.12-sunxi
  - kernel: linux-4.12-sunxi-xen-dom0
  - kernel: linux-4.12-xen-domu
  - busybox: minimal
```

### 4.1.1 I want to compile just a kernel

From the list that is shown to you, you must pick:

- a kernel to be compiled (in the *List of sources*),
- a kernel configuration (in the *List of configurations*),
- a toolchain (in the *List of sources*).

Make sure that all parameters are coherent together. For instance, do not pick a Xen configuration for a Linux kernel, or a Linux 3.4 configuration when you are trying to build a Linux 4.14. Configurations are also linked to a given architecture or SoC (e.g. cubietruck/sunxi), so make sure the toolchain you select is coherent with the product you want to build.

For instance, if you want to cross-build a Linux 4.12.0 for a Cubietruck (sunxi):

```
bootstrap.py --kernel linux-4.12.0 linux-4.12-sunxi
             --toolchain armv7-eabihf
```

### 4.1.2 I want to compile just a bootloader

From the list that is shown to you, you must pick:

- a U-Boot to be compiled (in the *List of sources*),
- a U-Boot configuration (in the *List of configurations*),
- a toolchain (in the *List of sources*).

For instance, if you want to build a U-Boot 2017.07 locally (assuming an ARM host):

```
bootstrap.py --uboot 2017.07 2017.07-minimal
             --toolchain local
```

### 4.1.3 I want to compile just Xen

From the list that is shown to you, you must pick:

- a Xen to be compiled (in the *List of sources*),
- a Xen configuration (in the *List of configurations*),
- a toolchain (in the *List of sources*).

For instance, if you want to cross-build a Xen 4.8.0 for a sunxi SoC:

```
bootstrap.py --xen 4.8.0 4.8-sunxi
             --toolchain armv7-eabihf
```

## 4.2 I want to generate a firmware image

TODO :/

# CHAPTER 5

# SBXG Configuration

SBXG relies on two search paths that provide its configuration:

- the `board` search path and
- the `lib` search path.

These two concepts will be explained in further details in the following sections. If no search path is specified, SBXG will assume the directories `board/` and `lib/` in the source source directory of SBXG.

## 5.1 Search Path

SBXG's configuration consist in a collection of structured files. These structures reside in entries called the *search paths*.

If one needs to develop its own configuration, and wish to make it private (outside of SBXG), it shall replicate the file hierarchy described in the following sections, and set the search paths to the directorys containing this new hierarchy.

The first *search path* is the **library**. It contains configurations files that allow to retrieve and compile the various components that SBXG supports.

The second *search path* consists of **boards** configurations. These are files that describe how several components shall be aggregate together to generate a single firmware image. If you want to only build components without creating a firmware image, you do not need this.

You can call the `--show-lib` option of the `bootstrap.py` script to print the files that SBXG will look for. For example, from SBXG top source directory:

```
$ ./boostrap.py --show-lib
List of available boards (with variants):
  - cubietruck ( xen )
  - virtual ( vexpress-v7 )
  - orangepi-zero

List of sources:
```

(continues on next page)

(continued from previous page)

```
  - uboot: 2017.07
  - xen: 4.8.2
  - toolchain: local
  - toolchain: armv7-eabihf
  - kernel: linux-4.14.8
  - kernel: linux-4.14.6
  - kernel: linux-4.14.17
  - kernel: linux-4.12.0
  - busybox: 1.27.1

List of configurations:
  - bootscript: boot-sunxi-default
  - bootscript: boot-sunxi-xen
  - uboot: 2017.07-minimal
  - xen: 4.8-sunxi
  - kernel: linux-4.12-sunxi
  - kernel: linux-4.12-sunxi-xen-dom0
  - kernel: linux-4.14-sunxi-xen-dom0
  - kernel: linux-4.12-xen-domu
  - kernel: linux-4.14-xen-domu
  - busybox: minimal
```

When providing configuration or source files to SBXG, you will need to pass one of these files.

## 5.2 SBXG's Board Directory

First, let's start with an example:

```
boards/
├── cubietruck
│   ├── board.yml
│   ├── images
│   │   └── default.j2
│   └── xen.yml
├── orangepi-zero
│   ├── board.yml
│   └── images
│       └── default.j2
└── virtual
    ├── images
    │   └── guest.j2
    └── vexpress-v7.yml
```

Each subdirectory in `boards/` (which is the default directory searched by SBXG) holds the configuration files for a given board. In our example, we have three supported boards:

- Cubietruck,
- OrangePiZero,
- virtual (as a based to build virtual machines).

Within each of these directories `board.yml` is the default configuration file that describes how different components are aggregated together. You may want to have several configurations. These are called *variants* in SBXG's terminology. An example is given by `cubietruck/xen.yml`, which is an alternative configuration to `cubietruck/board.yml`. Notice the directories `images/`. They contain genimage configuration and describe the layout of the firmware image.

## 5.3 SBXG's Library Directory

First, let's start with an example:

```
lib
├── configs
│   ├── bootscripts
│   │   ├── boot-sunxi-default.j2
│   │   └── boot-sunxi-xen.j2
│   ├── busybox
│   │   └── minimal
│   ├── kernel
│   │   ├── linux-4.12-sunxi
│   │   ├── linux-4.12-sunxi-xen-dom0
│   │   ├── linux-4.12-xen-domu
│   │   ├── linux-4.14-sunxi-xen-dom0
│   │   └── linux-4.14-xen-domu
│   ├── uboot
│   │   └── 2017.07-minimal
│   └── xen
│       └── 4.8-sunxi
└── sources
    ├── busybox
    │   └── 1.27.1.yml
    ├── kernel
    │   ├── linux-4.12.0.yml
    │   ├── linux-4.14.17.yml
    │   ├── linux-4.14.6.yml
    │   └── linux-4.14.8.yml
    ├── toolchain
    │   ├── armv7-eabihf.yml
    │   └── local.yml
    ├── uboot
    │   └── 2017.07.yml
    └── xen
        └── 4.8.2.yml
```

There are two directories within the library search path:

- `sources/`: where configurations to fetch components reside: * `busybox/`: to retrieve Busybox * `kernel/`: to retrive the principal kernel (e.g. Linux base or Xen Dom 0), * `toolchain/`: to retrive the compilation toolchain, * `uboot/`: to retrive the boot loader, * `xen/`: to retrieve the Xen ARM hypervisor.
- `configs/`: where configurations to compile components reside: * `bootscripts/`: available boot scripts , * `busybox/`: per-Busybox version configurations, * `kernel/`: per-Linux version configurations, * `u-boot/`: per-U-boot version configurations, * `xen/`: per-Xen version configurations.