Juergen Beisert

# GeoServ Quickstart

For BSP revision 2

$Rev$ : 861 $Date$ : $2008-04-2322:05:55+0200(Mi, 23Apr2008)$

# Contents

# Part I

# Quickstart for Axus GeoServ

# 1 Some notes about GeoServ

The compiler mentioned in chapter 3.3 is not part of the OSELAS-Toolchain package (but you will need this package anyway).
Instead you will find it in this project. So when running the `ptxdist select` command you should use the `i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig` file from this project instead.

## 1.1 Special Things

### 1.1.1 No BIOS

This BSP contains a BIOS replacement

### 1.1.2 Make it Boot

As there is no standard BIOS, there is no need for a special preparation to make it boot. The coreboot boot code contains the FILO loader, that is able to load any file from (mostly) any source. In my case it loads the kernel from a harddisk connected to the IDE port of this system.

Forget anything you know about Master Boot Record and how to make a harddisk to boot anything. You don't need it anymore.
Just copy the kernel and the `menu.lst` to your harddisk and your are done.

All right, a joke. Its not so easy. Some things must be considered to make it work.

Where is FILO searching for its menu file?

This is defined in its config file while building. Take a look into its `defconfig` file and search for this line:

```
MENULST_FILE = "hda1:/boot/filo/menu.lst"
```

This line means FILO expects the `menu.lst` file on the first harddisk (**hda**1) and in its first partition (hda**1**).

Within this partition the path `boot/filo` must exist, and in this directory FILO expects the `menu.lst` file.

To make the system boot, everything is prepared in this BSP. But most users want to setup their system by their own. So here a list of files, you must touch, if your system should work in a different way:

**projectroot/etc/fstab** to be touched to support another partition layout

**projectroot/menu.lst** to be touched to support different boot scenarios

**build-target/filo-0.5-r45/defconfig** to be touched if you want to use other filesystem than `ext2`

Note: To make any changes in `build-target/filo-0.5-r45/defconfig` valid do the following steps:

1. run `ptxdist clean filo`
2. run `ptxdist extract filo`
3. modify `build-target/filo-0.5-r45/defconfig`

4. run `ptxdist go`

The last step will build FILO and coreboot and creates a new boot image in `images/tc320_coreboot-v2.rom`.

To make persistent changes in this file, you should use ptxdist's patch feature:

1. run `ptxdist clean filo`
2. run `ptxdist extract filo`
3. change to directory `build-target/filo-0.5-r45`
4. run `quilt new my_modifications.diff`
5. run `quilt edit defconfg` and do your modifications
6. run `quilt refresh`
7. run `ptxdist go`
8. change back to the main BSP directory and run `ptxdist go`

Note: You need `quilt` to be installed on your host system to make it work in this way.

The command `quilt refresh` will store your changes into the `patches/filo-0.5-r45/generic` directory. Whenever you rebuild everything or at least FILO/coreboot, `ptxdist` will ensure to patch `build-target/filo-0.5-r45/defconfig` in the same way you did it and build the exact binary as it did the last time.

## 1.2 Prepare the System

After the BSP was built, we can now setup the system to run. Two things are to be done:

- bring in coreboot to boot the system
- prepare the harddisk to contain the system

### 1.2.1 Bring in coreboot

An external flash programmer will be required.

### 1.2.2 Prepare the Harddisk

Preparing the harddisk cannot be done in the target system. I'm using a special USB to IDE adpater to prepare my harddisks.

After connecting it to your development host you should create the partitions you want or need in your system. To work with the result of this BSP you must create:

1. One big partition at the begin of the harddisk with ID 83 (should be later on `/dev/hda1`).
2. A second partition usesd as a swap partition with ID 82 (should be later on `/dev/hda2`). I use twice the size of target's memory for this swap partition.

Next you must format the two partitions:

- Format the first partition with the `ext2` fileformat.
- Format the second partition with the `mkswap` command.

Now mount the first partition into your host to bring in the data.

1. run a `ptxdist images` command in your BSP (not as user `root`!)
2. Change to the mountpoint and run the `tar xf <where/your/BSP/is>/images/root.tgz`. Note: You must be user `root` to to this to ensure a correct result.

Leave the mountpoint, unmount the disk, disconnect it and assemble it into your target system. Connect a serial line to your target and your host and run a terminal program on this connection. Switch your target's power on and enjoy...

Note: You won't need a password for your first login as root. Just enter `root` as the user at the login prompt. You should define a root password after your first logon.

# 2 Some hints on using GeoServ

## 2.1 About the Project

The goal of this project was to get a file server with a few other services in my small private network.

## 2.2 About the GeodeGX1 Processor

The GeodeGX1 processor is a Pentium MMX derivate. Its an old device and you cannot buy it anymore. But it was very successfull, built into many applications like terminals and other embedded systems.

Its most known feature is the extensive use of the System Management Mode to emulate hardware that doesn't exist on this platform. A GeodeGX1 based system is like a standard PC, but not really. It support VGA graphics and sound. But this hardware is not compatible with other VGA and sound hardware.

For example the VGA hardware cannot handle text mode. It only supports graphic modes. Thanks to the SMM it emulates text mode transparent to the software. Also the sound hardware has its own implementation. To avoid adapting every software the SMM emulates a soundblaster device. So its possible to reuse the old drivers for the soundblaster device.

But this extensive use of the System Management Mode was also the reason why many people damm it. Emulation is very slow. If this processor was used in a real time environment (Yes! Many people did so!) the SMM was always one reason to become desperate. The serial console does not work correctly and looses chars even on 9600Bd if the text screen scrolls one line! The SMM has the highest priority in this system, so every real time OS looses control.

But there are also advantages of this processor: Its low power consumption. Everytime the CPU runs into a HALT instruction it stops the clock. If the CPU is only frequently used the device becomes warm, not hot! This feature is very nice in combination with dynamic ticks, so the clock stops as long as ever possible.

## 2.3 About coreboot

To get rid of the awful SMM I replaced the standard BIOS with coreboot. coreboot is mostly a hardware configurator and initialiser. After this is done, it starts a so called payload. The payload does the rest to bring up the system as you like. There are different payloads available. It depends on the requirements what payload is the right one. I need support to boot from network or from local harddisk. So my payload is Etherboot and FILO.

The disadvantage of coreboot and its lack of SMM support is, for each hardware feature you will use you need a driver! You cannot use a Sound-Blaster driver, if there is no Sound-Blaster emulation present! But with a specific driver you will win performance. For example on a 233MHz Geode with Sound-Blaster emulation to play internet radio it consumes about 50% of the CPU. With a native sound driver to play an MPG3 audio it only consumes 5% of the CPU!

Also without the SMM you cannot use the standard X11 graphic driver. It expects the SMM and depends on it. I have no clue why this hardware specific driver uses the emulated registers (for example to save and restore the VGA registers). But with LinuxBIOS in the background it fails.

## 2.4 About the Hardware

The basic hardware was a Windows CE based graphical terminal from AXUS type TC320. Shipped with all interfaces someone needs, it does also a nice work as a low power server. Beside the external interfaces like USB, PS/2 components, VGA, sound and RJ45 network it also supports internally a 2.5inch harddrive connection. Thats all I need (the VGA, sound and PS/2 I don't need in this application).

This hardware comes with a 16MiB DiscOnChip device. It was used to hold the whole Windows CE system. Currently I cannot use this device as a boot device, due to FILO does not know it. Maybe someday I will find the time to add this feature to FILO.

## 2.5 Services

This sections describes the service that should run on this machine.

### 2.5.1 Webservice

This machine should provide a small intranet web service. It will be built by ptxdist.

### 2.5.2 Remote Sync Service

To update the internal websites and to backup data a rsync service should run on this machine. It will be built by ptxdist.

### 2.5.3 Network File System Service

To spread user's homes around all machines, they are hosted on this machine. To export them to all other hosts one folder on the local hardddisk should be exported via NFS.

### 2.5.4 Samba Network Service

To share some files also with Windows based hosts, a Samba service should run on this machine. It will be built by ptxdist.

### 2.5.5 Concurrent Versions System Service

To do some software development a revision control system should run on a central point. It will be built by ptxdist.

### 2.5.6 Disabling Console Output During Kernel Startup

Console output during kernel startup consumes a lot of time. Most of the information we will see at this point of time are more useful when we are developing our system. If our system runs in production it's more useful to save time when booting. If we add the kernel parameter `quiet` this will suppress printk messages. Note that printk messages are still buffered in the kernel and can be retrieved after booting using the `dmesg` command.

When the `init` process starts the console is activated again.

# 3 Getting a working Environment

## 3.1 Download Parts

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaulation board package, or by download from a world wide web site.

The central place for OSELAS related documentation is http://www.oselas.com. This website provides all required packages and documentation (at least for software components which are available to the public).

To build OSELAS.BSP-JB-GeoServ-2, the following archives should be available on the development host:

- ptxdist-1.0.2.tgz
- ptxdist-1.0.2-patches.tgz
- OSELAS.BSP-JB-GeoServ-2.tar.gz
- OSELAS.Toolchain-1.1.1.tar.bz2

## 3.2 PTXdist Installation

PTXdist is shipped divided into several archives. This chapter provides information about how to install and configure PTXdist on the development host to get a working environment to build root filesystems for target systems.

### 3.2.1 Main parts of PTXdist

The main tool of the OSELAS.BoardSupport() Package is PTXdist. So before starting any work we'll have to install PTXdist on the development host. PTXdist consists of the following parts:

**The `ptxdist` Program:** `ptxdist` is installed on the development host during the installation process. `ptxdist` is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the `ptxdist` program is used in a *workspace* directory, which contains all project relevant files.

**A Configuration System:** The config system is used to customize a *configuration*, which contains information about which packages have to be built and which options are selected.

**Patches:** Due to the fact that some upstream packages are not bug free – especially with regard to cross compilation – it is often necessary to patch the original software. PTXdist contains a mechanism to automatically apply patches to packages. The patches are bundled into a separate archive. Nevertheless, they are necessary to build a working system.

**Package Descriptions:** For each software component there is a "recipe" file, specifying which actions have to be done to prepare and compile the software. Additionally, packages contain their configuration sniplet for the config system.

**Toolchains:** PTXdist does not come with a pre-built binary toolchain. Nevertheless, PTXdist itself is able to build toolchains, which are provided by the OSELAS.Toolchain() project. More in-deep information about the OSELAS.Toolchain() project can be found here: http://www.pengutronix.de/oselas/toolchain/index_en.html

**Board Support Package** This is an optional component, mostly shipped aside with a piece of hardware. There are various BSP available, some are generic, some are intended for a specific hardware.

### 3.2.2 Extracting the Sources

To install PTXdist, at least two archives have to be extracted:

**ptxdist-1.0.2.tgz** The PTXdist software itself.

**ptxdist-1.0.2-patches.tgz** All patches against upstream software packets (known as the 'patch repository').

**ptxdist-1.0.0-projects.tgz** Generic projects (optional), can be used as a starting point for self-built projects.

The PTXdist and patches packets have to be extracted into some temporary directory in order to be built before the installation, for example the `local/` directory in the user's home. If this directory does not exist, we have to create it and change into it:

```
~$ cd
~$ mkdir local
~$ cd local
```

Next steps are to extract the archives:

```
~/local$ tar -zxf ptxdist-1.0.2.tgz
~/local$ tar -zxf ptxdist-1.0.2-patches.tgz
```

and if required the generic projects:

```
~/local$ tar -zxf ptxdist1.0.0-projects.tgz
```

This PTXdist-1.0.2 release is one of the stable series. So Pengutronix didn't create new project releases for it. Instead the project archive of the PTXdist-1.0.0 release can be used. But this archive extracts into `ptxdist-1.0.0` instead of `ptxdist-1.0.2`. To use it anyway we must move its content manually:

```
~/local$ mv ptxdist-1.0.0/projects ptxdist-1.0.2
```

If everything goes well, we now have a PTXdist-1.0.2 directory, so we can change into it:

```
~/local$ cd ptxdist-1.0.2
~/local/ptxdist-1.0.2$ ls -l

total 472
drwxr-xr-x   3 jb users   4096 2007-06-19 16:52 autoconf/
-rwxr-xr-x   1 jb users     28 2007-06-19 16:52 autogen.sh*
drwxr-xr-x   2 jb users   4096 2008-04-18 15:09 autom4te.cache/
drwxr-xr-x   3 jb users   4096 2008-04-15 14:44 bin/
-rw-r--r--   1 jb users 124874 2008-04-18 12:33 ChangeLog
drwxr-xr-x   9 jb users   4096 2007-06-19 16:52 config/
-rwxr-xr-x   1 jb users 192930 2008-04-18 15:09 configure*
-rw-r--r--   1 jb users   9775 2008-04-18 13:53 configure.ac
-rw-r--r--   1 jb users  18361 2007-06-19 16:52 COPYING
-rw-r--r--   1 jb users   2848 2008-04-07 13:08 CREDITS
drwxr-xr-x   3 jb users   4096 2007-06-19 16:47 debian/
drwxr-xr-x   3 jb users   4096 2008-04-17 12:51 Documentation/
drwxr-xr-x   8 jb users   4096 2007-06-19 16:52 generic/
-rw-r--r--   1 jb users    193 2008-04-17 13:07 INSTALL
-rw-r--r--   1 jb users   2686 2007-06-19 16:52 Makefile.in
drwxr-xr-x 134 jb users   4096 2008-04-07 13:08 patches/
-rw-r--r--   1 jb users   3985 2008-04-17 12:46 README
-rw-r--r--   1 jb users    691 2007-06-19 16:52 REVISION_POLICY
drwxr-xr-x   7 jb users  24576 2008-04-18 15:08 rules/
drwxr-xr-x   7 jb users   4096 2008-04-18 15:08 scripts/
drwxr-xr-x   3 jb users   4096 2007-06-19 16:51 tests/
-rw-r--r--   1 jb users  28512 2008-04-17 12:44 TODO
```

### 3.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs are installed on the development host. The configure script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
~/local/ptxdist-1.0.2$ ./configure
```

This will check your system for required components PTXdist relies on. If all required components are found the output ends with:

```
[...]
configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in

ptxdist version 1.0.2 configured.
Using '/usr/local' for installation prefix.

Report bugs to ptxdist@pengutronix.de

~/local/ptxdist-1.0.2$
```

Without further arguments PTXdist is configured to be installed into `/usr/local`, which is the standard location for user installed programs. To change the installation path to anything non-standard, we use the `--prefix` argument to the `configure` script. The `--help` option offers more information about what else can be changed for the installation process.

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it.

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing. If this happens, the missing tools have to be installed from the distribution befor re-running the `configure` script.

In this early PTXdist version not all tests are implemented in the `configure` script yet. So if something goes wrong or you don't understand some error messages send a mail to `support@pengutronix.de` and help us improve the tool.

When the `configure` script is finished successfully, we can now run

```
~/local/ptxdist-1.0.2$ make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into it's final location. In order to write to `/usr/local`, this step has to be performed as root:

```
~/local/ptxdist-1.0.2$ su
[enter root password]
/home/username/local/ptxdist-1.0.2$ make install
[...]
```

If we don't have root access to the machine it is also possible to install into some other directory with the `--prefix` option. We need to take care that the `bin/` directory below the new installation dir is added to our `$PATH` environment variable (for example by exporting it in `~/.bashrc`).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-1.0.2$ cd
~$ rm -fr local/ptxdist-1.0.2
```

### 3.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
~$ ptxdist setup
```

Due to PTXdist is working with sources only, it needs various source archives from the world wide web. If these archives are not present on our host, PTXdist starts the `wget` command to download them on demand.

#### 3.2.4.1 Proxy Setup

To do so, an internet access is required. If this access is managed by a proxy `wget` command must be adviced to use it. PTXdist can be configured to advice the `wget` command automatically: Navigate to entry *Proxies* and enter the required addresses and ports to access the proxy in the form:

```
<protocol>://<address>:<port>
```

#### 3.2.4.2 Source Archive Location

Whenever PTXdist downloads source archives it stores it project locally. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry *Source Directory* and enter the path to the directory where PTXdist should store archives to share between projects.

#### 3.2.4.3 Generic Project Location

If we already installed the generic projects we should also configure PTXdist to know this location. If we already did so, we can use the command `ptxdist projects` to get a list of available projects and `ptxdist clone` to get a local working copy of a shared generic project.

Navigate to menu entry *Project Searchpath* and enter the path to projects that can be used in such a way. Here we can configure more than one path, each part can be delemited by a colon. For example for PTXdist's generic projects and our own previous projects like this:

```
/usr/local/lib/ptxdist-1.0.2/projects:/office/my_projects/ptxdist
```

Leave the menu and store the configuration. PTXdist is now ready for use.

## 3.3 Toolchains

### 3.3.1 Abstract

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the *GNU Compiler Collection*, `gcc`. The gcc packet includes the compiler frontend, `gcc`, plus several backend tools (cc1, g++, ld etc.) which actually perform the different stages of the compile process. `gcc` does not contain the assembler, so we also need the *GNU Binutils package* which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the *GNU target*. For example, the cross compilers for ARM and powerpc may look like

- `arm-softfloat-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
~$ gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
~$ arm-softfloat-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the "toolchain" is the runtime library (libc, dynamic linker). All programs running on the embedded system are linked against the libc, which also offers the interface from user space functions to the kernel.

The compiler and libc are very tightly coupled components: the second stage compiler, which is used to build normal user space code, is being built against the libc itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime libc is identical with the libc the compiler was built against.

PTXdist doesn't contain a pre-built binary toolchain. Remember that it's not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

### 3.3.2 Using Existing Toolchains

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.

The OSELAS.BoardSupport() Packages shipped for PTXdist have been tested with the OSELAS.Toolchains() built with the same PTXdist version. So if an external toolchain is being used which isn't known to be stable, a target may fail. Note that not all compiler versions and combinations work properly in a cross environment.

Every OSELAS.BoardSupport() Package checks for its OSELAS.Toolchain it's tested against, so using a different toolchain vendor requires an additional step:

Open the OSELAS.BoardSupport() Package menu with:

```
~$ ptxdist menuconfig
```

and navigate to `PTXdist Config, Architecture` and `Check for specific toolchain vendor`. Clear this entry to disable the toolchain vendor check.

### 3.3.3 Building a Toolchain

PTXdist-1.0.2 handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the OSELAS.BoardSupport() Package.

A PTXdist project generally allows to build into some project defined directory; all OSELAS.Toolchain projects that come with PTXdist are configured to use the standard installation paths mentioned below.

All OSELAS.Toolchain projects install their result into `/opt/OSELAS.Toolchain-1.1.1/`.

Usually the `/opt` directory is not world writable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions so that the user can write (`mkdir /opt/OSELAS.Toolchain-1.1.1 ; chown <username> /opt/OSELAS.Toolchain-1.1.1; chmod a+rwx /opt/OSELAS.Toolchain-1.1.1`).

#### 3.3.3.1 Building the OSELAS.Toolchain for OSELAS.BSP-JB-GeoServ-2

To compile and install an OSELAS.Toolchain we have to extract the OSELAS.Toolchain archive, change into the new folder, configure the compiler in question and start the build.

The required compiler to build the OSELAS.BSP-JB-GeoServ-2 board support package is

`i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18`.

So the steps to build this toolchain are:

```
~$ tar xf OSELAS.Toolchain-1.1.1.tar.bz2
~$ cd OSELAS.Toolchain-1.1.1
~/OSELAS.Toolchain-1.1.1$ ptxdist select
  ptxconfigs/i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig
~/OSELAS.Toolchain-1.1.1$ ptxdist go
```

At this stage we have to go to our boss and tell him that it's probably time to go home for the day. Even on reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

Measured times on different machines:

- Single Pentium 2.5 GHz, 2 GiB RAM: about 2 hours
- Dual Athlon 2.1 GHz, 2 GiB RAM: about 1 hour 20 minutes
- Dual Quad-Core-Pentium 1.8 GHz, 8 GiB RAM: about 25 minutes

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

### 3.3.4 Freezing the Toolchain

As we build and install this toolchain with regular user rights we should modify the permissions as a last step to avoid any later manipulation. To do so we could set all toolchain files to read only or changing recursivley the owner of the whole installation to user root.

This is an important step for reliability. Do not omit it!

#### 3.3.4.1 Building additional Toolchains

The OSELAS.Toolchain-1.1.1 bundle comes with various predefined toolchains. Refer the `ptxconfigs/` folder for other definitions. To build additional toolchains we only have to clean our current toolchain projekt, removing the current `ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.1.1$ ptxdist clean
~/OSELAS.Toolchain-1.1.1$ rm ptxconfig
~/OSELAS.Toolchain-1.1.1$ ptxdist select
       ptxconfigs/any_another_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.1.1$ ptxdist go
```

All toolchains will be installed side by side architecture dependend into directory

`/opt/OSELAS.Toolchain-1.1.1/architecture_part`.

Different toolchains for the same architecture will be installed side by side version dependend into directory

`/opt/OSELAS.Toolchain-1.1.1/architecture_part/version_part`.

# 4 Building Description

## 4.1 Prepare and Build

This chapter assumes we installed ptxdist and the toolchain in a way the last chapter describes it.

It also assumes we have a big wire into the internet, as all the sources are downloaded from there. If we setup ptxdist to use a generic source archive directory the download only happens once.

Each ptxdist project uses a toolchain. This is the first thing we must setup prior starting the build.

```
jb@jupiter:~/ $ cd OSELAS.BSP-JB-GeoServ-2
jb@jupiter:~/OSELAS.BSP-JB-GeoServ-2 $ ptxdist toolchain
            /opt/OSELAS-1.1.1/i586-pmmx-linux-gnu/bin
```

Now everything is ready to build the project:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoServ-2 $ ptxdist go
```

This will take a while.

Sometimes downloading a source archive may fail. Most of the time the maintainer moves the archive to another location or some delete the older releases when a new version was released. A simple solution is to search for this archive in the web, download it manually and store it into the generic source archive directory. If the archive file is already present, no download will happen.

After building the project, we can find everything we need in the `root/` directory. This directory is intended to be used as an NFS root filesystem. To do so, we must fill the `root/dev/` directory with at least three device nodes: `console`, `null` and `zero`. ptxdist rejects to work as root, so we must create these nodes manually. We could do so with:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoServ-2 $ sudo cp -a /dev/console /dev/null
            /dev/zero root/dev
```

[*sound of entering root password*]

Done. Now we could boot this filesystem as our TC320's root filesystem.

But first we must replace the TC320's current BIOS by LinuxBIOS. I found no other way then using an external flash programmer to replace flash's content.

Refer `http://www.coreboot.org/AXUS_TC320_Build_Tutorial` how to get access to the flash device. The coreboot file to flash we can found in `images/linuxbios.rom`.