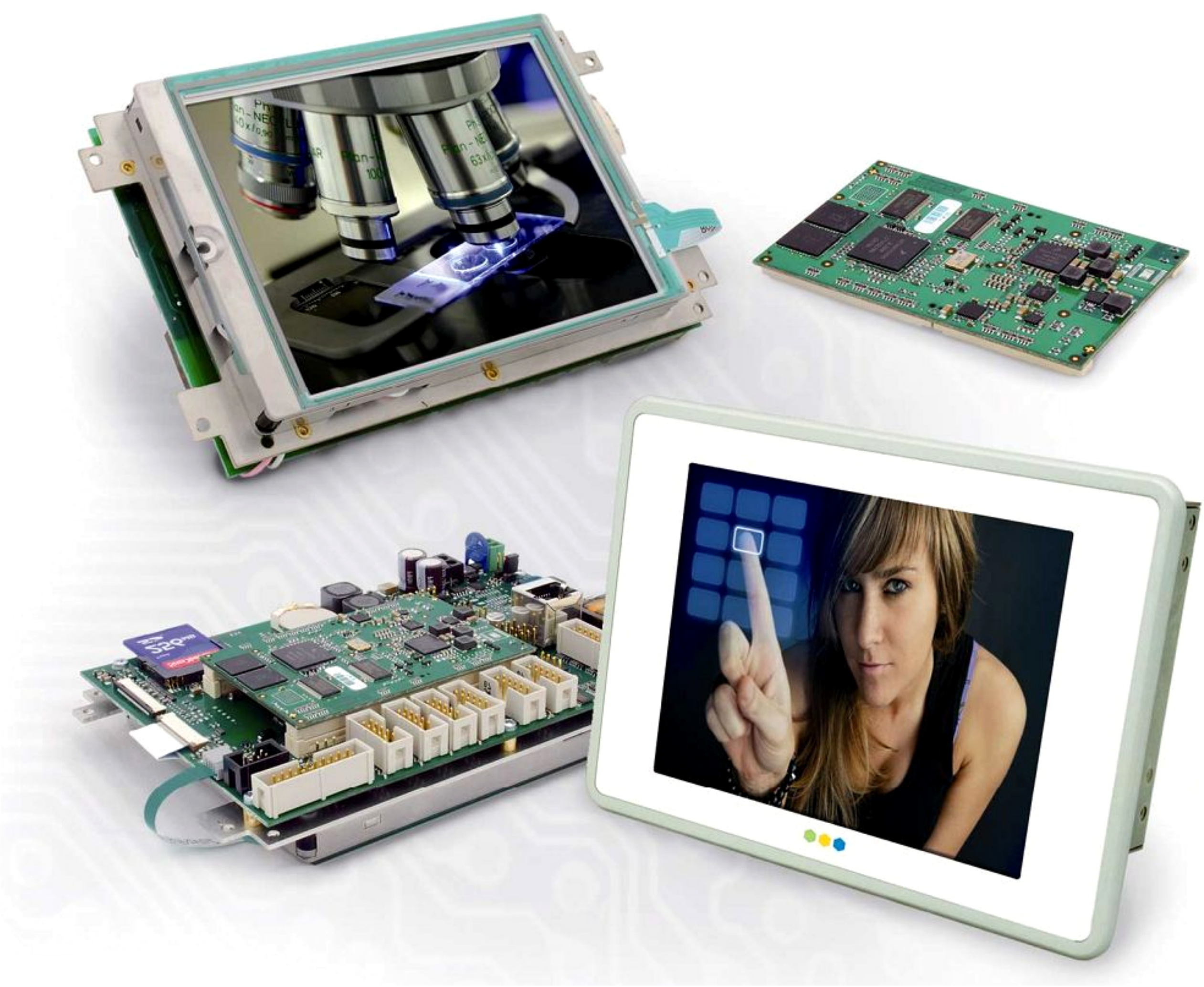
Garz & Fricke

Embedded Systems

Garz & Fricke

Embedded Computer Systems



Linux · User Manual

SANTARO-1.44.4-0

freescale Enabled

Windows® Embedded Silver Partner

Zuverlässige
Qualität
Made in Germany

Important hints

Thank you very much for purchasing a Garz & Fricke product. Our products are dedicated to professional use and therefore we suppose extended technical knowledge and practice in working with such products.

The information in this manual is subject to technical changes, particularly as a result of continuous product upgrades. Thus this manual only reflects the technical status of the products at the time of printing. Before design-in the device into your or your customer's product, please verify that this document and the therein described specification is the latest revision and matches to the PCB version. We highly recommend contacting our technical sales team priorto any activity of that kind. A good way getting the latest information is to check the release notes of each product and/or service. Please refer to the chapter [▶ 10 Related documents and online support].

The attached documentation does not entail any guarantee on the part of Garz & Fricke GmbH with respect to technical processes described in the manual or any product characteristics set out in the manual. We do not accept any liability for any printing errors or other inaccuracies in the manual unless it can be proven that we are aware of such errors or inaccuracies or that we are unaware of these as a result of gross negligence and Garz & Fricke has failed to eliminate these errors or inaccuracies for this reason. Garz & Fricke GmbH expressly informs that this manual only contains a general description of technical processes and instructions which may not be applicable in every individual case. In cases of doubt, please contact our technical sales team.

In no event, Garz & Fricke is liable for any direct, indirect, special, incidental or consequential damages arising out of use or resulting from non-compliancy of therein conditions and precautions, even if advised of the possibility of such damages.

Before using a device covered by this document, please carefully read the related hardware manual and the quick guide, which contain important instructions and hints for connectors and setup.

Embedded systems are complex and sensitive electronic products. Please act carefully and ensure that only qualified personnel will handle and use the device at the stage of development. In the event of damage to the device caused by failure to observe the hints in this manual and on the device (especially the safety instructions), Garz & Fricke shall not be required to honour the warranty even during the warranty period and shall be exempted from the statutory accident liability obligation. Attempting to repair or modify the product also voids all warranty claims.

Before contacting the Garz & Fricke support team, please try to help yourself by the means of this manual or any other documentation provided by Garz & Fricke or the related websites. If this does not help at all, please feel free to contact us or our partners as listed below. Our technicians and engineers will be glad to support you. Please note that beyond the support hours included in the Starter Kit, various support packages are available. To keep the pure product cost at a reasonable level, we have to charge support and consulting services per effort.

Shipping address:
Garz & Fricke GmbH
Tempowerkring 2
21079 Hamburg
Germany

Support contact:

Phone	+49 (0) 40 / 791 899 - 30
Fax	+49 (0) 40 / 791 899 - 39
Email	▶ support@garz-fricke.com
URL	▶ www.garz-fricke.com

Contents

1 Introduction

Garz & Fricke systems based on **Freescale i.MX6** can be used with an adapted version of Linux, a royalty-free open-source operating system. The Linux kernel as provided by Garz & Fricke is based on extensions by Freescale that currently have not been contributed back into the mainline kernel. Furthermore, Garz & Fricke has made several modifications and extensions to the kernel which are currently not contributed back to the mainline kernel as well. Nevertheless, the full source code is available as a board support package (BSP) from Garz & Fricke.

A Garz & Fricke device normally comes with a pre-installed Garz & Fricke Linux operating system. However, since Linux is an open source system, the user is able to build the complete BSP from source, modify it according to his needs and replace the pre-installed Linux system with a custom one.

This manual contains information about the usage of the Garz & Fricke Linux operating system for **SANTARO-1.44.4-0**, as well as the build process of the Garz & Fricke Linux BSP and the integration of custom software components. The BSP can be downloaded from the Garz & Fricke support server:

- http://support.garz-fricke.com/projects/Santaro

It does not include the complete source code to all packages. Instead, several external packages are downloaded during the build process from the Garz & Fricke packages mirror:

- support.garz-fricke.com/mirror

Modifications to these packages are provided as source code patches, which are part of the BSP.

Please note that Linux development at Garz & Fricke is always in progress. Thus, there are new releases of the BSP at irregular intervals. Due to differences between the various Linux BSP platforms and versions, a separate manual is available for every platform/version combination above version 1.29.0. To avoid confusion, the version number of the manual exactly matches the BSP version number.

In addition to this manual, please also refer to the dedicated hardware manuals which can be found on the Garz & Fricke website as well.

2 Overview

A Garz & Fricke Linux System generally consists of four basic components:

- the bootloader
- the Linux kernel
- the root file system
- the device configuration

These software components are usually installed on separate partitions on the backing storage of the embedded system.

Newer Garz & Fricke devices are shipped with a separate small ramdisk-based Linux system called **Flash-N-Go System** which is installed in parallel to the main operating system. The purpose of Flash-N-Go is to provide the user a comfortable and secure update mechanism for the main operating system components.

2.1 The bootloader

There are several bootloaders available for the various Linux platforms in the big Linux world. For desktop PC Linux systems, GRUB or LILO are commonly used. Those bootloaders are started by hardwired PC-BIOS.

Embedded Systems do not have a PC-like BIOS. In most cases they are started from raw flash memory or an eMMC device. For this purpose, there are certain open source boot loaders available, like RedBoot, U-Boot or Barebox. Furthermore, Garz & Fricke provides its own bootloader called **Flash-N-Go Boot** for its newer platforms (e.g. SANTARO).

SANTARO uses the bootloader **Flash-N-Go Boot**.

2.2 The Linux kernel

The Linux OS kernel includes the micro kernel specific parts of the Linux OS and several internal device and subsystem drivers.

2.3 The root file system

The root file system is simply a file system. It contains the Linux file system hierarchy folders and files. Depending on the system configuration, the root file system may contain:

- system configuration files
- shared runtime libraries
- dynamic device and subsystem drivers - so called **loadable kernel modules** - in contrast to kernel-included device and subsystem drivers
- executable programs for system handling
- fonts
- etc.

Usually, a certain standard set of runtime libraries can be found in almost every Linux system, including standard C/C++ runtime libraries, math support libraries, threading support libraries, etc.

Embedded Linux systems principally differ in dealing with the graphical user interface (GUI). The following list gives some examples for GUI systems that are commonly used in embedded Linux systems:

- no GUI framework
- Qt Embedded on top of a Linux frame buffer device
- Qt Embedded on top of DirectFB graphics acceleration library
- Qt Embedded on top of an X-Server
- GTK+ on top of DirectFB graphics acceleration library
- GTK+ on top of a X-Server
- Nano-X / Microwindows on top of a Linux frame buffer device

Some system may additionally be equipped with a window manager of small footprint or a desktop system like KDE ore GNOME. However, in practice most embedded Linux Systems are running only one GUI application and a desktop system generates useless overhead.

SANTARO-0 is equiped with **Qt Embedded on top of a Linux frame buffer device**.

2.4 The device configuration

Most embedded bootloaders have to deal with some configuration setup parameters. These parameters are necessary for features that cannot be auto-detected by the OS (e.g. the display geometry).

The Flash-N-Go Boot bootloader shipped with Garz & Fricke systems manages these parameters in the form of an XML file called **config.xml**. This config.xml file is located on the second boot partition of the eMMC backing storage device. The bootloader itself does not use any XML parameters, because its design is aimed to be minimalistic without unnecessary hardware initialization. Instead, the bootloader loads the contents of the config.xml file into to a configurable location in RAM and passes its physical address offeset to the Linux kernel initialization.

Note: Flash-N-Go Boot is a propriatary Garz & Fricke bootloader that is only used on Garz & Fricke systems. Therefore, unmodified mainline kernels can't handle this parameter passing mechanism. Custom kernels have to be modified in the same way like those shipped by Garz & Fricke if Flash-N-Go Boot is used.

2.5 The partition layout

As already stated in chapter [▶ 2 Overview], the different components of the embedded Linux system are stored in different partitons of the backing-storage. The backing-storage type of SANTARO is eMMC. In addition to the partitions for the basic Linux components there may be some more partitions depending on the system configuration.

The partition layout for the SANTARO-0 platform is:

Partition	File System	Contents
mmcblk0boot0	none	bootloader image
mmcblk0boot1	FAT32	XML configuration parametes (config.xml) and touchscreen configuration (ts.conf)
mmcblk0p1	FAT32	Linux kernel image file (linuximage), bootloader command file (boot-alt.cfg) and Flash-N-Go ramdisk file (root.cpio.gz)
mmcblk0p2	FAT32	Linux kernel image file (linuximage), bootloader command file (boot.cfg) for the Garz & Fricke Linux system
mmcblk0p3	ext3	root file system

Flash-N-Go Boot can start the following Linux kernel image types:

- **zImage** compressed image
- **uImage** compressed image with u-boot header
- **Image** uncompressed image

2.6 Further information

For readers who are not familar with Linux in general, the following link may be helpful:

▶ http://tldp.org/LDP/intro-linux/html

Information regarding embedded Linux systems can be found in the following book:

- "Building Embedded Linux systems 2nd Edition", Karim Yaghmour, John Masters, Gilad Ben-Yossef, Philippe Gerum, O'Reilly, 2008, ISBN: 978-0-596-52968-0

Information regarding Linux infrastructure issues in general can be found at:

▶ http://tldp.org/LDP/Pocket-Linux-Guide/html
▶ http://www.linuxfromscratch.org

Information about Qt/Embedded can be found at:

▶ http://directfb.org

Information about the X window system can be found at:

- http://www.freedesktop.org

Information about Qt/Embedded can be found at:

- http://qt-project.org

Information about Nano-X / Microwindows can be found at:

- http://www.microwindows.org

Information about GTK+ can be found at:

- http://www.gtk.org

Information about U-Boot can be found at:

- http://www.denx.de/wiki/U-Boot

Information about the RedBoot can be found at:

- http://ecos.sourceware.org/docs-latest/redboot/redboot-guide.html

3 Accessing the target system

A Garz & Fricke hardware platform can be accessed from a host system using the following technologies:

- **Serial console** console access over RS-232
- **Telnet** console access over Ethernet
- **SSH** encrypted console access and file transfer over Ethernet
- **TFTP** file downoad over Ethernet
- **FTP** file upload and download over Ethernet

Each of the following chapters describes one of these possibilities and, where applicable, gives a short example of how to use it. For all examples, the Garz & Fricke target system is assumed to have the IP address **192.168.1.1**.

3.1 Serial console

The easiest way to access the target is via the serial console. Simply connect the first RS-232 port on the device (see the according Hardware manual to determine the correct connector and pin layout) to your target system using a null-modem cable and set up your favourite terminal program (e.g. minicom) with the following settings:

- 115200 baud
- 8 data bits
- no parity
- 1 stop bit
- no hardware flow control
- no software flow control

From the very first moment when the target is powered, you should see debug messages in the terminal. After the boot process has finished, you will see the Linux login shell:

```
starting pid 638, tty '/dev/console': '/sbin/getty -L 115200 ttymxc0 vt100'
SANTARO login:
```

You can log in as user **root** without any password by default.

3.2 SSH console

Using SSH, you can access the console of the device and copy files to or from the target. Please note that SSH must be installed on the host system in order to gain access.

To login via SSH, type on the host system:

```
$ ssh root@192.168.1.1
```

The first time you access the target system from the host system, the target is added to the list of known hosts. You have to confirm this step in order to establish the connection.

```
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
RSA key fingerprint is e5:86:89:19:50:a5:46:52:15:35:e5:0e:d2:d1:f9:62.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.1' (RSA) to the list of known hosts.
root@SANTARO>:~
```

To return to your host system's console, type:

```
$ exit
```

You can use **secure copy (scp)** on the device or the host system to copy files from or to the device.

Example: To copy the file **myapp** from the host's current working directory to the target's **/usr/bin** directory, type on the host's console:

```
$ scp ./myapp root@192.168.1.1:/usr/bin/myapp
```

To copy the target's **/usr/bin/myapp** file back to the host's current working directory, type:

```
$ scp root@192.168.1.1:/usr/bin/myapp ./myapp
```

3.3 Telnet console

Telnet can be used to access the console. Please note that Telnet must be installed on the host system in order to gain access.

To login via Telnet, type on the host system:

```
$ telnet 192.168.1.1
```

The login prompt appears and you can login with username and password:

```
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
SANTARO login: root
Password: [Enter password]
root@SANTARO:~
```

3.4 Uploading files with TFTP

You can copy files from the host system to the target system using the target's TFTP client. Please note that a TFTP server has to be installed on the host system. Usually, a TFTP server can be installed on every Linux distribution. To install the TFTP server under Debian based systems with apt, the following command must be executed on the host system:

```
$ sudo apt-get install xinetd tftpd tftp
```

The TFTP server must be configured as follows in the **/etc/xinetd.d/tftpd** file on the host system in order to provide the directory **/srv/tftp** as TFTP directory:

```
service tftp
{
    protocol        = udp
    port            = 69
    socket_type     = dgram
    wait            = yes
    user            = nobody
    server          = /usr/sbin/in.tftpd
    server_args     = /srv/tftp
    disable         = no
}
```

The **/srv/tfp** directory must be created on the host system with the following commands:

```
$ sudo mkdir /srv/tftp
$ sudo chmod -R 777 /srv/tftp
$ sudo chown -R nobody /srv/tftp
```

After the above modification the xinetd must be restarted on the host system with the new TFTP service with the following command:

```
$ sudo service xinetd restart
```

From now on, you can access files in this directory from the target.

Example: Copying the file **myapp** from the host system to the target's **/usr/bin** directory. To achieve this, first copy the file **myapp** to your TFTP directory on the host system:

```
$ cp ./myapp /
```

The host system is assumed to have the ip address **192.168.1.100**. On the target system, type:

```
root@SANTARO:~ tftp -g 192.168.1.100 -r myapp -l /usr/bin/myapp
```

3.5 Uploading files with FTP

You can exchange files between the host system and the target system using an FTP client on the host system. Simply choose your favourite FTP client and connect to **ftp://192.168.1.1**.

4 Services and utilities

The Garz & Fricke Linux BSP includes several useful services for flexible application handling. Some of them are just run-once services directly after the OS has been started, others are available permanently.

4.1 Services

The services on Garz & Fricke Linux systems are usually started with start scripts. This is a very common technique on Linux systems. Garz & Fricke uses the **run-parts** tool for this purpose. The **run-parts** tool is triggered by the **busybox** init process. The sequence can be viewed in the file **/etc/init.d/rcS**:

```
[...]
echo "running rc.d services..."
run-parts -a start /etc/rc.d
[...]
```

The **run-parts** process executes all scripts (the links to scripts) in **/etc/rc.d** starting with the character **S**, passing the parameter **start** to the scripts. Furthermore, the naming convention states that the **S** character is followed by a number to determine the (numeric) execution order.

4.1.1 Udev

The **udev** service dynamically creates the device nodes in the **/dev** directory on system start up, as they are reported by the Linux kernel.

Furthermore, udev is user-configurable to react on asynchronous events from device drivers reported by the Linux kernel like hotplugging. The according rules are located in the root file system at **/lib/udev/rules.d**.

Additionally, udev is in charge of loading firware if a device driver requests it. Drivers that use the firmware subsystem have to place their firmware in the folder **/lib/firmware**.

The udev service has a startup link that points to the corresponding start script:

- **/etc/rc.d/S00udev -> /etc/init.d/udev**

Udev can be configured in **/etc/udev/udev.conf**.

More information about udev can be found at:

- https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html

More information about the firmware subsystem in the Linux kernel can be found in the BSP directory after building the SANTARO platform at:

- **platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/firmware_class/README**

The following driver in the Linux kernel can serve as example for the usage of the firmware subsystem in a kernel driver:

- **platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/input/touchscreen/atmel_mxt_ts.c**

4.1.2 Services only starting once after system installation

The **rc-once** service is responsible for starting tasks that have do be executed once after system istallation. It has a startup link that points to the corresponding start script:

- **/etc/rc.d/S01rc-once -> /etc/init.d/rc-once**

The main function of rc-once is to execute the script **/lib/init/rc-once.sh**. This scripts looks for further scripts in **/etc/rc.once.d**, executes them and marks the execution as done. On subsequent boots, these scripts will not be executed anymore.

Garz & Fricke systems use this mechanism to initially generate an SSH key. The corresponding script can be found in the target's root file system at **/etc/rc.once/openssh**.

4.1.3 D-Bus

The **dbus** service is a message bus system, a simple way for applications to communicate with each another. Additionally, D-Bus helps coordinating the process lifecycle: it makes it simple and reliable to code a **single instance** application or daemon, and to launch applications and daemons on demand when their services are needed.

Garz & Fricke systems are shipped with dbus bindings for glib and Qt. Therefore, the corresponding APIs can be used for application programming. Furthermore, Garz & Fricke systems are configured to support **HALD** that allows to detect hotplugging events in applications asynchronously. For Qt, Garz & Fricke provides an example in its BSP under **local_src/common/qt4-guf-dbus**.

The dbus service has a startup link that points to the corresponding start script:

- **/etc/rc.d/S12dbus -> /etc/init.d/dbus**

More information about dbus can be found at:

- http://www.freedesktop.org/wiki/Software/dbus

More information about the Qt dbus bindings can be found at:

- http://qt-project.org/doc/qt-4.7/intro-to-dbus.html

More information about the glib dbus bindings can be found at:

- http://dbus.freedesktop.org/doc/dbus-glib

4.1.4 Banner

The **banner** service is responsible for displaying the banner figlet on the serial console during the boot process, right before the login prompt. It has a startup link that points to the corresponding start script:

- **/etc/rc.d/S99banner -> /etc/init.d/banner**

The banner service uses the **/usr/bin/figlet** application to generate the figlet as a combination of the string **Garz+Fricke** and the hostname, which is determined by calling **/bin/hostname**.

4.1.5 System time initialization

Two services are involved in the system time initialization:

- **hwclock**
- **ntpclient**

The **hwclock** service has a startup link that points to the corresponding start script:

- **/etc/rc.d/S12hwclock -> /etc/init.d/hwclock**

It simply reads the RTC and sets the system time with **/sbin/hwclock**. The usage of this program is shown by executing:

```
root@SANTARO:~ /sbin/hwclock --help
BusyBox v1.18.5 (2013-04-26 15:13:42 CEST) multi-call binary.

Usage: hwclock [-r|--show] [-s|--hctosys] [-w|--systohc] [-l|--localtime] [-u|--utc]
    ↪ [-f FILE]

Query and set hardware clock (RTC)

Options:
        -r      Show hardware clock time
        -s      Set system time from hardware clock
        -w      Set hardware clock to system time
        -u      Hardware clock is in UTC
        -l      Hardware clock is in local time
        -f FILE Use specified device (e.g. /dev/rtc2)
```

The ntpclient service is triggered by the up codition of eth0 in

- /etc/network/interfaces

It starts /usr/sbin/ntpclient as a daemon. With the options set by Garz & Fricke per default, it reads the time from the server ntp.ubuntu.com (if reachable) and updates the system time every 10 minutes.

The usage of the ntpclient is shown by executing:

```
root@SANTARO:~ /usr/sbin/ntpclient
Usage: ntpclient [-c count] [-f frequency] [-g goodness] -h hostname
        [-i interval] [-l] [-p port] [-q min_delay] [-s] [-t]
```

4.1.6 SSH service

The ssh service allows the user to log in on the target system. Futhermore, the SFTP and SCP functionalities are activated to allow secure file transfers. The communication is encrypted.

The ssh service has a startup link that points to the corresponding start script:

- /etc/rc.d/S16openssh -> /etc/init.d/openssh

The startup script simply starts /usr/sbin/sshd as a daemon. The sshd configuration can be found in the target's root file system at /etc/ssh/sshd_config.

More information about OpenSSH can be found at:

▶ http://www.openssh.org

4.1.7 Telnet service

The telnet service allows the user to log in on the target system.

Note: Due to the fact that telnet does not use encryption, it is recommended to deactivate this service in final products in order to avoid security leaks.

The telnet service has a startup link that points to the corresponding start script:

- /etc/rc.d/ -> /etc/init.d/telnetd

The startup script simply starts /sbin/utelnetd as a daemon. The usage of telnetd is shown by executing:

```
root@SANTARO:~ telnetd --help
BusyBox v1.18.5 (2013-04-26 15:13:42 CEST) multi-call binary.

Usage: telnetd [OPTIONS]

Handle incoming telnet connections

Options:
        -l LOGIN        Exec LOGIN on connect
        -f ISSUE_FILE   Display ISSUE_FILE instead of /etc/issue
        -K              Close connection as soon as login exits
                        (normally wait until all programs close slave pty)
        -p PORT         Port to listen on
        -b ADDR[:PORT]  Address to bind to
        -F              Run in foreground
        -i              Inetd mode
        -w SEC          Inetd 'wait' mode, linger time SEC
        -S              Log to syslog (implied by -i or without -F and -w)
```

4.1.8 FTP service

The **ftp** service allows the user to transfer files to the target system via the FTP protocol.

Note: Due to the fact that ftp does not use encryption, it is recommended to deactivate this service in final products in order to avoid a security leak.

The ftp service has a startup link that points to the corresponding start script:

- **/etc/rc.d/S16ftpd -> /etc/init.d/ftpd**

The startup script simply starts **ftpd** as a daemon, using **/usr/bin/tcpsvd**. The usage of ftpd is shown by executing:

```
root@SANTARO:~ /usr/sbin/ftpd
BusyBox v1.18.5 (2013-04-26 15:13:42 CEST) multi-call binary.

Usage: ftpd [-wvS] [-t N] [-T N] [DIR]

Anonymous FTP server

ftpd should be used as an inetd service.
ftpd's line for inetd.conf:
        21 stream tcp nowait root ftpd ftpd /files/to/serve
It also can be ran from tcpsvd:
        tcpsvd -vE 0.0.0.0 21 ftpd /files/to/serve

Options:
        -w      Allow upload
        -v      Log to stderr
        -S      Log to syslog
        -t,-T   Idle and absolute timeouts
        DIR     Change root to this directory
```

4.1.9 Module loading

The **modules** service is responsible for external module loading at system startup. It has a startup link that points to the corresponding start script:

- **/etc/rc.d/S22modules -> /etc/init.d/modules**

The startup script simply looks which modules are listed in **/etc/modules** and loads them using **/sbin/modprobe**.

To ensure that the module loading works correctly, the module dependencies in **/lib/modules/<kernel version>/modules.dep** have to be consistent.

4.1.10 Network initialization

The **network initialization** service is responsible for initializing all network interfaces at system startup. Garz & Fricke systems use ifplugd to detect if an ethernet cable or an WLAN stick is plugged. It has a startup link that points to the corresponding start script:

- **/etc/rc.d/S92ifplugd -> /etc/init.d/ifplugd**

The network interfaces are listed on the target system in the configuration file **/etc/network/interfaces**. On conventional Linux systems, the user configures the network interfaces by hand using this file. On Garz & Fricke systems, there is a service called **sharedconf** as described in [▶ 4.1.11 Garz & Fricke shared configuration] that generates this file automatically according to the settings in the global XML configuration.

If the user wants to change the network settings, it is recommended to use the **sconfig** script as described in [▶ 4.2.1 Garz & Fricke system configuration].

Note: Changes that are made to **/etc/network/interfaces** directly will be overwritten by the sharedconf service on the next system startup and have no effect.

4.1.11 Garz & Fricke shared configuration

The **sharedconf** service reads shared configuration settings from the XML configuration and configures the Linux system accordingly. This includes network (as described in [▶ 4.1.10 Network initialization]) and touch configuration.

The sharedconf service has a startup link that points to the corresponding start script:

- **/etc/rc.d/S24sharedconf -> /etc/init.d/sharedconf**

4.1.12 Garz & Fricke Autocopy

This service is executed after the OS has booted and when a storage medium has been inserted. It is triggered together with the the **Autostart** service (see chapter [▶ 4.1.13 Garz & Fricke Autostart]) via UDEV. **Autocopy** always runs before **Autostart**.

The **Autocopy** service provides a comfortable installation and/or update functionality as well as copy mechanism for specific files that are not included in the OS (e.g. for runtime libraries).

Subfolders and files within a folder named **autocopy** on a USB stick, SD card or in the NAND flash will be copied to the root of the device resp. its equivalent targets. Non-existing folders will be created automatically.

The autocopy mechanism includes the following components:

- **/lib/udev/rules.d/80-udev-automount.rules**: UDEV rule that triggers the automounter and autounmounter script when a storage media is plugged/unplugged
- **/usr/sbin/automounter.sh**: Script that starts the automount script if a storage media is plugged
- **/usr/sbin/autounmounter.sh**: Script that unmounts the storage media and removes the mount point during unplugging
- **/usr/sbin/autounmount.sh**: and Script that is called by the automounter that enumerates creates a new mount point, mounts the device to it and executes the the autocopy and autostart sequence

The interesting part of the **/usr/sbin/automount.sh** is shown as follows:

```
...
# Check if an autocopy folder exists on the partition
if [ -d "$mount_point/autocopy" ]; then
        echo "Found an autocopy folder. Copying files ..." > /dev/console
        cp -R $mount_point/autocopy/* /
        echo "... done." > /dev/console
fi
...
```

The user may customize the copying process according to his / hers needs.

Example: The user has created and application **myapp** that should be installed to **/usr/bin/myapp** on the target and a library **mylib.so** used by this application that should be installed to **/usr/lib/mylib.so**. The layout of the storage medium is shown in figure [▶ Figure 4.1.12].

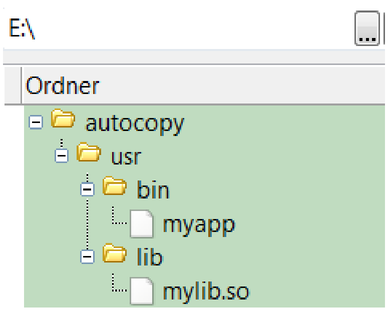


Figure 1: Layout of the storage medium after preperation with the custom files

After the target device is up and the storage media is plugged, the following message on the target's console is shown:

```
Mounted sda1 to /mnt/mstick1
Found an autocopy folder. Copying files ...
... done.
```

[▶ Figure 2] illustrates what happens in background. The files are properly transferred to the target.

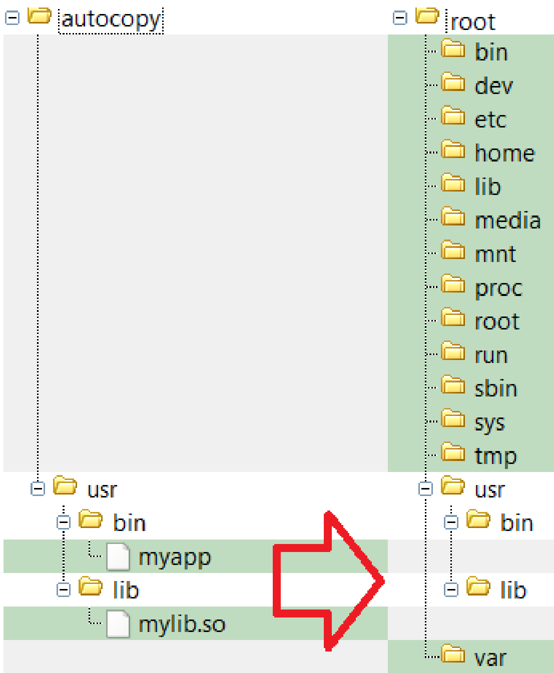


Figure 2: Automatic transfer process from storage medium (left hand) to the targets root file system (right hand) after plugging the storage

Warning: The user should be carefull by copying system files that may lead to an unusable system. Garz & Fricke refuses to carry responsibility for damages caused by the users copying process. Further, the user should consider to disable or restrict this mechanism to copy only carefully selected files by customizing the /usr/sbin/automount.sh script in the field to prevent missuse. Again, the responsibility is up to the user.

4.1.13 Garz & Fricke Autostart

The autostart service on Garz & Fricke Linux platforms comprises two different parts:

- The autostart mechanism if a storage media is plugged
- The initialization service startup

The former uses nearly the same mechanism as autocopy described in chapter [▶ 4.1.12 Garz & Fricke Autocopy] except for the autostart specific part in /usr/sbin/automount.sh:

```
...
# Search for autostart files and run them
if [ -d "$\$$mount_point/autostart" ]; then
        echo "Found an autstart folder. Executing files ..." > /dev/console
        for file in `find $\$$mount_point/autostart -type f -maxdepth 1 | sort`
        do
                if [ -x $\$$file ]; then
                        echo "Executing $\$$file..." > /dev/console
```

```
                        $\$$file
                fi
        done
        echo "... done." > /dev/console
fi
...
```

The autostart simply searches for an autostart folder in the root directory of the storage media. If found, the executable files are executed in alphabetic order. Filenames starting with digits are executed before those starting with letters in numeric order. Files in subfolders of the autostart are ignored.

The execution of the autostart files is shown in the Linux console during start up:

```
Mounted sda1 to /mnt/mstick1
Found an autocopy folder. Copying files ...
... done.
Found an autstart folder. Executing files ...
Mount point is /mnt/mstick1
Executing /mnt/mstick1/autostart/01autostart.sh...
Executing /mnt/mstick1/autostart/27another_application...
Executing /mnt/mstick1/autostart/a_autostart...
Executing /mnt/mstick1/autostart/b_autostart...
... done.
```

As already stated in [▶ 4.1.12 Garz & Fricke Autocopy] autocopy is executed before autostart.

The user may desire to autostart an application from a storage media with command line args. In this case a start script can be placed in the autostart folder that starts the aplication itself in a subfolder of autostart with the desired command line args. It is important to place the application itself in a subfolder. Otherwise the autostart mechainism will try to start this application without the command line args in parallel to the start script.

Example: The user created a Qt application (e.g. myapp) to run ontop of the QWS server. This makes it necessary to pass -qws to myapp. The application is placed e.g. in the folder /autostart/custom on the storage media. Consequently, the start script (e.g. myapp.sh) must have the following contents:

```
#!/bin/sh

./custom/myapp -qws
```

The start scrip must be placed in the folder /autostart on the storage media. The layout of the storage medium is shown in figure [▶ Figure 3].

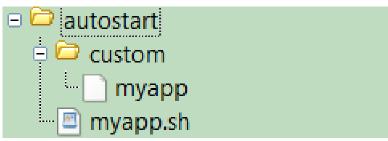


Figure 3: Layout of the storage medium after preperation with the custom files

After plugging the storage media into the target system. The start script and the apllication should start properly.

The second part of the autostart mechanism affects the automatic start of services and applications that are installed on the target's root file system in contrast to those running from a plugged storage media.

This mechanism is very common on Linux systems and is called System V init. The System V init searches the folder /etc/rc.d for scripts that follow the naming conventtion <S|K><Number><Name> (e.g. S95myapp). Scripts starting with S are executed during system startup with the command line option start in numerical order. Similary, scripts starting with K are executed during system shutdown with the command line option stop in numerical order. To be able to use one script for startup and shutdown these scripts are links to real scripts in the directory /etc/init.d (e.g. /etc/init.d/myapp). This script has the following basic layout:

```
#!/bin/sh

. /etc/profile

case "$\$$1" in
start)
        # Add here command that should execute during system startup.
        ;;
stop)
        # Add here command that should execute during system shutdown.
        ;;
*)
        echo "Usage: ... " >&2
        exit 1
        ;;
esac
```

In chapter [▶ 8 Building a user application for the target system] this mechanism will be used for automatic application start up.

4.2 Utilities

4.2.1 Garz & Fricke system configuration

The /etc/init.d/sharedconf script (see [▶ 4.1.11 Garz & Fricke shared configuration]) can be used to change the shared system configuration. For this purpose, there is a link to the script at /usr/bin/sconfig which can be called without the absolute path:

```
root@SANTARO:~ sconfig
```

If called without any parameters, the command prints the usage:

```
Usage: /usr/bin/sconfig {start | setting [value]}
   Call without [value] to read a setting, call with [value] to write it.
Available settings:
   serialdiag  switch serial debug console on or off
   dhcp        switch DHCP on or off
   ip          set IP address
   mask        set subnet mask
   gateway     set standard network gateway
   mac         set MAC address
   name        set device name
   serial      set serial number (affects MAC address and device name)
   rotation    set display rotation
```

If the script is called with a setting as parameter, the setting is read from the XML configuration and displayed on the console. If additionally a value is appended, this value is written to the according setting in the XML configuration.

Example 1: To activate DHCP on the device, type:

```
root@SANTARO:~ sconfig dhcp on
```

Example 2: To deactivate DHCP and set a static IP address, type:

```
root@SANTARO:~ sconfig dhcp off
root@SANTARO:~ sconfig ip 192.168.1.123
```

4.2.2 Installing a custom bootlogo

Garz & Fricke systems shipped with displays or HDMI controllers can show a PNG bootlogo on system start up. For this purpose, a custom PNG logo support has been added to the Linux kernel. Custom bootlogos can only be shown under one of the following conditions:

- A generic bootlogo license (XML file) is installed on the system
- The PNG logo to be shown contains a bootlogo license

Both features are subject to a license fee. Garz & Fricke offers two license models:

- The economic **basic license** contains one single PNG file, that will be signed with an invisible watermark by Garz & Fricke containing the license. This PNG can be used for all Garz & Fricke systems, independent from quantity and system type.
- The flexible **corporate license** contains a generic bootlogo license file as well as the **logolic** Windows tool. The XML file (**rb-logolicense.xml**) allows the usage of any PNG file, signed or not, which makes it easier for the end-user to modify the bootlogo. The **logolic** Windows tool enables the user to prepare a simple PNG file into a signed bootlogo PNG file. This license model is also totally independent from the quantity of operated systems.

For more information about logo licensing or the **logolic** Windows tool, please contact the Garz & Fricke sales.

Example 1 shows how to install the generic logo license from a TFTP server with the IP address **192.168.1.100**:

```
root@SANTARO:~ tftp -g -r rb-logolicense.xml 192.168.1.100
root@SANTARO:~ xconfig import -b rb-logolicense.xml
```

Because the bootlogo has to be shown as early as possible, the Linux kernel cannot wait until the file system is up. Hence, the booloader has to pass the PNG file via a main memory area. The bootloader can be instructed to load a PNG file from one of the eMMC boot partitions. This can be done by placing the following command **before** the **exec** command in the **boot.cfg** file on **/dev/mmcblk0p2**:

```
load -b <physical memory address> [-p <partition name>] <file name>
```

Furthermore, the kernel has to be informed about the physical memory address where the file is located and the file size via the **logo** kernel command parameter. The following format has to be used:

```
logo=<physical address>,<file size>
```

Apart from the above format, a display test logo can be shown with the **logo** kernel command line parameter (without any bootlogo license) as follows:

```
logo=test
```

The orientation of the bootlogo is bound to the global **rotation** system setting. The **rotation** can be adjusted with the **sconfig** tool described in [▶ 4.2.1 Garz & Fricke system configuration].

Example 2 shows how to load a PNG bootlogo **logo.png** with a file size of **30515 bytes** to **/dev/mmcblk0p2** from a TFTP server with the IP address **192.168.1.100** and how to configure the bootloader and the kernel to pass and show this bootlogo.

First step: upload the logo.

```
root@SANTARO:~ mount /dev/mmcblk0p2 /mnt
root@SANTARO:~ tftp -g -r logo.png -l /mnt/logo.png 192.168.1.100
```

Second step: configure bootloader and kernel. Edit the file **/mnt/boot.cfg** (e.g. with **nano** or **vi**) and add the highlighted sections:

```
load -b 0x12000000 -p config config.xml
load -b 0x12008000 logo.png
load linuximage
exec "console=ttymxc0,115200 root=/dev/mmcblkp3 logo=0x12008000,30515 xmlram=0
    ↪ x12000000"
```

```
root@SANTARO:~ umount /mnt
```

5 Accessing the hardware

This chapter gives a short overview of how to access the different hardware parts and interfaces from within the Linux operating system. It is written universally in order to fit all Garz & Fricke platforms in general.

5.1 Digital I/O

The digital I/O pins for a platform are controlled by the kernel's GPIO Library interface driver. This driver exports a sysfs interface to the user space. For each pin, there is a virtual folder in the file system under /sys/class/gpio/ with the same name as in the hardware manual.

Each folder contains the following virtual files that can be accessed like normal files. In the command shell, there are the standard Linux commands cat for read access and echo for write access. To acces those virtual files from C/C++ code, the standard POSIX operations open(), read(), write() and close() can be used.

sysfs file	Valid values	Meaning
value	0, 1	The current level of the GPIO pin. The acive_low flag (see below) has to be taken into account for interpretation.
direction	in, out	The direction of the GPIO pin.
active_low	0, 1	Indicates if the pin is interpreted active low.

Most of the Garz & Fricke hardware platforms include a dedicated connector with isolated digital I/O pins. On these pins, the direction cannot be changed, since it is determined by the wiring. Thus, the direction file is missing here. Some platforms also have a keypad connector, which can be used as a bi-directional GPIO port.

The following examples show how to use the virtual files in order to control the GPIO pins.

Example 1: Verify that the GPIO pin keypad_pin7, which is pin 7 on the keypad connector, is interpreted as active high, configure the pin as an output and set it to high level in the Linux shell:

```
root@SANTARO:~ cat /sys/class/gpio/keypad_pin7/active_low
0
root@SANTARO:~ echo out > /sys/class/gpio/keypad_pin7/direction
root@SANTARO:~ echo 1 > /sys/class/gpio/keypad_pin7/value
```

Example 2: Verify that keypad_pin12 is an input pin and interpreted as active low and verify that the level LOW is recognized by this pin in the Linux shell:

```
root@SANTARO:~ cat /sys/class/gpio/keypad_pin12/direction
in
root@SANTARO:~ cat /sys/class/gpio/keypad_pin12/active_low
1
root@SANTARO:~ cat /sys/class/gpio/keypad_pin12/value
1
```

Example 3: C function to set / clear the dig_out1 output pin:

```
void set_gpio(unsigned int state)
{
        int fd = -1; // GPIO file handle
        char gpio[4];

        fd = open("/sys/class/gpio/dig_out1/value", O_RDWR);
        if (fd == -1)
        {
                printf("GPIO-ERR\n");
                return;
        }
        sprintf(gpio, "%d", state);
        write(fd, gpio, strlen(gpio));
        close(fd);
}
```

A more detailed documentation of the GPIO handling in the Linux kernel can be found in the documentation directory of the Linux kernel source tree. After building the BSP for SANTARO, this documentation can be found under:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/gpio.txt

5.2 Serial interfaces (RS-232 / RS-485 / MDB)

Most of the serial interfaces are exported as TTY devices and thus accessible via their device nodes under **/dev/ttymxc<number>**. Depending on your hardware platform (and maybe its assembly option), different transceivers (RS232, RS485, MDB) can be connected to these TTY devices. See the following table for the mapping between device nodes and hardware connectors on SANTARO:

TTY device	Hardware function
`/dev/ttymxc0`	RS-232 No. 1 (X13)
`/dev/ttymxc1`	RS-232 No. 2 / MDB (X13)
`/dev/ttymxc2`	RS-485 (X39)
`/dev/ttymxc3`	internal UART (X11)

RS485 can be used in half duplex or full duplex mode. This mode has to be set on the hardware (see the according hardware manual) as well as in the software. Per default, the interface is working in full duplex mode. See the following C code example for setting the RS485 interface to half duplex mode:

```
#include <termios.h>

void set_rs485_half_duplex_mode()
{
    struct serial_rs485 rs485;
    int fd;

    /* Open port */
    fd = open ("/dev/ttymxc2", O_RDWR | O_SYNC);

    /* Enable RS485 half duplex mode */
    rs485.flags = SER_RS485_ENABLED | SER_RS485_RTS_ON_SEND;
    ioctl(fd, TIOCSRS485, &rs485);

    close(fd);
}
```

For a full source code example, see the BSP folder **local_src/common/ltp_guf_tests/testcases/rs485pingpong**.

Interfaces with an MDB transceiver should not be accessed directly via their device nodes. Instead, there is a library for MDB communication in the BSP. Please see the folder **local_src/common/libmdb_test** for a full source code example.

5.3 Ethernet

If all network devices are properly configured as described in [▶ 4.2.1 Garz & Fricke system configuration] and [▶ 4.1.10 Network initialization] they can be used by the POSIX socket API.

There is a huge amount of documentation about socket programming available. Therefore it is not documented here.

The POSIX specification is available at:

▶ http://pubs.opengroup.org/onlinepubs/9699919799/functions/contents.html

5.4 Real Time Clock (RTC)

All Garz & Fricke systems are equipped with a hardware real time clock. The system time is automatically set during the boot sequence by reading the RTC. You can read the current time and date using the Linux **hwclock** command:

```
root@SANTARO:~ hwclock --show
Fri Jun  1 14:51:12 UTC 2012
```

The RTC time cannot be adjusted directly in one command because only the current system time can be transferred to the RTC. Thus, the system time has to be set first, using the **date** command, and can then be written to the RTC:

```
root@SANTARO:~ date 2010.09.09-16:50:00
Thu Sep  9 16:50:00 UTC 2010
root@SANTARO:~ hwclock --systohc
```

5.5 SPI

There are two ways of controlling SPI bus devices from a Linux system:

- By writing a kernel SPI device driver from space and accessing this driver from user space by using its interface.
- By accessing the SPI bus via the Linux kernels **spidev** API directly.

Describing the process of writing a Linux SPI device driver is out of this scope of this manual. The AT25 SPI eemprom can serve as a good and simple example for such a driver:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/drivers/misc/eeprom/at25.c

The interface provided to the user space by such a kernel driver depends of the sort of this driver (e.g. character misc driver, input subsystem device driver, etc.). A very common usecase for an SPI driver is a touchscreen driver that uses the input event subsystem.

Accessing the SPI bus from userspace directly via spidev is described in the Linux kernel documentation and available in the BSP under:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/spi/spidev

Additionally, there is an example C program available in the same location:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/spi/spidev_test.c

The header for spidev is available inside the BSP under:

- platform-SANTARO/sysroot-target/kernel-headers/include/linux/spi/spidev.h

Note: If spidev is used to access the SPI bus directly, the user is responible for keeping the interoperability consistent with all other SPI devices that are controlled by the Linux kernel.

5.6 I2C

There are two ways of controlling I2C bus devices from a Linux system:

- By writing a kernel I2C device driver from space and accessing this driver from user space by using its interface.
- By accessing the I2C bus via the Linux kernels **i2c-dev** API directly.

Describing the process of writing a Linux I2C device driver is out of this scope of this manual. The AT24 I2C eemprom can serve as a good and simple example for such a driver:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/drivers/misc/eeprom/at24.c

The interface provided to the user space by such a kernel driver depends of the sort of this driver (e.g. character misc driver, input subsystem device driver, etc.). A very common usecase for an I2C driver is a touchscreen driver that uses the input event subsystem.

Accessing the I2C bus from userspace directly via spidev is described in the Linux kernel documentation and avauilable inside the BSP under:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/i2c/dev-interface

The header for i2c-dev is available inside the BSP under:

- platform-SANTARO/sysroot-target/kernel-headers/include/linux/i2c-dev.h

Note: If i2c-dev is used to access the I2C bus directly, the user is responible for keeping the interoperability consistent with all other I2C devices that are controlled by the Linux kernel.

5.7 CAN Bus

CAN bus devices are controlled through the SocketCAN framework in the Linux kernel. As a consequence, CAN interfaces are network interfaces. Applications receive and transmit CAN messages via the BSD Socket API. CAN interfaces are configured via the netlink protocol. Additionally, Garz & Fricke Linux systems are shipped with the **canutils** package to control and test the CAN bus from the command line.

On SANTARO the CAN bus is physically available on connector X39.

Example 1 shows how a CAN bus interface can be set up properly for 125 kBit/s from a Linux console:

```
root@SANTARO:~ canconfig can0 bitrate 125000
root@SANTARO:~ ifconfig can0 up
```

Note: Due to the use of the busybox version of the **ip** tool the following sequence does **NOT** work on Garz & Fricke Linux systems:

```
root@SANTARO:~ ip link set can0 type can bitrate 125000
root@SANTARO:~ ifconfig can0 up
```

As already stated above, CAN messages can be sent through the BSD Socket API. The structure for a CAN message is defined in the kernel header **platform-SANTARO/sysroot-target/kernel-headers/include/linux/can.h**:

```
struct can_frame {
        u32 can_id; /* 29 bit CAN_ID + flags */
        u8 can_dlc; /* data length code: 0 .. 8 */
        u8 data[8];
};
```

Example 2 shows how to initialize SocketCAN from a C program:

```
int iSock;
struct sockaddr_can addr;

iSock = socket(PF_CAN, SOCK_RAW, CAN_RAW);

addr.can_family = AF_CAN;
addr.can_ifindex = if_nametoindex("can0");

bind(iSock, (struct sockaddr *)&addr,
        sizeof(addr);
```

Example 3 shows how a CAN message is sent from a C program:

```
struct can_frame frame;

frame.can_id = 0x123;
frame.can_dlc = 1;
frame.data[0] = 0xAB;

nbytes = write(iSock, &frame,
        sizeof(frame));
```

Example 4 shows how a CAN message is received from a C program:

```
struct can_frame frame;

nbytes = read(iSock, &frame, sizeof(frame));

if (nbytes > 0) {
        printf("ID=0x%X DLC=%d data[0]=0x%X\n",
        frame.can_id,
        frame.can_dlc,
        frame.data[0]);
}
```

Example 5 shows how a CAN message with four bytes with the standard ID 0x20 is sent on **can0** from the Linux shell, using the **cansend** tool. The CAN bus has to be physically prepared properly and there has to by at least one other node that is configured to read on this message ID for this task. Furthermore, all nodes must have the same bittiming.

```
root@SANTARO:~ cansend can0 -i 0x20 0xca 0xbe 0xca 0xbe
```

Example 6 shows how all CAN messages are read on **can0** using the **candump** tool:

```
root@SANTARO:~ candump can0
```

A more detailled documentation of the CAN bus handling in the Linux kernel can be found in the documentation directory of the Linux kernel source tree. After building the BSP for SANTARO, this documentation can be found under:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/networking/can.txt

5.8 USB

There are two general types of USB devices:

- **USB Host**: the Linux platform device is the host and contols several devices supported by corresponding Linux drivers
- **USB Device**: the Linux platform device acts as a USB device itself by emulating a specific device type

Additionally, if supported, an OTG-enabled port can automatically detect, which of the above roles the platform plays during the plugging process.

5.8.1 USB Host

For USB Host functionality, Garz & Fricke platforms per default support the following devices:

- USB Mass Storage
- USB Keyboard

There are many more device drivers available in the Linux kernel. They are not activated by default, because Garz & Fricke cannot maintain and test the huge amount of existing drivers. Instead, the customer may do this himself or engage Garz & Fricke to implement his special use case. Existing drivers can easily be activated by reconfiguring and rebuilding the Linux kernel inside the BSP.

The USB Host bus can also be directly accessed by using **libusb**. This library is installed on Garz & Fricke platforms per default.

The header is available inside the BSP under:

- platform-SANTARO-0/sysroot-target/usr/include/libusb-1.0/libusb.h

The library is available inside the BSP under:

- platform-SANTARO-0/sysroot-target/usr/lib/libusb-1.0.so

Further information about libusb can be found under:

▶ http://libusb.sourceforge.net/api-1.0

Note: If libusb is used to access the USB bus directly, the user is responsible to keep the interoperability consistent with all other USB devices that are controlled by the Linux kernel.

5.8.2 USB Device

For USB Device functionality, the following device emulations are supported per default:

- USB Serial Gadget

Again, further drivers can be activated by reconfiguring the Linux kernel. The USB Device drivers are not compiled into the kernel by default, but are located as modules in the file system. In order to use the Serial Gadget for example, the according module has to be loaded:

```
root@SANTARO:~ modprobe g_serial
```

The Serial Gadget creates a virtual serial port over USB, accessible via the device node **/dev/ttyGS0**.

5.9 Display backlight

The brightness of the display backlight can be adjusted in a range from 0 to 255. The value is exported as a virtual file in the sysfs under **/sys/class/backlight/pwm-backlight/brightness**. It can be accessed using the standard file operations open(), read(), write() and close().

Example 1: Reading and adjusting the current backlight brightness on the console:

```
root@SANTARO:~ cat /sys/class/backlight/pwm-backlight/brightness
255
root@SANTARO:~ echo 100 > /sys/class/backlight/pwm-backlight/brightness
```

Please note that this value is not persistent, i.e. it gets lost when the device is rebooted. In order to change the brightness permanently, it has to be set in the XML configuration, which can be done using the **xconfig** tool.

Example 2: Adjusting the backlight brightness permanently on the console:

```
root@SANTARO:~ xconfig addattribute -p /configurationFile/variables/display/backlight
    ↪ -n level_ac -v 100
```

Please note that adjusting this value does not affect the brightness immediately. A reboot is required for this setting to take effect. If you want to adjust the brightness immediately and permanently, you have to execute both examples.

5.10 SD cards and USB mass storage

SD cards and USB mass storage devices can be accessed directly by using their devices nodes. The SD card can be found under **/dev/mmcblk0**, its single partitions are located under **/dev/mmcblk0pX** with X being a positive number. The USB mass storage devices can be found under **/dev/sdX with** X=a..z, its single partitions under **/dev/sdXY** with Y being a positive number.

Example 1: Create a FAT32 file system on the first partition of an SD card:

```
root@SANTARO:~ mkfs.vfat /dev/mmcblk0p1
```

If the first partition on an SD card or a USB mass storage device already contains a file system when it is plugged into the device, it is mounted automatically by the Garz & Fricke **automount** service, which is implemented using **udev** rules. SD card partitions are mounted to mount point **/mnt/mmcX** with X being a positive number, and USB mass storage devices are mounted to mount point **/mnt/mstickX** with X beeing a positive number.

All further partitions or partitions with a newly created file system have to be mounted manually, like shown in the following examples.

Example 2: Mount the first partition on an SD Card into a newly created directory:

```
root@SANTARO:~ mkdir /my_sdcard
root@SANTARO:~ mount /dev/mmcblk0p1 /my_sdcard
```

Example 3: Mount the first partition on a USB mass storage device into a newly created directory:

```
root@SANTARO:~ mkdir /my_usb_drive
root@SANTARO:~ mount /dev/sda1 /my_usb_drive
```

5.11 Touchscreen

Although, the most common way to access the touchscreen is to use it in conjuction with a GUI framework like Qt while the access to it is hidden in a backend, it is possible to access the touchscreen directly from two lower levels:

- by using the **tslib** library
- by using the **Linux input subsystem** kernel API

5.11.1 tslib

The header of tslib is available inside the BSP under:

- platform-SANTARO/sysroot-target/usr/include/tslib.h

The library is available inside the BSP under:

- platform-SANTARO/sysroot-target/usr/lib/libts.so

The **ts_test** application included in tslib's source tree can serve as a programming example:

- platform-SANTARO/build-target/tslib-1.0/tests/ts_test.c

Garz & Fricke ensured that signal filtering is already optimized for the touchscreens that are shipped with their products by choosing a suitable set of filters with suitable parameters in tslib. The filter configuration can be altered in the configuration file **/etc/ts.conf** in the target's root filesystem. This should only be done if the user is familar with tslib's filter system and the properties and characteristics of its filters. It is also important to reboot the system properly after altering this configuartion file or executing the **sync** command. Otherwise, the changes may get lost during a hard reset.

5.11.2 Input subsystem

Documentation about the input event subsystem can be found after building the BSP under:

- platform-SANTARO/build-target/linux-fsl-3.0.35-1.44.4-0/Documentation/input/input.txt

The header for the input subsystem is available under:

- platform-SANTARO/sysroot-target/kernel-headers/include/input/input.txt

There is currently no full-featured demo application available for using the touchscreen directly through the input subsystems evdev inteface. However the tslib's **input-raw** module shows how this interface can be handled:

- platform-SANTARO/build-target/tslib-1.0/plugins/input-raw.c

Note: Due to the nature of electric circuits, there may be more or less noise on the signals of the touchscreen that has be filtered out. Usually this isn't done by the device driver itself. Instead, a set of filters in userspace are used (e.g. in tslib). If using the input event subsystem directly, the user has to take care of the filtering by himself.

5.12 Audio

Garz & Fricke systems with an integrated audio codec make use of ALSA (Advanced Linux Sound Architecture) as a software interface. This project includes a user-space library (alsa-lib) and a set of tools (aplay, arecord, amixer, alsactl) for controlling and accessing the audio hardware from user applications or the console. Please refer to the official ALSA webpage for a full documentation:

► http://www.alsa-project.org

For a quick start here are three short examples of how to play/record an audio file and how to adjust the playback volume.

Example 1: Play the audio file **my_audio_file.wav** from the console using **aplay**:

```
root@SANTARO:~ aplay my_audio_file.wav
```

Example 2: Record the audio file **my_recorded_audio_file.wav** from the console using **arecord**:

```
root@SANTARO:~ arecord -f cd -t wav > my_recorded_audio_file.wav
```

Example 3: Set the playback volume to half of the maximum:

```
root@SANTARO:~ amixer sset 'Master Playback Volume' 50%
```

The **amixer** command can be used for several settings regarding the audio hardware. Called without parameters, it gives a list of all available settings along with their possible values.

ALSA is also used in conjuction with playing multimedia files with GStreamer via the **alsasink** plugin after decoding the audio data from an audio stream.

Example 4: Play a sine signal with a frequency of 440 Hz (default settings) with GStreamer's **adiotestsrc** and **alsasink** plugins:

```
root@SANTARO:~ gst-launch audiotestsrc ! audioconvert ! alsasink
```

To play a **.wav** file from a Qt application, a **QSound** object can be used. A demo application can be found inside the BSP at:

- **local_src/common/qt4-guf-sound**

5.13 SRAM

The battery-backed SRAM is controlled by the MTD subsystem in the Linux kernel. Therefore, it can be handled like a typical MTD device via the device handles **/dev/mtdX** and **/dev/mtdblockX**, where **X** is the logical number of the device. This number can be found by executing:

```
root@SANTARO:~ cat /proc/mtd | grep SRAM
```

Per default, the SRAM is located at **/dev/mtd0**.

A very common use of the SRAM in conjuction with the MTD subsystem is to create a file system on top of it, like shown in the following example.

Example: Create a JFFS2 file system on **/dev/mtd0** with the mtd-utils and mount it to **/mnt**

```
root@SANTARO:~ flash_eraseall /dev/mtd0
root@SANTARO:~ mkfs.jffs2 /dev/mtdblock0
root@SANTARO:~ mount /dev/mtdblock0 -t jffs2 /mnt
```

5.14 Video

Garz & Fricke systems make use of **Video4Linux2** to handle video input (if present) and output. On top of this, the **GStreamer** framework is installed for easy playback of video files. GStreamer automatically uses the VPU (video processing unit) of the i.MX6 MCU for hardware acceleration. For this purpose, several libraries and plugins from Freescale have been integrated into the BSP.

A complete list of supported GStreamer plugins can be listed with the command:

```
root@SANTARO:~ gst-inspect
```

To list the plugins shipped by Freescale only, the following command can be used:

```
root@SANTARO:~ gst-inspect | grep imx
```

You can use **gplay** to play videos from the shell:

```
root@SANTARO:~ gplay /usr/share/qt4-guf-demo/videos/cc-BigBuckBunny-Trailer-800x448-
    ↪ h264-stereo.mp4
```

Videos can be integrated into a Qt application using **Phonon**. The Garz & Fricke demo application in the BSP shows how to use Phonon using the **VideoPlayer** class:

- local_src/common/qt4-guf-demo/qvideo.cpp

The VideoPlayer class is optimized for easy usage. For higher flexibility, the **VideoWidget** class can be used instead.

Further information about GStreamer can be found under:

- http://gstreamer.freedesktop.org

Further information about Phonon can be found under:

- http://qt-project.org/doc/qt-4.7/phonon-module.html

5.15 HDMI

An HDMI device connected to SANTARO is configured automatically via EDID. The monitor is registered as a separate framebuffer **/dev/fb2**. Without any modifications, the HDMI device will show the SANTARO bootlogo in the center of the screen.

In Qt, the HDMI port can be used as primary or secondary display. Furthermore, an audio-enabled HDMI device can be used as secondary sound card. A set of Qt environment variables defines how the HDMI device is used. They are configured in **/etc/profile**:

- **QWS_DISPLAY**: configures the display devices
- **PHONON_GST_AUDIOSINK**: configures the GStreamer sound device
- **QWS_MOUSE_PROTO**: configures the input devices (touchscreen, mouse)

The following sections describe how the variables have to be set for different use cases. Most example lines are already present in **/etc/profile** with a preceeding comment sign ('#'). They can be activated by simply removing this sign and adding it to the default line instead.

For the changes to take effect, either the device has to be rebooted, or the file has to be sourced into the current shell:

```
root@SANTARO:~ . /etc/profile
```

In each case, the Qt application has to be restarted afterwards.

5.15.1 Configuring Qt to use an HDMI display

Case 1: Using the built-in display only

This is the default setting. The built-in display is registered as framebuffer device **/dev/fb0**. If your device does not have a built-in display, this device node is present anyway.

```
export QWS_DISPLAY="transformed:rot$ROTATION:/dev/fb0:0"
```

Case 2: Using the HDMI display only

The HDMI device can be used as the primary display in Qt by changing **/dev/fb0** to **/dev/fb2**:

```
export QWS_DISPLAY="transformed:rot$ROTATION:/dev/fb2:0"
```

Case 3: Using built-in display and HDMI display simultaneously

This setting is a bit more complex. It includes both framebuffers:

```
export QWS_DISPLAY="multi: transformed:rot$ROTATION:/dev/fb0:0 linuxfb:/dev/fb2:
    ↪ offset=0,$YRES:1 :2"
```

Using this line makes the built-in display the primary display (starting at coordinates 0/0), while the HDMI display virtually starts below the built-in display (starting at coordinates 0/<display-height>).

5.15.2 Setting the HDMI display resolution

If your HDMI device does not support EDID to report its resolution and framerate to the HDMI host or if you want to use a resolution different from the native resolution, you can do that by appending a **video** parameter to the kernel command line. It is stored in the **boot.cfg** file on **/dev/mmcblk0p2**. See [▶ 7.1.2 Target configuration] for information on how to edit this file.

The following example configures the HDMI device to use a resolution of 800x600, 16 Million colours and 60 Hz framerate:

```
load -b 0x12000000 -p config config.xml
load linuximage
exec "console=ttymxc0,115200 root=/dev/mmcblkp3 video=mxcfb1:dev:hdmi,800x600-16M@60
    ↪ xmlram=0x12000000"
```

5.15.3 Configuring Phonon/gstreamer to use an HDMI audio device

Some HDMI devices have built-in speakers and thus can be used as an audio device. Garz & Fricke has extended the behaviour of **PHONON_GST_AUDIOSINK** with an ALSA device selection option that is passed to GStreamer's **alsasink** parameter. In this way, the Phonon sound output can be switched between different sound devices. Instead of only setting **PHONON_GST_AUDIOSINK="alsasink"** the ALSA device can additionally appended, separated by a blank (**PHONON_GST_AUDIOSINK="alsasink <ALSA device>"**).

The default setting uses the internal audio codec as the system's audio device:

```
export PHONON_GST_AUDIOSINK="alsasink hw:0,0"
```

This line uses to the external HDMI audio device:

```
export PHONON_GST_AUDIOSINK="alsasink hw:1,0"
```

5.15.4 Additional information

Additional information about the **QWS_MOUSE_PROTO** and the **QWS_DISPLAY** environment variables can be found at:

- http://doc.qt.digia.com/4.7/qt-embedded-envvars.html

Additional information about the **PHONON_GST_AUDIOSINK** environment variable can be found at:

- http://community.kde.org/Phonon/Environment_Variables

5.16 WLAN

The Garz & Fricke products support WLAN using WLAN USB dongles. Encryption WPA and WPA2 is supported via the WPA Supplicant tool.

The following explanations were tested using the **AbiCom WL250N-USB** dongle. This dongle uses a Ralink chipset compatible with mainline kernel drivers. Because encryption should be used for security reasons, the WLAN connection needs to be configured before it will function properly.

This is done using the **/etc/wpa_supplicant.conf** file. It may be edited using a text editor, e.g. nano. The following listing shows an example of the **/etc/wpa_supplicant.conf** file.

```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=1
ap_scan=1
network={
        ssid="[YOUR-SSID]"
        scan_ssid=1
        #proto=WPA
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP TKIP
        group=CCMP TKIP
        psk="[YOUR-WIFI-PASSWORD]"
}
```

Replace the parts in square brackets with your configuration. The **ssid** states the network name and the **psk** contains the WLAN password. To use WPA instead of WPA2 encryption, replace the **proto** assignment to **proto=WPA**.

Due to the usage of **ifplugd** in combination with **udev** the network interface is reconfigured automatically on hotplug.

WLAN configuration can also be done automatically:

```
root@SANTARO-0:~ wpa_supplicant -i wlan0 -c /etc/wpa_supplicant.conf -B
```

If the WLAN offers DHCP, **udhcpc** can be used like this:

```
root@SANTARO-0:~ udhcpc -i wlan0
udhcpc (v1.18.5) started
Sending discover...
Sending discover...
Sending select for 192.168.0.107...
Lease of 192.168.0.107 obtained, lease time 604800
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.0.1
```

For further information consult some of the following Internet resources:

- http://hostap.epitest.fi/wpa_supplicant/
- http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html
- http://www.linuxhomenetworking.com/wiki/

If you want to modify the **/etc/network/interfaces** file, you should do this by modifiying the **generate_network_interfaces** function in **/etc/init.d/sharedconf**.

6 Building a Garz & Fricke embedded Linux system from source

This chapter describes how to build a Linux BSP for a Garz & Fricke platform from source. All steps, including the installation of the build system and the required toolchains, are covered here.

6.1 General information about Garz & Fricke embedded Linux systems

Garz & Fricke uses Pengutronix **PTXdist** for building embedded Linux systems for their platforms by providing a Board Support Package (BSP). PTXdist is a configurable build system specializing in building embedded Linux systems. This chapter contains information about the handling of Linux with **PTXdist** and OSELAS.Toolchain() for Garz & Fricke systems. For further documentation on **PTXdist** please refer to the official **PTXdist website**:

- http://www.ptxdist.org/software/ptxdist/index_en.html

Information regarding OSELAS.Toolchain() can be found at:

- http://www.pengutronix.de/oselas/toolchain/index_en.html

In order to build a Linux based system, the following list of packages should be installed (Debian and Ubuntu package names):

- atftpd
- autoconf
- automake
- bison
- expect
- flex
- gawk
- g++
- nfs-kernel-server
- libncurses-dev
- libtool
- libxml-parser-perl
- make
- minicom
- python-dev
- python3-dev
- quilt
- texlive-all
- texinfo
- xinetd

On Debian based Linux distributions packages can be installed using the **apt-get** command:

```
$ sudo apt-get install <package_Name>
```

To install all the previous listed packages type:

```
$ sudo apt-get install autoconf automake bison expect flex gawk g++ kernel-nfs-server
    ↪  libncurses-dev libtool libxml-parser-perl make minicom quilt texinfo xinetd
```

Building a Linux system is a very complex task. In addition to a GNU toolchain consisting of a compiler (**gcc**), a set of binary utilities (**as**, **readelf**, **strip** ...), a set of basic runtime libraries (**libc**, **libm**, ...) and a debugger (**gdb**) some tools like **GNU autotools**, **pkg-config**, **make**, **qmake**, **cmake**, **sed**, etc. are needed to satisfy this task. For this reason the complexity of the build process is hidden behind the Linux build system **PTXdist**.

In contrast to a desktop Linux system, which is completely built with a native GNU toolchain, an embedded Linux system is built with a GNU cross toolchain. A cross toolchain must have the ability to produce target specific opcode while running on a different host system.

To distinguish between the native GNU toolchain and the GNU cross toolchain, the GNU cross tools are prefixed with a triplet. E.g. if the toolchain produces opcode for an **ARMv5TE** core having library routines that can deal with Linux system calls satisfying the **GNU EABI**, the compiler is named **arm-v5te-linux-gnueabi-gcc**, the assembler is named **arm-v5te-linux-gnueabi-as**, and so on. Sometimes a toolchain prefix is only named **arm-linux-** or something else. This depends on the toolchain vendor. Garz & Fricke uses the naming convention stated before.

The build of the embedded Linux system is divided into three parts, covered in the following chapters:

- Installing PTXdist [▶ 6.2 Installing PTXDist]
- Installing the GNU cross toolchain for the target architecture [▶ 6.3 Installing the GNU cross toolchain for the target architecure]
- Building the BSP for the target platform [▶ 6.5 Building the BSP for the target platform with PTXDist]

6.2 Installing PTXDist

PTXdist supports Linux as a host system only. To install PTXdist the following files from the CD / USB stick shipped with the starter kit for SANTARO have to be extracted:

- **Tools/ptxdist-2011.09.0-guf-0.tgz**

The PTXdist and patches packets have to be extracted into a working 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
```

The next step is to copy the archive from the mounted USB stick and extract it:

```
$ cp /<mountpoint of USB mass storage>/Tools/ptxdist-2011.09.0-guf-0.tar.bz2 .
$ tar -xf ptxdist-2011.09.0-guf-0.tar.bz2
```

If everything went right, we have a **ptxdist-2011.09.0-guf-0** directory now, so we can change into it:

```
$ cd ptxdist-2011.09.0-guf-0
```

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:

```
$ ./configure
```

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

```
[...]
checking whether /usr/bin/patch will work... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in
ptxdist version <version> configured.
Using '/usr/local' for installation prefix.
Report bugs to ptxdist@pengutronix.de
```

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, 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 whether 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 before re-running the configuration script.

When the configuration-script is finished successfully, we can start the build process:

```
$ make
```

If there are no errors we can install PTXdist into its final location. In order to write to /usr/local, this step has to be performed as user root:

```
$ sudo make install
[enter password]
[...]
```

If we do not have root access to the machine it is also possible to install PTXdist 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 done now, so the temporary folder may be removed:

```
$ cd ../../
$ rm -rf local
```

To be sure that PTXDist can access all source packages to build from web, the source download URL must be set up correctly:

```
$ ptxdist setup
```

The menu as shown in [▶ Figure 4] appears.

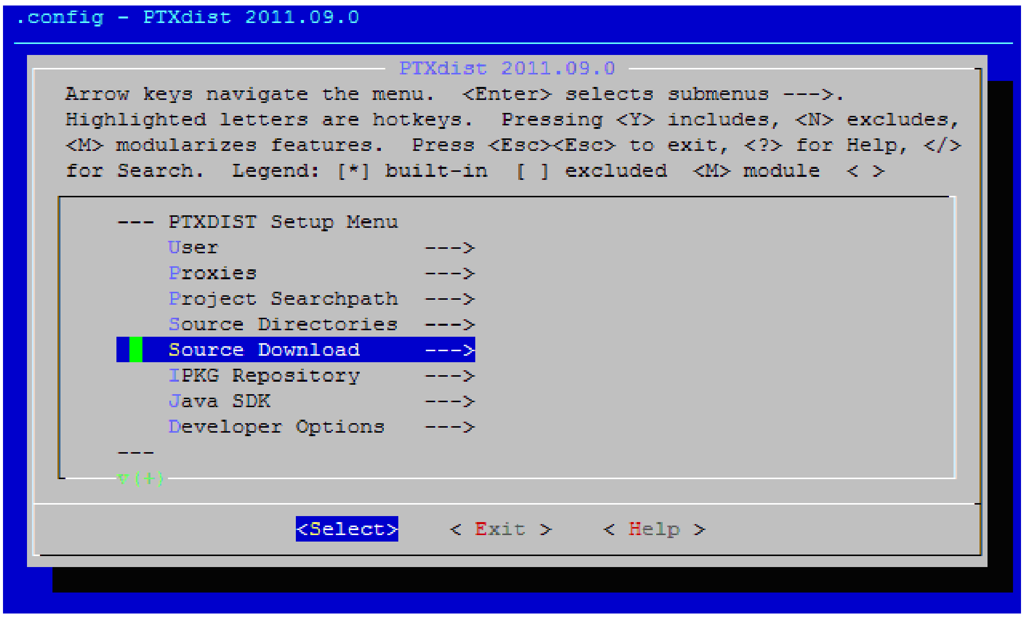


Figure 4: PTXDist setup menu

Select **Source Download** and set the **PTXDist Mirror** to **http://support.garz-fricke.com/mirror**, as shown in [▶ Figure 5]:

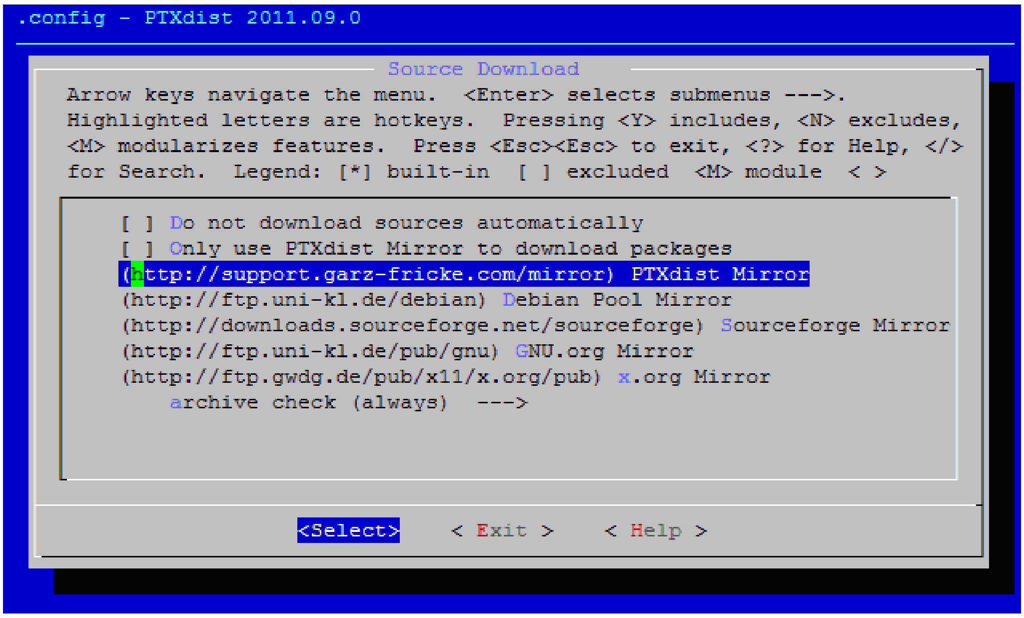


Figure 5: PTXDist mirror selection

6.3 Installing the GNU cross toolchain for the target architecure

When developing software for an embedded system, usually a cross toolchain is needed. A cross toolchain is a set of tools (e.g. the compiler gcc) which run on a host system of a specific architecture (such as x86) but produce binary code (executables) to run on a different architecture (e.g. ARM). Garz & Fricke systems use the **OSELAS.Toolchain** for cross compilation.

There are two possible ways to install the toolchain:

- Short way: Installing a pre-compiled toolchain
- Safe way: Building the toolchain with PTXdist

Note: For the short way, Garz & Fricke provides the necessary prebuilt toolchain packages on the Starter Kit's USB sticks and their FTP server. You can download and install them into your root file system. Please be aware of the fact that this may not work on all systems, since there may appear problems concerning user access rights. If the build process fails using a prebuilt toolchain, please remove it and go for the safe way by building the toolchain yourself.

There are different toolchains (and toolchain versions) needed depending on the platform the software will be built for. Because SANTARO is based on a i.MX6 MCU, the **arm-cortexa9-linux-gnueabi** toolchain is needed.

6.3.1 Installing a pre-compiled toolchain

Extract the pre-compiled toolchain for SANTARO from the CD / USB stick / FTP server on your host system as the user root:

```
$ sudo tar -xf OSELAS.Toolchain-2011.11.1-arm-cortexa9-linux-gnueabi.tar.bz2 -C /
```

The toolchain binary directory is now located at:

- **/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin**

6.4 Building the toolchain with PTXDist

PTXdist 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.BSP** package.

All OSELAS.Toolchain projects install their result into **/opt/OSELAS.Toolchain-2011.11.1/**. Usually the **/opt** directory is not world writeable. So in order to build our **OSELAS.Toolchain** into that directory we need to use a root account to change the permissions. PTXdist detects this case and asks if we want to run **sudo** to do the job for us. **sudo** is used to access the directories which are not world writable such as **/opt/**, **/etc/**, **/srv/** etc. If the user is already a root user we can enter:

```
$ mkdir /opt/OSELAS.Toolchain-2011.11.1
$ chmod a+w /opt/OSELAS.Toolchain-2011.11.1
```

Usually a normal user cannot access certain directories and **opt** is one of them. Hence, we change the access rights for all the users to write in order to make further changes.

It is recommended to keep this installation path as PTXdist expects the toolchains at **/opt**. Whenever we go to select a platform in a project, PTXdist tries to find the right toolchain from data read from the platform configuration settings and a toolchain at /opt that matches these settings. However, that happens for our convenience only. If we decide to install the toolchains at a different location, we still can use the toolchain parameter to define the toolchain to be used on a per-project base.

In order to compile and install the **OSELAS.Toolchain-2011.11.1-guf-0** we have to extract the **OSELAS.Toolchain-2011.11.1-guf-0.tar.bz2** archive, change into the new folder, configure the compiler and start the build. First, we copy the archive from the USB stick into our working directory:

```
$ cd ~/local
$ cp /<mountpoint of USB mass storage>/Tools/OSELAS.Toolchain-2011.11.1-guf-0.tar.bz2
```

The steps to build the toolcahin for SANTARO are as follows:

```
$ tar xf OSELAS.Toolchain-2011.11.1-guf-0.tar.bz2
$ cd OSELAS.Toolchain-2011.11.1
$ ptxdist select ptxconfigs/arm-cortexa9-linux-gnueabi_gcc-4.6.2_glibc-2.14.1
   ↪ _binutils-2.21.1a_kernel-2.6.39-sanitized.ptxconfig
$ ptxdist go
```

The build will take between 20 min and 2 hours depending on the host system.

After a successful build the toolchains bin directory is located at:

- **/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin**

It is strongly recommended that the new toolchain path is write-protected after the build. You can achieve this by typing:

```
$ chmod a-w /opt/OSELAS.Toolchain-2011.11.1
```

If the PTXdist version used to create the toolchain's configuration files does not match your installed PTXdist version, the followin error appears:

```
The ptxconfig file version and ptxdist version do not match:

        configfile version: 2011.11.0
        ptxdist version:    2011.09.0

        You can either migrate from an older ptxdist release with:
        'ptxdist migrate'

        or, to ignore this error, add '--force'
        to ptxdist's parameters, e.g.:
        'ptxdist --force go'
```

In this case you can either install the correct PTXdist version, or use

```
ptxdist migrate ptxconfigs/arm-cortexa9-linux-gnueabi_gcc-4.6.2_glibc-2.14.1_binutils
    ↪ -2.21.1a_kernel-2.6.39-sanitized.ptxconfig
```

to migrate the configuration file to your version of PTXdist.

6.5 Building the BSP for the target platform with PTXDist

The following steps describe the way of building a Linux BSP for the Garz & Fricke SANTARO platform. In this step the Linux kernel and the root file system are built.

The file **OSELAS-BSP-GUF-Linux-SANTARO-1.44.4-0.tar.bz2** must be copied from the CD / USB stick shipped with the starter kit to the host system. The file can be found in the subfolder **/BSP**:

```
$ cd ~/local
$ cp /<mountpoint of USB mass storage>/BSP/OSELAS.BSP-GUF-Linux-1.44.4-0.tar.bz2
```

In order to work with the PTXdist based project for SANTARO we have to extract the archive first:

```
$ tar -xf OSELAS.BSP-GUF-Linux-1.44.4-0.tar.bz2
$ cd OSELAS.BSP-GUF-Linux-1.44.4-0
```

The next step is to choose the hardware platform:

```
$ ptxdist platform configs/santaro/platformconfig
```

The toolchain is found automatically if it is installed in the default location:

```
found and using toolchain:
'/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/
gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin'
```

Otherwise you have to select the toolchain manually:

```
$ ptxdist toolchain /opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc
    ↪ -4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin
```

The next step is to select the configuration for the target's system configuration:

```
$ ptxdist select configs/santaro/ptxconfig
```

Now, the BSP can be built:

```
$ ptxdist go
```

This step compiles everything needed for the target system from source, including the kernel and the root file system. Depending on your host system and the target's system configuration, this will take between 0.5 and 3 hours.

At the end of the build process you will see the following output on the build console:

```
-------------------------------------------------------------------
For a proper NFS-root environment, some device nodes are essential.
In order to create them root privileges ere required.
-------------------------------------------------------------------

(Please press enter to start 'sudo' to gain root privileges.)

WARNING: NFS-root might not be working correctly!
```

At this point you have missed the creation of the essential device nodes for the target system. Device nodes can (and should always) be created only with root privileges. However, PTXdist can do that job by simply typing again:

```
$ ptxdist go
```

After a few seconds you have the possibility to create the device nodes again:

```
-----------------------------------------------------------------
For a proper NFS-root environment, some device nodes are essential.
In order to create them root privileges ere required.
-----------------------------------------------------------------

(Please press enter to start 'sudo' to gain root privileges.)
```

Press enter, enter the root password for your build PC and you get the following output:

```
Creating device node: platform-SANTARO/root/dev/null
Creating device node: platform-SANTARO/root/dev/zero
Creating device node: platform-SANTARO/root/dev/console
Creating device node: platform-SANTARO/root-debug/dev/null
Creating device node: platform-SANTARO/root-debug/dev/zero
Creating device node: platform-SANTARO/root-debug/dev/console
```

In order to create the image of the root file system which can be written to the target system, we have to type:

```
$ ptxdist images
```

The following table lists, which components have been built and where they are located now:

Component	Path
Kernel	OSELAS.BSP-GUF-Linux-1.44.4-0/platform-SANTARO/images/linuximage
Root file system	OSELAS.BSP-GUF-Linux-1.44.4-0/platform-SANTARO/images/root.tgz

7 Deploying the Linux system to the target

The deployment of the Linux system has to be separated into two cases:

- Development deployment
- Release deployment

7.1 Development deployment

It is common for embedded Linux developers to use a technique called **root over NFS** and load the Linux kernel from a TFTP server during the development phase. In this manner, the backing storage stays unused and many write cycles to the backing storage are saved.

7.1.1 Host configuration

In order to make the host ready for this technique there must be a TFTP server and an NFS server installed on the host system. Usually, the packages tftp and nfs-utils can be installed on every Linux distribution.

The configuration of the TFTP server has already been described in chapter [▶ 3.4 Uploading files with TFTP]. The NFS server must be configured as follows in the **/etc/exports** file on the host system in order to provide the directory **/rootfs** as an NFS share for the target with the IP address **192.168.1.1**:

```
/rootfs 192.168.1.1(rw, no_subtree_check, no_root_squash)
```

Server restart has to be done when changes are made:

```
$ service nfs-server restart
```

Most Linux distributions provide tools (e.g. YAST on SuSE) to perform those settings. Consult your Linux distribution documentation for further information.

When the host setup stated before is done successfully you can copy the kernel **linuximage** from **OSELAS.BSP-GUF-Linux-1.44.4-0/platform-SANTARO/images** to the TFTP directory **/tftpboot** and the contents of the root file system from **OSELAS.BSP-GUF-Linux-1.44.4-0/platform-SANTARO/root** to the NFS share **/rootfs**:

```
$ cp platform-SANTARO/images/linuximage /tftpboot/linuximage
$ rm -Rf /rootfs/*
$ cp -R platform-SANTARO/root/* /rootfs/
```

Now we are ready to connect the target to the host system. Make sure that the target is connected to the host with a null modem cable for a serial terminal connection and with an Ethernet connection for networking. Set up a serial connection as described in chapter [▶ 3.1 Serial console].

7.1.2 Target configuration

For this chapter, the host system is assumed to have the ip address **192.168.1.100**.

First, we have to configure the kernel to get the root file system over NFS. This is done in the kernel command line, which is stored in the Linux boot partition **/dev/mmcblk0p2** of the eMMC. In order to change it, we have to mount this partition:

```
root@SANTARO:~ mount /dev/mmcblk0p2 /mnt
```

Open the file **boot.cfg** in a text editor, e.g. **nano**:

```
root@SANTARO:~ nano /mnt/boot.cfg
```

The file should look similar to this:

```
load -b 0x12000000 -p config config.xml
load linuximage
exec "console=ttymxc0,115200 root=/dev/mmcblk0p3 xmlram=0x12000000"
```

Remove the **root=...** key and replace it with **root=/dev/nfs nfsroot=192.168.1.100:/rootfs**:

```
load -b 0x12000000 -p config config.xml
load linuximage
exec "console=ttymxc0,115200 root=/dev/nfs nfsroot=192.168.1.100:/rootfs xmlram=0
    ↪ x12000000"
```

Exit the editor by pressig Ctrl-X and save the file.

The second task is to load the new kernel via TFTP and store it on the eMMC. The kernel image is located on the same partition, so we can overwrite it directly using the tftp command:

```
root@SANTARO:~ tftp -g 192.168.1.100 -r linuximage -l /mnt/linuximage
```

Finally, unmount the boot partition and reboot the device:

```
root@SANTARO:~ cd ..
root@SANTARO:~ umount /mnt
root@SANTARO:~ reboot
```

The device should boot your new kernel using the root file system on your host machine. If the kernel crashes and the device cannot boot anymore, you can use the **Flash-N-Go** system to recover it. The Flash-N-Go system can be started by holding down the **Clar All** button while powering/resetting the device.

Please note that the kernel has to be stored on the eMMC in order to boot it, it cannot be executed directly from the RAM. To automate this process you can place the follwing shell script **tftp.sh** into a folder called **autostart** on an SD card or USB memory drive (see [▶ 4.1.13 Garz & Fricke Autostart] for a description of the autostart mechanism):

```
#!/bin/sh
cat /etc/hostname | grep FLASH-N-GO > /dev/null || exit 0
mkdir -p /mnt
mount /dev/mmcblk0p2 /mnt
tftp -g 192.168.1.100 -r linuximage -l /mnt/linuximage
reboot
```

If the storage media is plugged into the device, this script will automatically execute after the Flash-N-Go system has booted. It fetches the current kernel from the TFTP server, writes it to the eMMC and reboots the device. The second line of the script prevents it from being executed in your development Linux system.

7.2 Release deployment

Garz & Fricke SANTARO-0 is shipped with a RAM disk based Linux called **Flash-N-Go System** which is installed in parallel to the real operating system.

Flash-N-Go System is intended to use for service tasks e.g. operating system updates.

To update the real operating system connect **GND**, **RS232_TXD1** and **RS232_RXD1** of connector **X13** to a COM port of a PC and start a terminal program with the settings **115200 baud**, **8 bits**, **1 stop bit**, **no parity**, **handshake off**. The signals **TXD** and **RXD** have to be connected **cross-over** in the same way like a null modem cable does. The location of the X13 connector and the necessary pins can be found in figure [▶ Figure 6], [▶ Figure 7] and the following tabular:

Pin	Name	Description
1	GND	Ground
2	RS232_TXD1	Port#1: Transmit data (Output)
3	RS232_RXD1	Port#1: Receive data (Input)
4	RS232_RTS1	Port#1: Request-to-send (Output)
5	RS232_CTS1	Port#1: Clear-to-send (Input)
6	GND	Ground
7	RS232_TXD2	Port#2: Transmit data (Output)
8	RS232_RXD2	Port#2: Receive data (Input)
9	RS232_RTS2	Port#2: Request-to-send (Output)
10	RS232_CTS2	Port#2: Clear-to-send (Input)

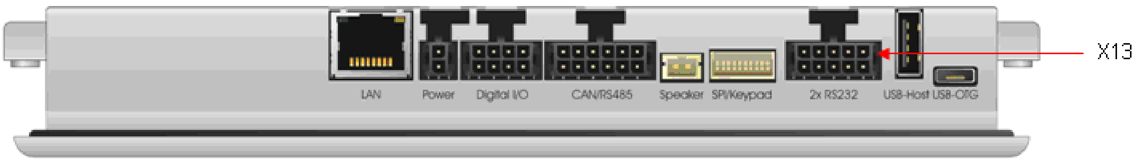


Figure 6: Location of the X13 connector

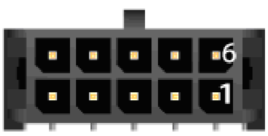


Figure 7: Pinning of the X13 connector

The Flash-N-Go System can be started by keeping SW2 pressed while supply power. The location of SW2 is shown in figure [▶ Figure 8].

Figure 8: Location of the SW2 switch

After a few seconds the command prompt of the Garz & Fricke Flash-N-Go System should appear in the terminal window:

```
---------------------------------
 Garz & Fricke Flash-N-Go System
---------------------------------

FLASH-N-GO:/
```

Intstall a TFTP-Server on a host PC and establish a Ethernet connection. The Ethernet connector can be found in [▶ Figure 9].

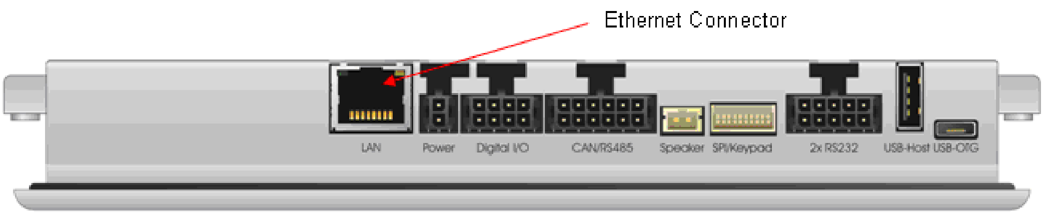


Figure 9: Location of the Ethernet connector

The Ethernet can be set up with the **sconfig** command line tool. The help can be shown by executing sconfig without parameters:

```
FLASH-N-GO:/ sconfig
Usage: /usr/bin/sconfig {start | setting [value]}
   Call without [value] to read a setting, call with [value] to write it.
```

```
Available settings:
   serialdiag  switch serial debug console on or off
   dhcp        switch DHCP on or off
   ip          set IP address
   mask        set subnet mask
   gateway     set standard network gateway
   mac         set MAC address
   name        set device name
   serial      set serial number (affects MAC address and device name)
   rotation    set display rotation
```

Example 1: Set IP Address 192.168.1.1 and netmask 255.255.255.0 and reboot the system to apply the network configuration:

```
FLASH-N-GO:/ sconfig ip 192.168.1.1
FLASH-N-GO:/ sconfig mask 255.255.255.0
FLASH-N-GO:/ reboot
```

Note: The SW2 has to be pressed to boot Flash-N-Go.

Example 2: Set DHCP and reboot the system to apply the network configuration:

```
FLASH-N-GO:/ sconfig dhcp on
FLASH-N-GO:/ reboot
```

Note: The SW2 has to be pressed to boot Flash-N-Go.

If the RS-232 and the network connections are established, the Linux kernel and the root file system from the build process can be installed. Garz & Fricke supplies the following bash script to accomplish this task:

```
#!/bin/sh -e

# This script is designed to run from the Flash-N-Go system v0.3 or higher.
# It partitions the device for Linux, loads Linux images over TFTP and
# writes them to the internal eMMC memory.
# To run the script, configure your TFTP server so that it points to the
# OSELAS-BSP-GUF-Linux folder, activate DHCP on your device and run the
# following line from Flash-N-Go (replace the IP address by your TFTP
# server's IP address):

# export TFTP=172.20.21.146; curl tftp://$TFTP/fng_linux_complete.sh > /tmp/a.sh; sh
    ↪ /tmp/a.sh

PLATFORM=SANTARO-0
LOCAL_PATH_PREFIX=images/

ROOTFS_TYPE=ext3

# Exit if not in FLASH-N-GO system
cat /etc/hostname | grep FLASH-N-GO > /dev/null || (
        echo "This script can only be run from FLASH-N-GO"
        exit 1
)

# Exit if not on an eMMC-based system
test -e /dev/mmcblk0boot1 || (
        echo "This script works for eMMC only"
        exit 1
)
```

```
# Partition sizes
KERNEL_SIZE=16

# Calculate offsets and sizes
EMMC_SIZE=$(parted /dev/mmcblk0 unit mib print | grep Disk |
                        awk '{print $3}' | awk -F "M" '{print $1}')
FLASH_N_GO_SIZE=$(parted /dev/mmcblk0 unit mib print | grep " 1 " |
                                        awk '{print $4}' | awk -F "." '{print $1}')
ROOTFS_SIZE=$(expr $EMMC_SIZE - $FLASH_N_GO_SIZE - $KERNEL_SIZE)

# Create partitions
sfdisk --force -uM /dev/mmcblk0 << EOF
,${FLASH_N_GO_SIZE},b
,${KERNEL_SIZE},b
,${ROOTFS_SIZE},83
EOF

# Wait until device nodes are there
sleep 2

# Format all Android partitions
mkfs.vfat -n LINUX -F 16      /dev/mmcblk0p2
mkfs.${ROOTFS_TYPE} -L ROOTFS           /dev/mmcblk0p3

# Write boot linuximage (kernel) and boot configuration
mkdir -p /tmp/emmc
mount /dev/mmcblk0p2 /tmp/emmc
echo "curl tftp://$TFTP/${LOCAL_PATH_PREFIX}linuximage${RELEASE_PLATFORM}${
    ↪ RELEASE_VERSION} > /tmp/emmc/linuximage"
curl tftp://$TFTP/${LOCAL_PATH_PREFIX}linuximage${RELEASE_PLATFORM}${RELEASE_VERSION}
    ↪  > /tmp/emmc/linuximage
echo "load -b 0x12000000 -p config config.xml" > /tmp/emmc/boot.cfg
echo "load linuximage" >> /tmp/emmc/boot.cfg
echo "exec \"console=ttymxc0,115200 root=/dev/mmcblk0p3 xmlram=0x12000000\"" >> /tmp/
    ↪ emmc/boot.cfg
umount /tmp/emmc
# Write rootfs partition
mount /dev/mmcblk0p3 -t ${ROOTFS_TYPE} /tmp/emmc
curl tftp://$TFTP/${LOCAL_PATH_PREFIX}root${RELEASE_PLATFORM}${RELEASE_VERSION}.tgz |
    ↪  tar -C /tmp/emmc -xz
umount /tmp/emmc
sync
rm -Rf /tmp/emmc

exit 0
```

To execute the installation, this script (**fng_linux_complete.sh**) and the images folder containing **linuximage** (the Linux kernel) and **rootfs.tgz** (the Linux root file system) of the OSELAS.BSP.Linux–1.44.4-0 must be placed in the root directory of your TFTP server.

Than, execute the following command from the Flash-N-Go command shell (replace <TFTP-Server IP> with the IP address of your TFTP server):

```
FLASH-N-GO:/ export TFTP=<TFTP-Server IP>; curl tftp://$TFTP/fng_linux_complete.sh >
    ↪ /tmp/a.sh; sh /tmp/a.sh
```

The installation procedure will take some minutes. The output of the installation procedure will appear on the terminal console. The installation procedure finishes by outputting the Flash-N-Go prompt again:

```
FLASH-N-GO:/
```

8 Building a user application for the target system

There are two types of user applications which will be covered in this chapter: Applications with a graphical user interface (GUI) and applications without a GUI. GUI applications are only supported on platforms shipped with a display. They can either be built manually using the cross toolchain, or integrated into the BSP using PTXdist as a build system. This leads to four different scenarios of building a user application:

- Non-GUI user application without PTXdist
- Non-GUI user application integrated into PTXdist
- Qt-based GUI user applications without PTXdist
- Qt-based GUI user applications integrated into PTXdist

The following sections describe how to build a simple **Hello Wold!** application for each of these options, if supported by the target system.

In addition to running native applications, the device can also be configured to display a website using Qt Webkit. The Garz & Fricke Linux BSP comes with a configurable web demo application, which is covered in a separate section in this chapter.

8.1 Non-GUI user application

The Non-GUI user applications described here will display the message **Hello World!** on the serial debug console. In order to see the output, the serial debug console has to be enabled and a null-modem cable has to be connected between the device's first serial port and your host system.

8.1.1 Non-GUI user application outside from PTXDist

Create a directory in your home directory on the host system and change to it:

```
$ cd ~
$ mkdir myapp
$ cd myapp
```

Create the empty files **main.cpp** and **Makefile** in this directory:

```
$ touch main.cpp Makefile
```

Edit the contents of the main.cpp file as follows:

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  cout << "Hello World!" << endl;
  return 0;
}
```

Edit the contents of the **Makefile** as follows:

```
CROSS_COMPILE=/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-
    ↪ glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized-sanitized/bin/arm-
    ↪ cortexa9-linux-gnueabi-

myapp: main.cpp
  $(CROSS_COMPILE)g++ -o $@ $<
  $(CROSS_COMPILE)strip $@

clean:
  rm -f myapp *.o *~ *.bak
```

If the toolchain is installed in the default directory, this example compiles for the target system by typing

```
$ make
```

in the **myapp** directory. Otherwise the **CROSS_COMPILE** variable must be set according to the toolchain installation.

After a successful build, the **maypp** executable is created in the **myapp** directory. You can transfer this application to the target system's **/usr/bin** directory using one of the ways described in chapter [▶ 3 Accessing the target system] and execute it from the device shell. It might be necessary to change the access rights of the executable first, so that all users are able to execute it.

8.1.2 Non-GUI user application integrated into PTXDist

In the BSP directory **OSELAS.BSP-GUF-Linux-1.44.4-0**, create a new **myapp** directory under **local_src/common** and change to it:

```
$ cd local_src/common
$ mkdir myapp
$ cd myapp
```

Create the files **main.cpp** and **Makefile.am** in this directory:

```
$ touch main.cpp Makefile.am
```

Edit the contents of the file **main.cpp** as follows:

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  cout << "Hello World!" << endl;
  return 0;
}
```

Edit the contents of the file **Makefile.am** as follows:

```
bin_PROGRAMS = \
  myapp

myapp_SOURCES = \
  main.cpp
```

Now run **autoscan** and rename the created file **configure.scan** to **configure.ac**:

```
$ autoscan
$ mv configure.scan configure.ac
```

Change the contents of the file **configure.ac** as follows (you should use your own email address instead of the stated one):

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.63])
AC_INIT([myapp], [1.0], [carsten.behling@garz-fricke.com])
AC_CONFIG_SRCDIR([main.cpp])
AC_CONFIG_HEADERS([config.h])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
```

```
# Checks for programs.
AC_PROG_CXX

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Now, run **autoreconf**:

```
$ autoreconf --install
```

Change to the BSP base directory and create a new PTXdist package (you should use your own email address instead of the stated one):

```
$ cd ../../..
$ ptxdist newpackage target
ptxdist: creating a new 'target' package:
ptxdist: enter packet name.......: myapp
ptxdist: enter version number....: trunk
ptxdist: enter URL of basedir....: file://$(PTXDIST_WORKSPACE)/local_src/common
ptxdist: enter suffix............:
ptxdist: enter packet author.....: Carsten Behling <carsten.behling@garz-fricke.com>
```

Change the contents of the created file **rules/myapp.in** as follows:

```
## SECTION=fixme

config MYAPP
  bool
  prompt "myapp"
  help
    The myapp application.
```

Change the contents of the created file **rules/myapp.make** as follows (**Note:** Use tab key for all indented lines, because it is a GNU makefile!):

```
# -*-makefile-*-
#
# Copyright (C) 2010 by Carsten Behling <carsten.behling@garz-fricke.com>
#
# See CREDITS for details about who has contributed to this project.
#
# For further information about the PTXdist project and license conditions
# see the README file.
#

#
# We provide this package
#
PACKAGES-$(PTXCONF_MYAPP) += myapp

#
# Paths and names
#

MYAPP_VERSION   := 1.0
```

```
MYAPP           := myapp
MYAPP_SUFFIX    :=
MYAPP_SRCDIR    := $(PTXDIST_WORKSPACE)/local_src/common/$(MYAPP)
MYAPP_DIR       := $(BUILDDIR)/$(MYAPP)
MYAPP_LICENSE   := unknown

# ----------------------------------------------------------------------------
# Get
# ----------------------------------------------------------------------------

$(MYAPP_SOURCE):
	@$(call targetinfo)
	@$(call get, MYAPP)

# ----------------------------------------------------------------------------
# Extract
# ----------------------------------------------------------------------------

$(STATEDIR)/myapp.extract:
	@$(call targetinfo)
	@$(call clean, $(MYAPP_DIR))
	@rm -Rf $(MYAPP_DIR)
	@cp -R $(MYAPP_SRCDIR) $(MYAPP_DIR)
	@$(call patchin, MYAPP)
	@$(call touch)

# ----------------------------------------------------------------------------
# Prepare
# ----------------------------------------------------------------------------

#MYAPP_CONF_ENV  := $(CROSS_ENV)

#
# autoconf
#
MYAPP_CONF_TOOL  := autoconf

#MYAPP_CONF_OPT  := $(CROSS_AUTOCONF_USR)  #$(STATEDIR)/myapp.prepare:
#	@$(call targetinfo)
#	@$(call clean, $(MYAPP_DIR)/config.cache)
#	cd $(MYAPP_DIR) && \
#		$(MYAPP_PATH) $(MYAPP_ENV) \
#		./configure $(MYAPP_CONF_OPT)
#	@$(call touch)

# ----------------------------------------------------------------------------
# Compile
# ----------------------------------------------------------------------------

#$(STATEDIR)/myapp.compile:
#	@$(call targetinfo)
#	@$(call world/compile, MYAPP)
#	@$(call touch)

# ----------------------------------------------------------------------------
# Install
# ----------------------------------------------------------------------------

#$(STATEDIR)/myapp.install:
#	@$(call targetinfo)
#	@$(call world/install, MYAPP)
#	@$(call touch)

# ----------------------------------------------------------------------------
# Target-Install
# ----------------------------------------------------------------------------
```

```
$(STATEDIR)/myapp.targetinstall:
	@$(call targetinfo)

	@$(call install_init, myapp)
	@$(call install_fixup, myapp,PRIORITY,optional)
	@$(call install_fixup, myapp,SECTION,base)
	@$(call install_fixup, myapp,AUTHOR,"Carsten Behling <carsten.behling@garz-
	    ↪ fricke.com>")
	@$(call install_fixup, myapp,DESCRIPTION,missing)

	@$(call install_copy, myapp, 0, 0, 0755, $(MYAPP_DIR)/myapp, /usr/bin/myapp)

	@$(call install_finish, myapp)

	@$(call touch)

# ----------------------------------------------------------------------------
# Clean
# ----------------------------------------------------------------------------

#$(STATEDIR)/myapp.clean:
#	@$(call targetinfo)
#	@$(call clean_pkg, MYAPP)

#	vim: syntax=make
```

Now, we are ready to build our newly created application. First, activate the integration of your package in PTXdist:

```
$ ptxdist menuconfig
```

Unselect the item **Garz & Fricke demo application** and select the item **myapp** as shown in [▶ Figure 10] (You can navigate between the items with the arrow keys and select/deselect an item with **[SPACE]**).

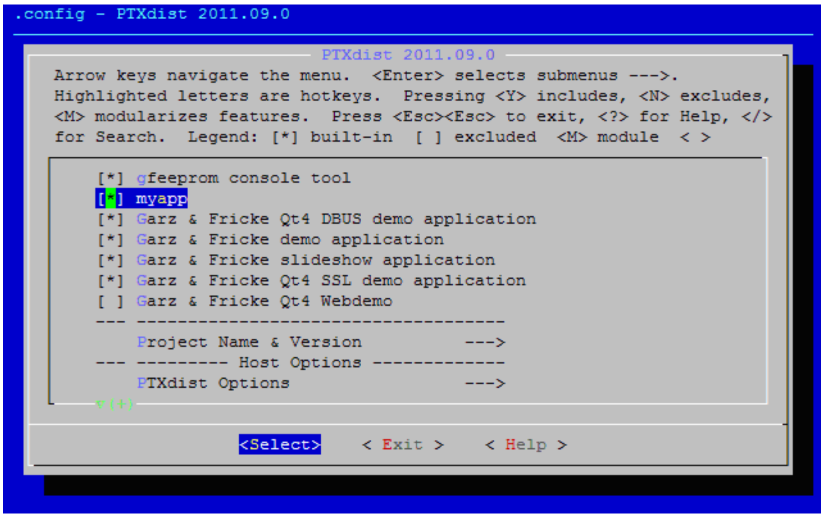


Figure 10: Selection of the created application in PTXDist

After selecting the application, exit and save the changes.

To rebuild the target system with the new application, simply run PTXdist in the BSP directory **OSELAS.BSP-GUF-Linux-1.44.4-0**. If you have already built the BSP before, only the newly created application package will be built:

```
$ ptxdist go
$ ptxdist images
```

Now, you can deploy the new target system like described in chapter [▶ 7 Deploying the Linux system to the target].

If you want to modify your application, e.g. change the **main.cpp** file, you can rebuild your application **maypp** by removing the state files for the **myapp** package and rebuild the system with PTXdist:

```
$ rm -f platform-SANTARO/state/myapp.*
$ ptxdist go
$ ptxdist images
```

Additional information of handling packages with PTXdist can be found in the PTXdist documentation from the CD/USB stick shipped with the starter kit or the Garz & Fricke FTP server in the Documentation folder (**OSELAS.BSP-Pengutronix-Generic-arm-Quickstart.pdf**).

8.1.3 Using the Eclipse IDE

Eclipse/CDT can be used as an IDE for C/C++ development for the target system. Apart from editing code, a cross toolchain can be involved with the **C/C++ Cross Compiler Support plugin** to build the software for the target platform. Additionally, with the **Remote System Explorer plugin** and the **C/C++ Remote Launch plugin** the software components can be transferred to the system using SFTP and executed remotely using SSH. Further, remote debug sessions can be executed by involving the cross gdb debugger on the host and the gdbserver on the target. Again, the handling of those tools is done automatically by Eclipse if configured properly.

To run Eclipse on the host PC a Java VM must be installed. Various JREs can be used to run Eclipse. However, the safest way is to use the original JRE from Sun Microsystems. The Sun JRE can be installed by extracting the tar archive from the Garz & Fricke USB memory stick located at **Tools/jre-7u5-linux-x64.tar.gz**:

```
$ cd /usr/lib/jvm
$ sudo tar -xf <USB memory stick mount point>/Tools/jre-7u5-linux-x64.tar.gz
```

This will install the Sun JRE to **/usr/lib/jvm/jre-1.7.0_05**.

Additionally, the Linux host system must be configured to use the JRE per default. On a Debian based system this can be done by using the commands:

```
$ sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jre1.7.0_05/bin/
    ↪ java 1
$ sudo update-alternatives --config java
```

If this is not the only JRE on the host system, a menu will appear that lists all possibilities for the java executable. Select the one with the path **/usr/lib/jvm/jre1.7.0_05/bin/java**.

Now, the following check must be done to verify the correct version:

```
$ java -version
Java version "1.7.0_05"
[...]
```

Next, Eclipse/CDT has to be installed on the host PC. Eclipse/CDT is installed by extracting the tar archive from the Garz & Fricke USB mass storage device located at

- **Tools/eclipse-cpp-juno-linux-gtk-x86_64.tar.gz**

into the users home directory on the host:

```
$ cd ~
$ tar -xf <USB memory stick mount point>/Tools/eclipse-cpp-juno-linux-gtk-x86_64.tar.
    ↪ gz
```

To extend Eclipse with the necessary plugins stated above, Eclipse must be started:

```
$ ~/eclipse/eclipse
```

After a while during the first start up of Eclipse, a dialog will appear as shown in [▶ Figure 11], that asks for the workspace location that is by default **/home/<user>/workspace**.

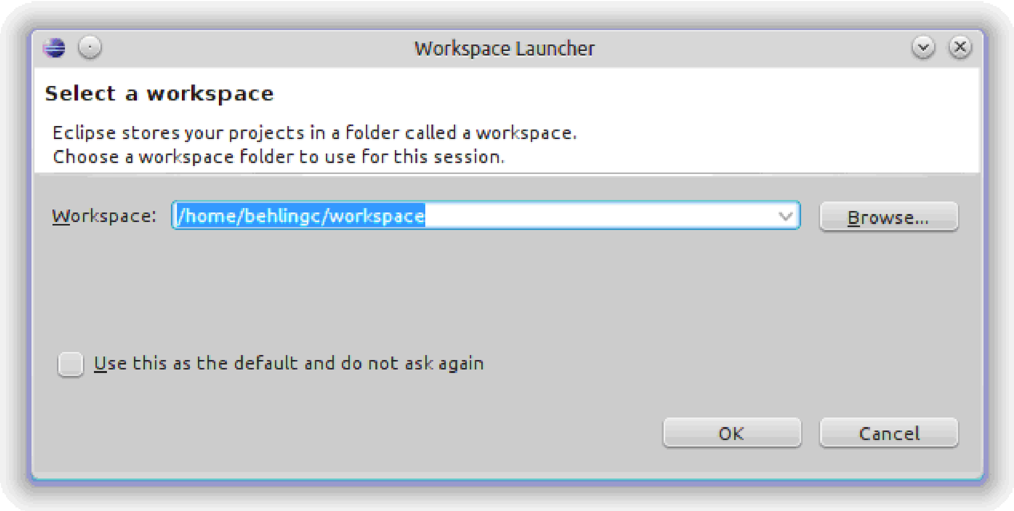


Figure 11: Eclipse workspace selection

The workspace path can be changed, if needed. The option **Use this as default and do not ask again** prevents the appearance of the screen at the next start of Eclipse.

By pressing the **OK** button the main screen as shown in [▶ Figure 12] appears.

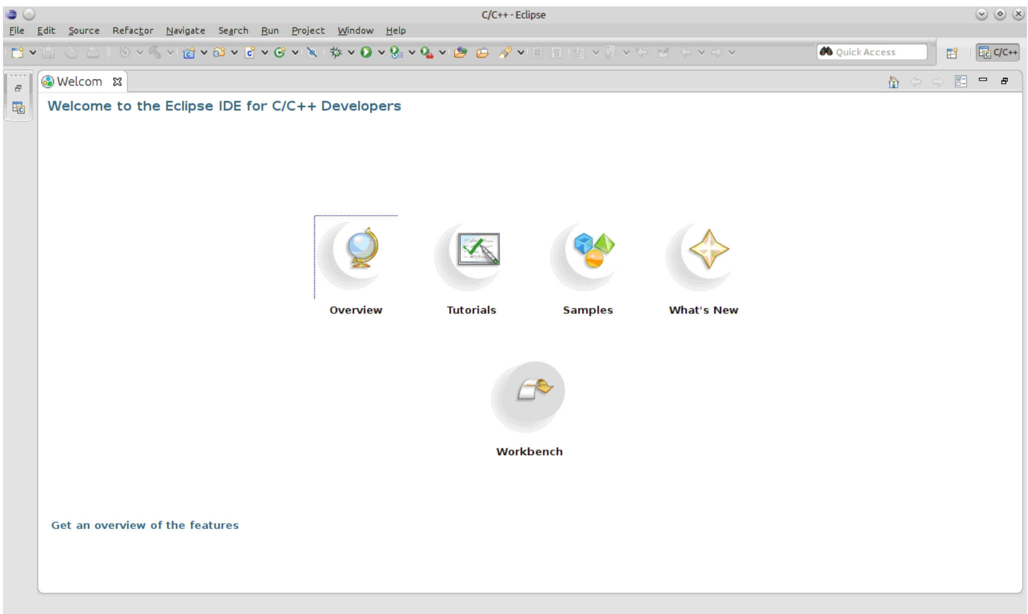


Figure 12: Eclipse main screen

The additional plugins can be installed by selecting **Help > Install New Software**. The plugin installation screen as shown in [▶ Figure 13] appears.

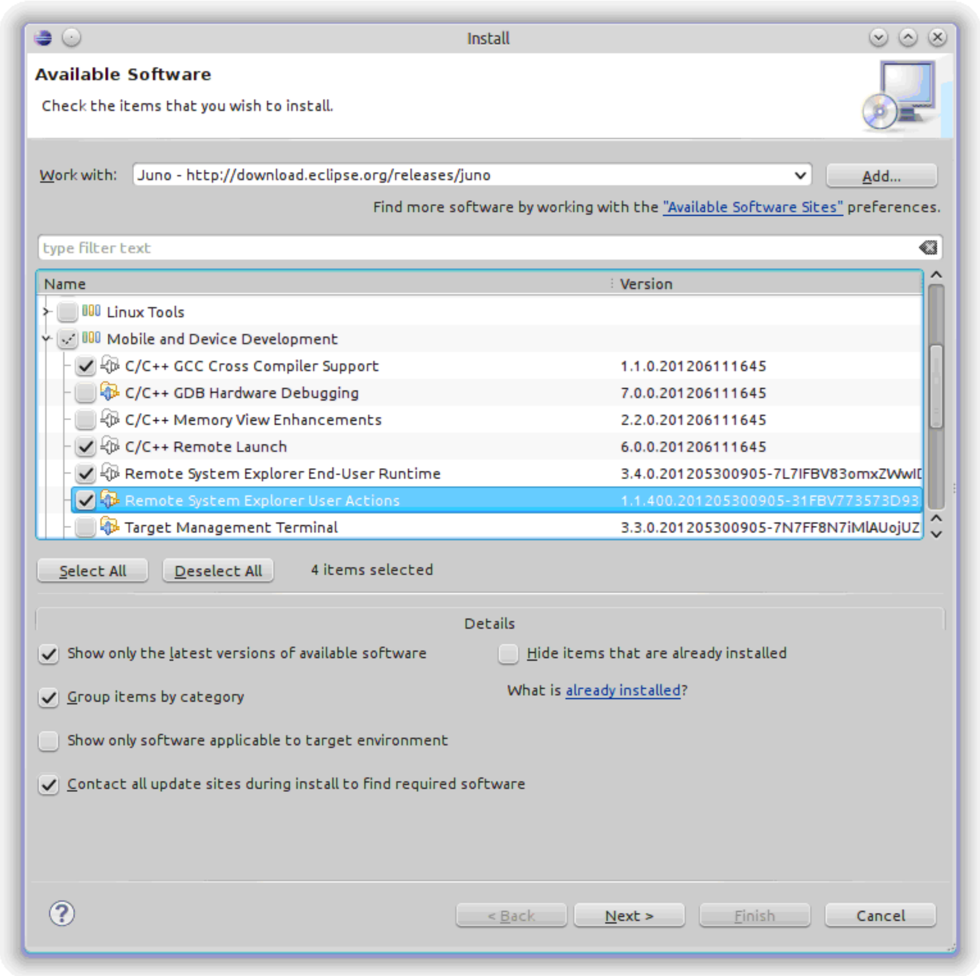


Figure 13: Eclipse plugin installation screen

To install the additional plugins, select the download site under **Work with:** and select the plugins like shown in [▶ Figure 13].

After pressing the button **Next >** the installation details are listed as shown in [▶ Figure 14].

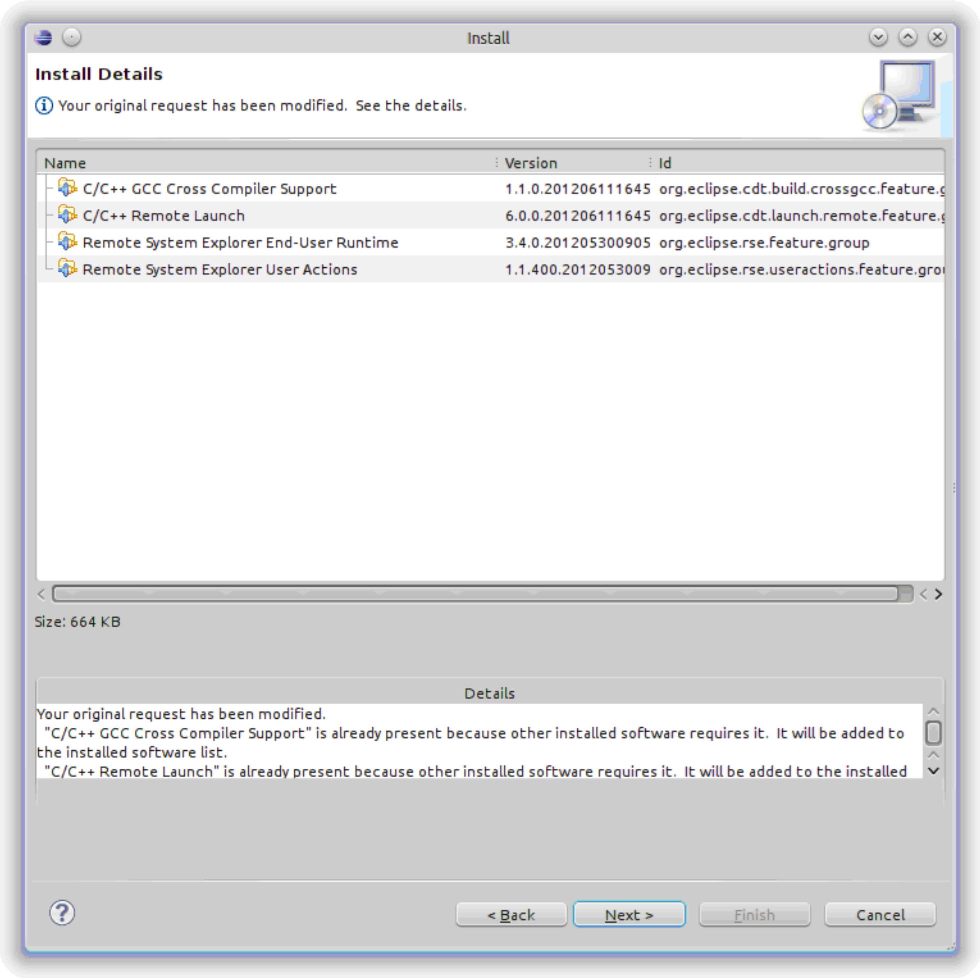


Figure 14: Eclipse plugin installation details screen

This is a simple verification of the previous steps. Again, press **Next >** no proceed. The license screen appears as shown in [▶ Figure 15].

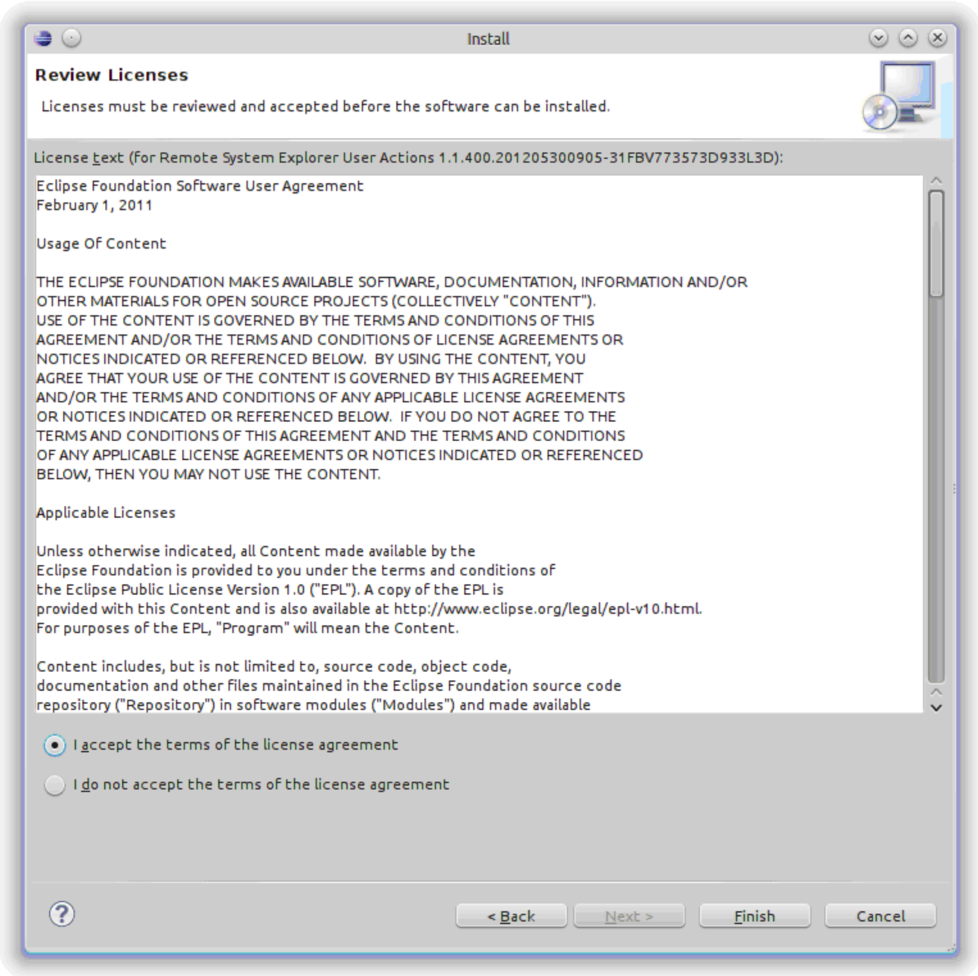


Figure 15: Eclipse license screen

Please accept the terms of the license agreement after reading and confirm with **Finish**. The installation process starts and a progress screen appears as shown in [▶ Figure 16].

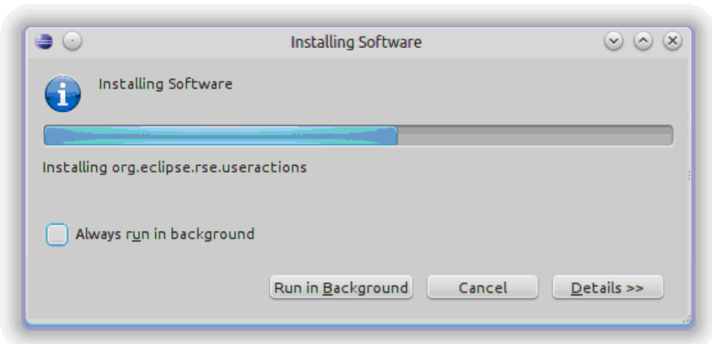


Figure 16: Eclipse plugin installation progress screen

At the end the user will be prompted to start Eclipse again to make the changes available as shown in [▶ Figure 17].

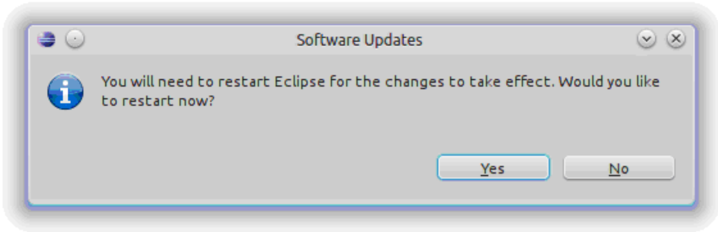


Figure 17: Eclipse restart request after plugin installation

Confirm this dialog with **Yes**. After restart, the plugins are available and the installation is done. For now Eclipse can be closed.

After succesfull installation, a simple **Hello World!** application can be created with the project wizard. This can be done by starting the project wizard with **File->New->C Project** as shown in [▶ Figure 18].

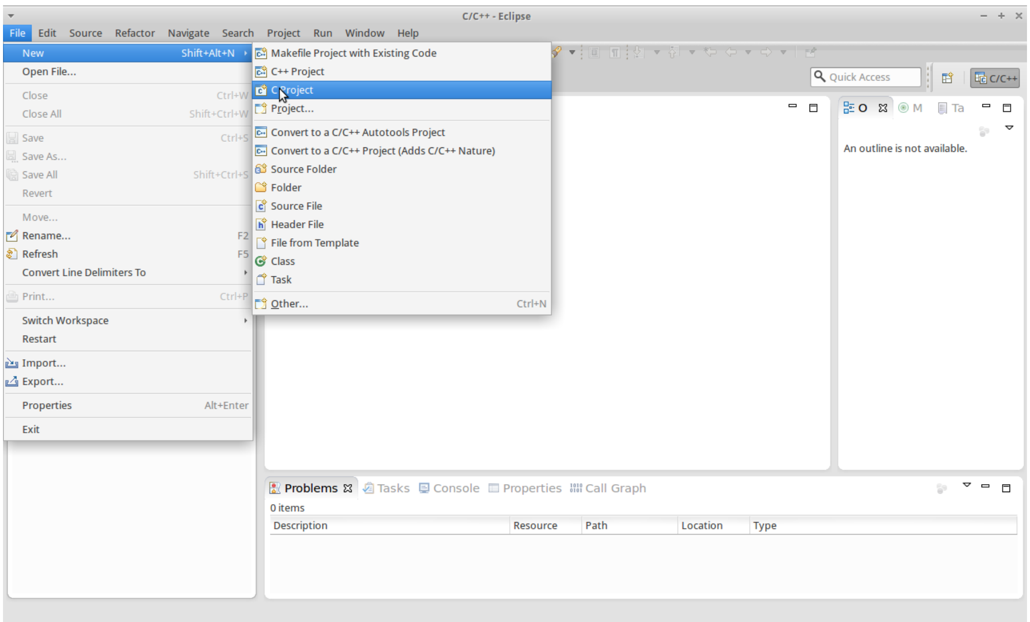


Figure 18: Creating a new project with Eclipse

In the first step of the wizard a template and project type can be selected and it can be choosen between a cross GCC or a native Linux GCC project. For this example **Executable->Hello Wold ANSI C Project** is used with the project name **myapp** and a **Cross GCC** toolchain as shown in [▶ Figure 19].

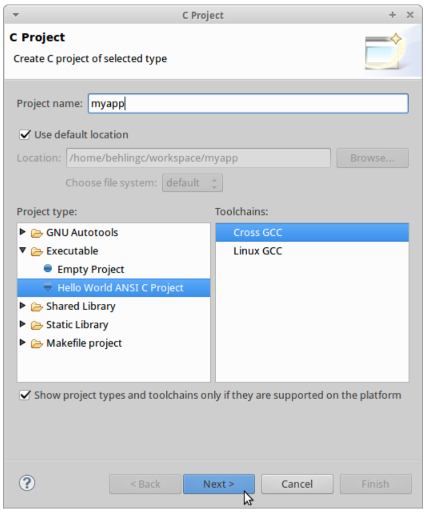


Figure 19: Selecting the project template

After pressing the **Next >** button, some basic project settings can be entered as shown in [▶ Figure 20].

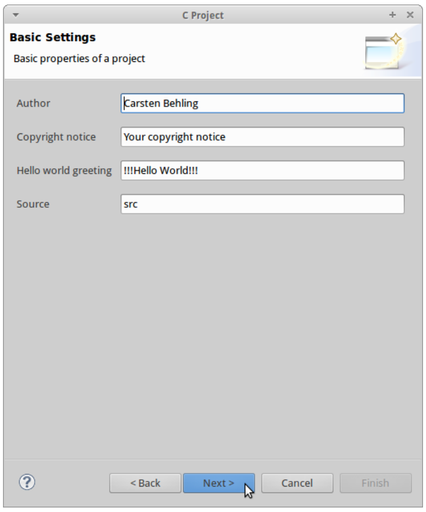


Figure 20: Project basic settings

After pressing the **Next >** button, the build configuartions can be chosen. For this example both selections, **Debug** and **Release** are chosen as shown in [▶ Figure 21].

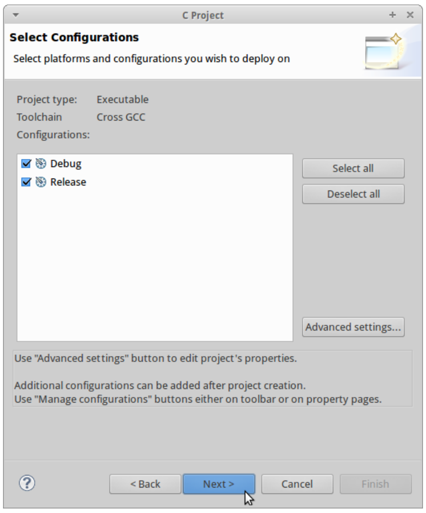


Figure 21: Selecting configurations

After pressing the **Next >** button, the toolchains **Cross compiler prefix** and the **Compiler path** must be set as shown in [▶ Figure 22].

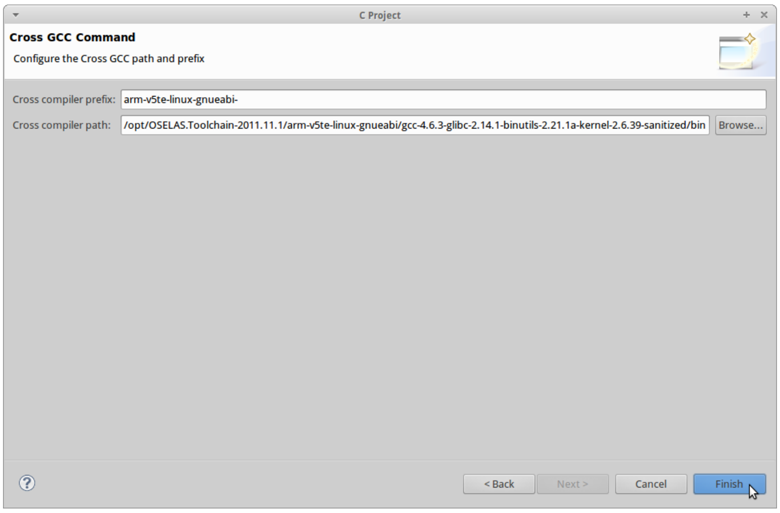


Figure 22: Selecting the cross toolchain

For SANTARO, the cross complie prefix is:

- **arm-cortexa9-linux-gnueabi-**

And the cross compiler path is:

- **/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-fsl-3.0.35-1.44.4-0-sanitized/bin**

After pressing the **Finish** button, the project is generated and can be accessed with the Eclipse IDE as shown in [▶ Figure 23].

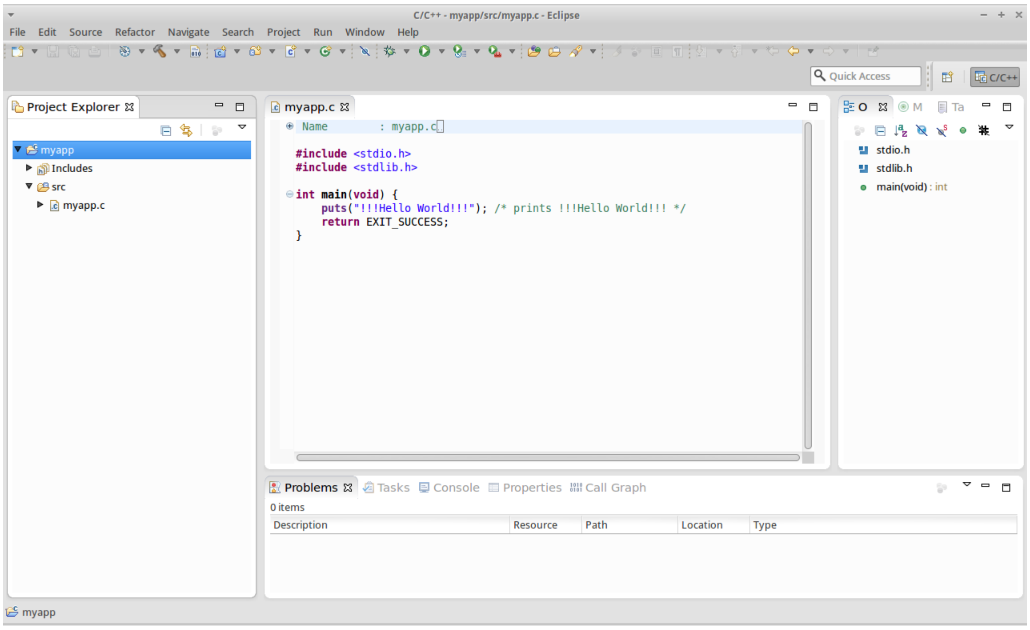


Figure 23: Project screen

The project can now be built by selecting **Build Project** in the context menu of **myapp** in the project explorer as shown in [▶ Figure 24].

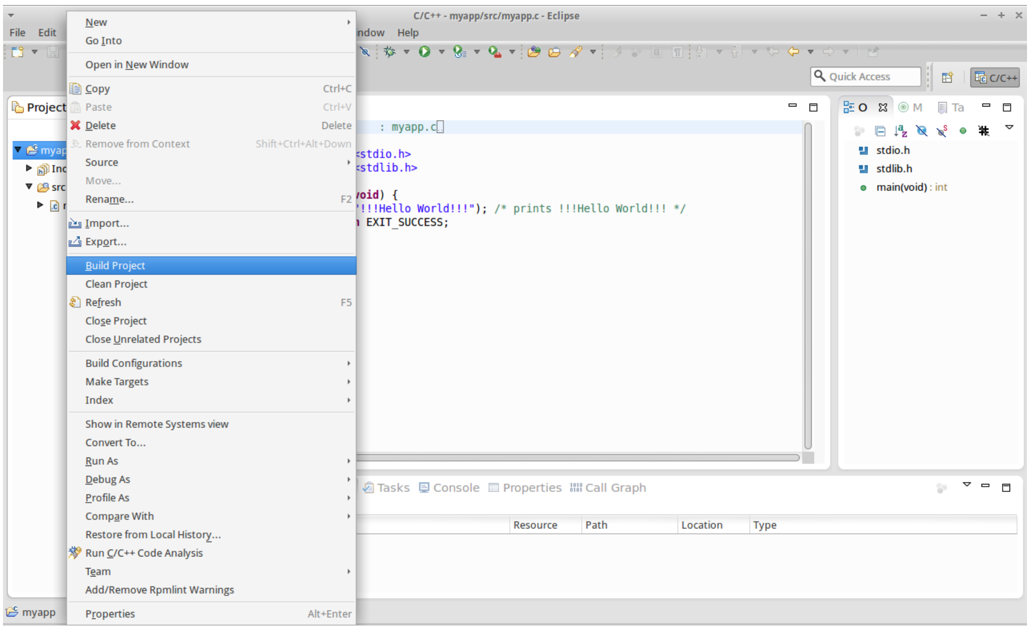


Figure 24: Building the project

After a successfull build, the myapp executable should appear in the project explorer as shown in [▶ Figure 25].

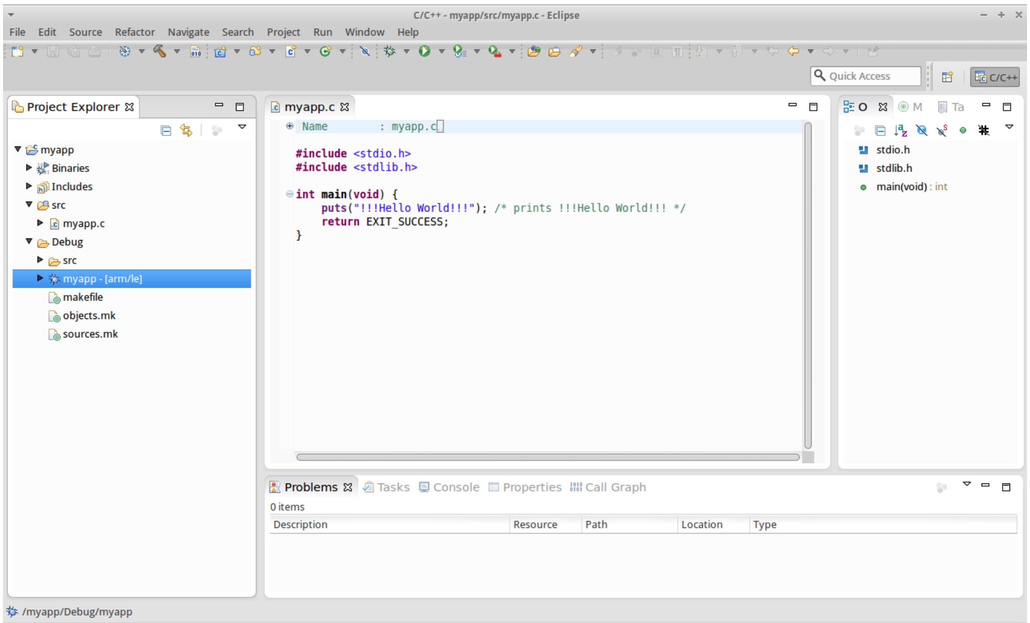


Figure 25: The executable in the project explorer

Because the debug configuration is active per default, the debug version of the executable is built. To switch to the release version as active configuration, **Build Configurations-> Set Active->Release** must be selected from the context menu of the myapp project as shown in [▶ Figure 26].

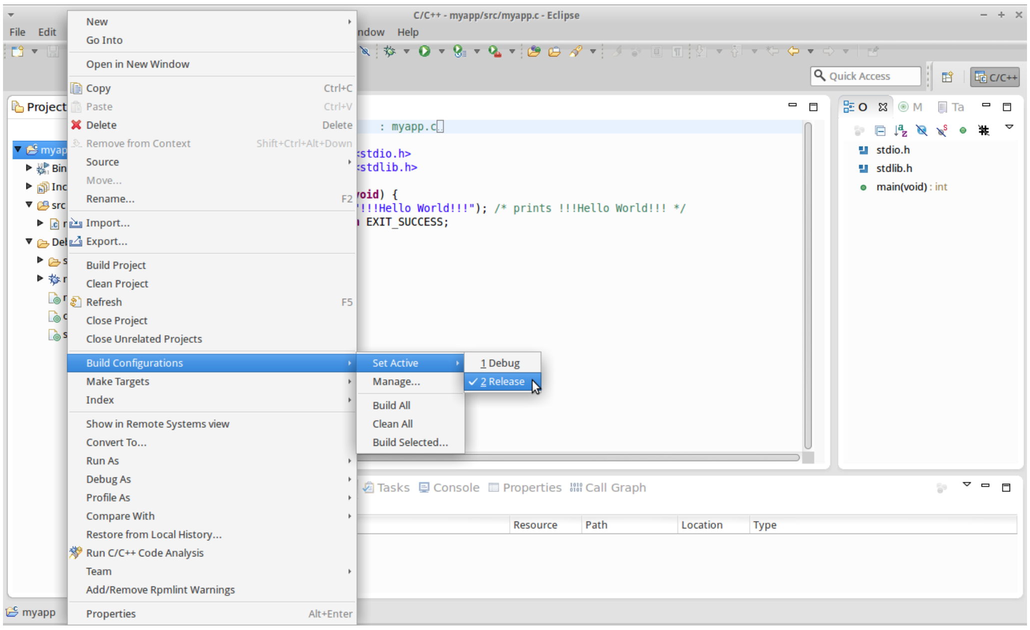


Figure 26: Selecting the release build configurartion

After building the project again, the release version should appear in the project explorer as shown in [▶ Figure 27].

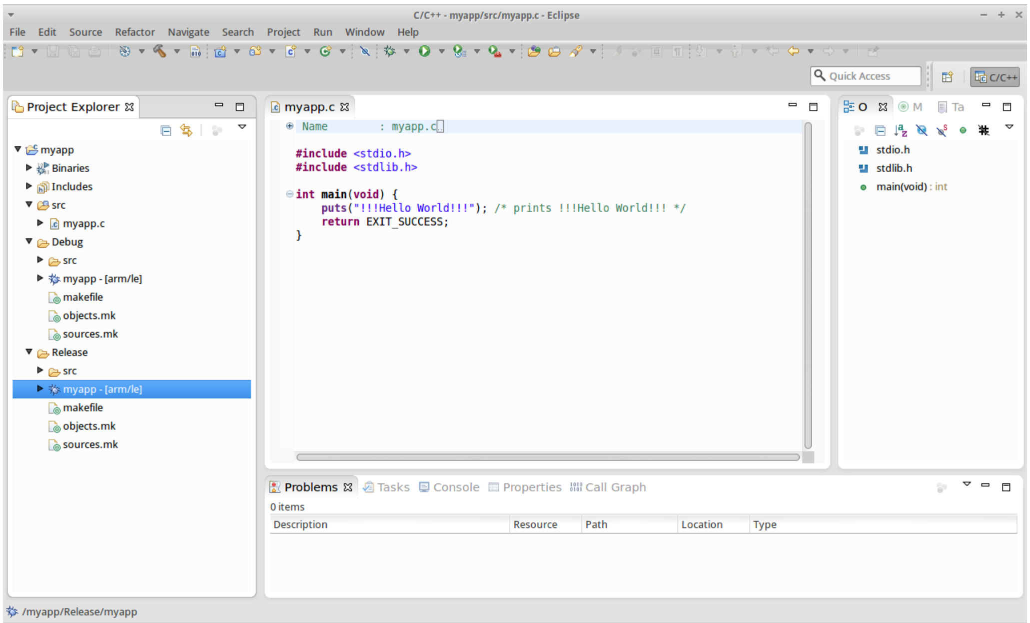


Figure 27: The release executable in the project explorer

The debug version of the executable can be found in the Eclipse workspace under:

- **myapp/Debug/src/myapp**

The release version of the executable can be found in the Eclipse workspace under:

- **myapp/Release/src/myapp**

To run the myapp application remotely on the target a remote system explorer connection to the target must be set up. To do this it is necessary to switch to the **Remote System Explorer Perspective** through **Window -> Open Perspective -> Other ...** as shown in [▶ Figure 28].

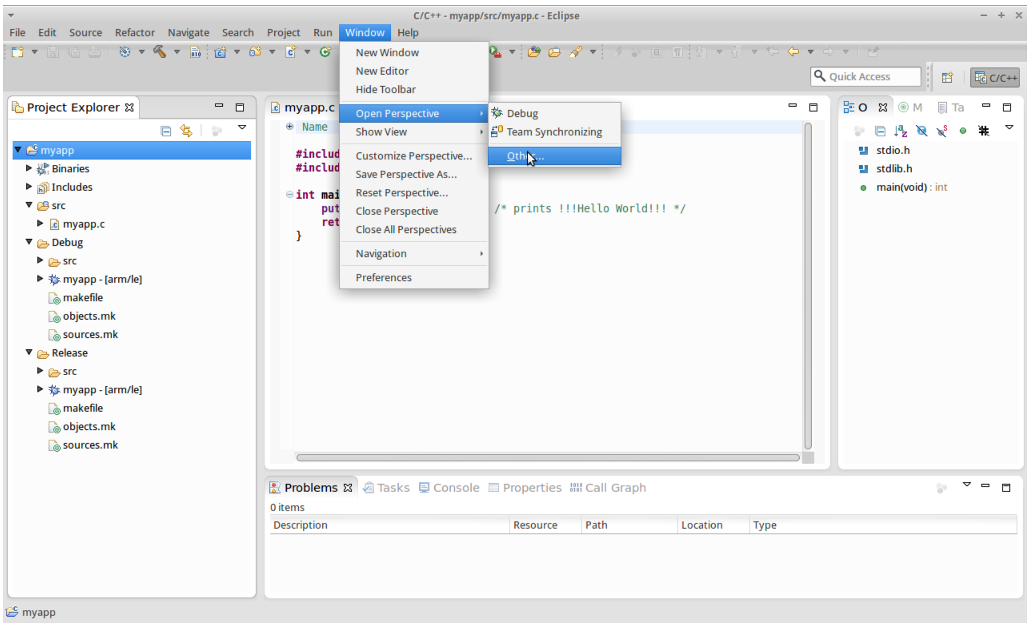


Figure 28: Changing the Eclipse perspective

In the following dialog, **Remote System Explorer Perspective** must be selected as shown in [▶ Figure 29].

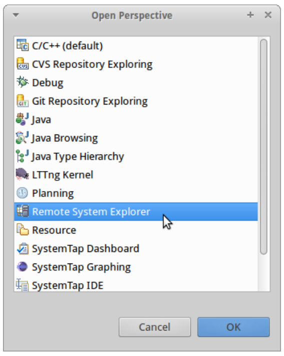


Figure 29: Changing to the remote system explorer perspective

In the remote system explorer perspective, a new connection can be created through **File -> New -> Other ...** as shown in [▶ Figure 30].

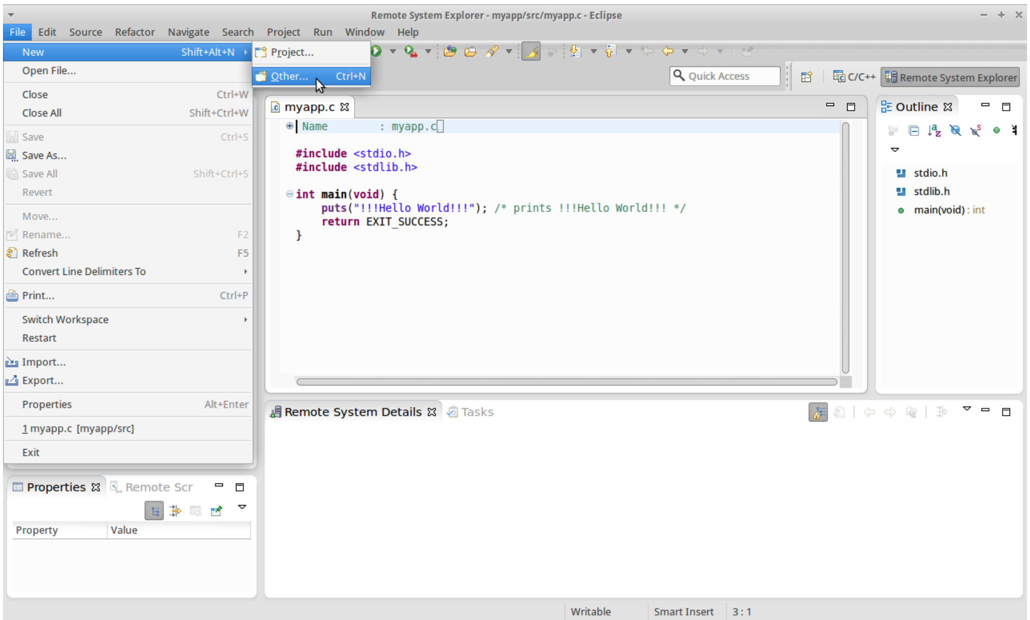


Figure 30: Creating a new connection

In the following dialog, the **Remote System Explorer -> Connection** must be selected as shown in [▶ Figure 31].

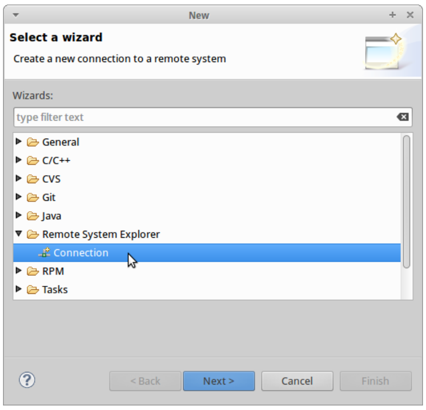


Figure 31: Creating a new Remote System Explorer connection

After pressing the **Next >** button, the remote system type **Linux** must be selected as shown in [▶ Figure 32].

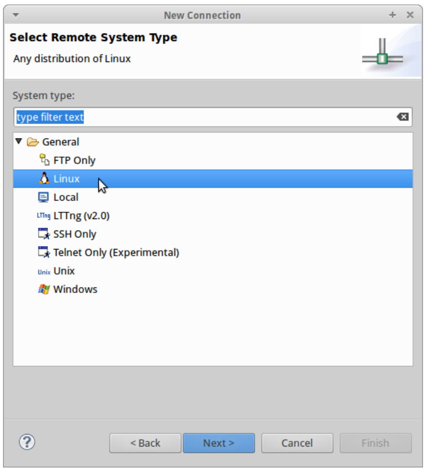


Figure 32: Selecting the connection system type

Assuming that the target has the default Garz & Fricke IP configuration **192.168.1.1/255.255.255.0** and the network connection between target and host is established on the target system, the settings as shown in [▶ Figure 33] can be used.

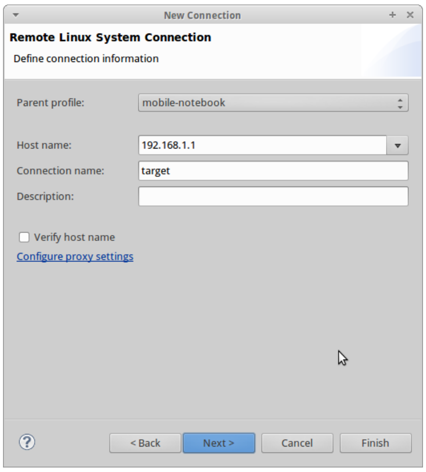


Figure 33: Setting up the network settings

After pressing the **Next >** button, **ssh.files** must be set up for file handling in the following dialog as shown in [▶ Figure 34].

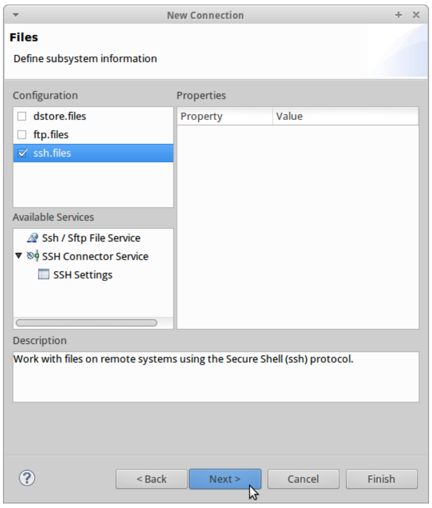


Figure 34: Setting up the file handling method

After pressing the **Next >** button, **processes.shell.linux** must be set up for process handling in the following dialog as shown in [▶ Figure 35].

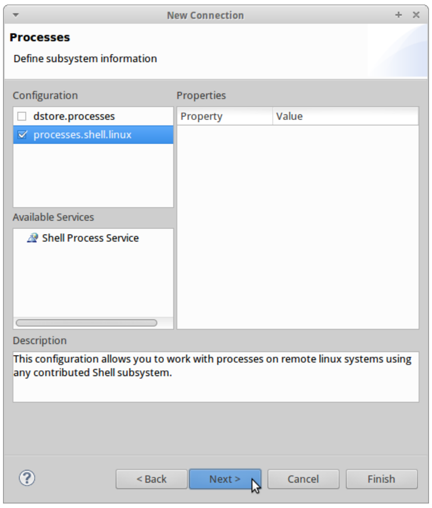


Figure 35: Setting up the process handling method

After pressing the Next > button, ssh.shells must be set up for shell handling in the following dialog as shown in [► Figure 36].

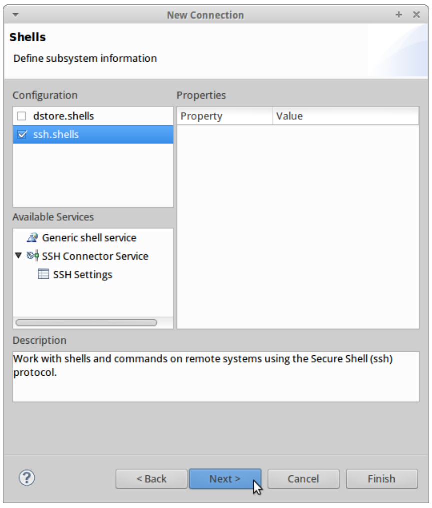


Figure 36: Setting up the shell handling method

After pressing the Next > button, ssh.terminals must be set up for terminal handling in the following dialog as shown in [► Figure 37].

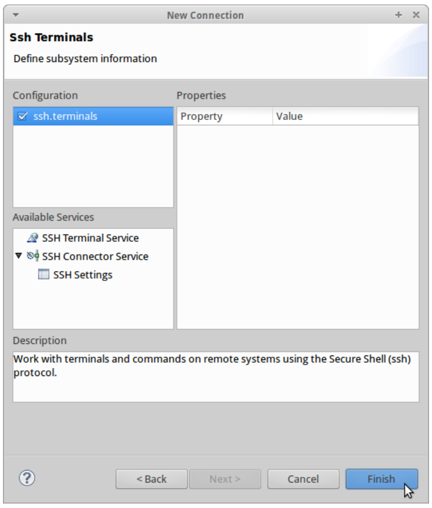


Figure 37: Setting up the terminal handling method

After pressing the **Next >** button, the new connection should appear in the remote system tab as shown in [▶ Figure 38].

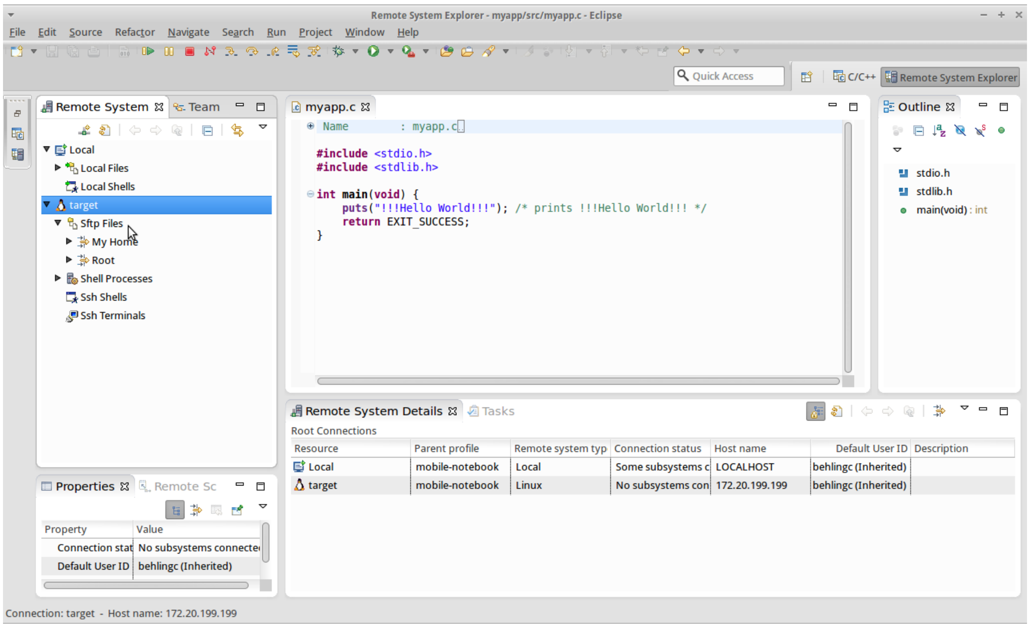


Figure 38: The new connection appears

If the new connection is set up properly, it should be possible to access the target's root file system remotely from the remote system explorer. This can be done by exploring the **Root** note within the new connection. The user will be asked for a **User ID:** and an optional **Password:** as shown in [▶ Figure 39].

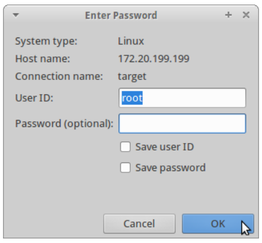


Figure 39: User ID and password request

If the login was successful the target's root file system should be explorable as shown in [▶ Figure 40].

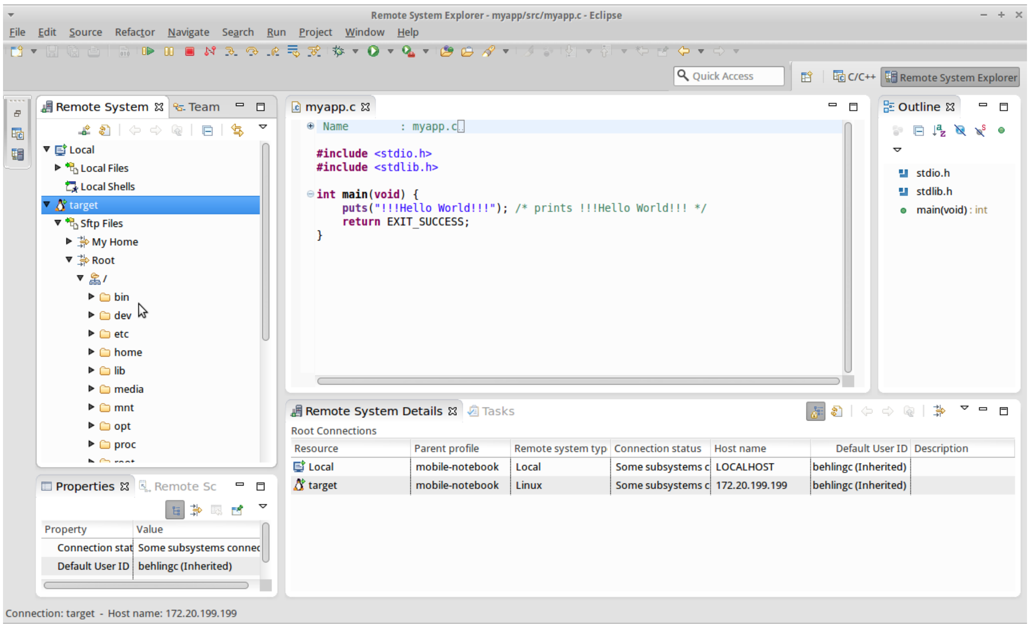


Figure 40: Exploring the target's root file system

Now, a run configuration must be set up. To do this it is necessary to switch back to the C/C++ perspective. Therefore, **Window -> Open Perspective -> Other ...** must be selected again as shown in [▶ Figure 41].

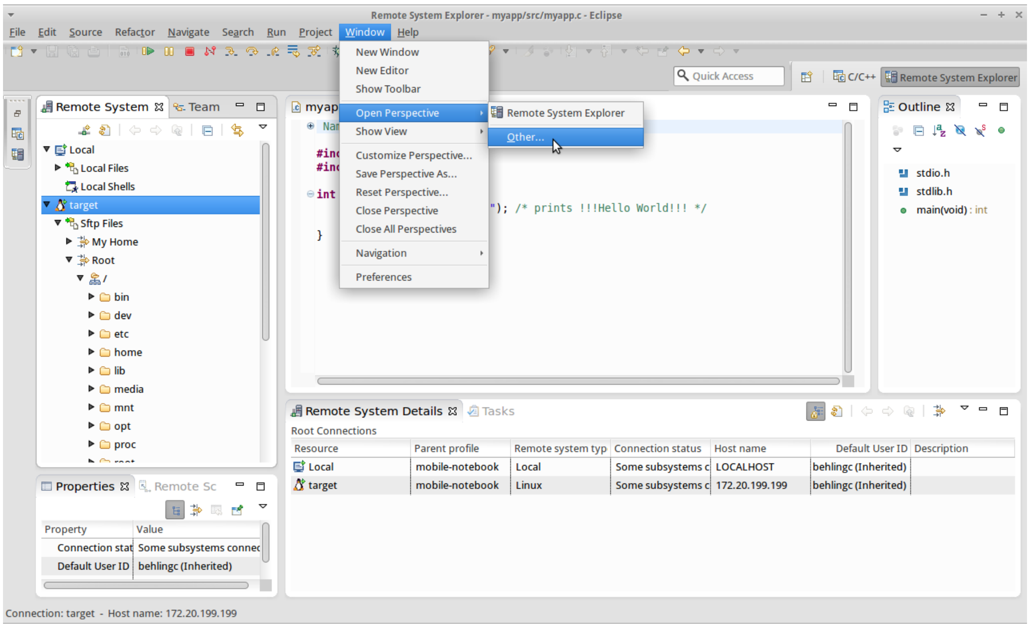


Figure 41: Switching again the Eclipse perspective

The C/C++ must be selected in the following dialog as shown in [▶ Figure 41].

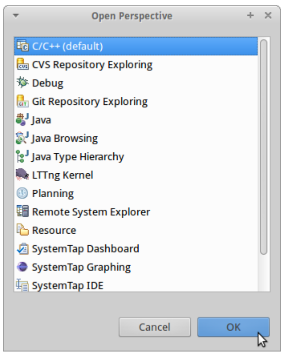


Figure 42: Selecting the C/C++ perspective

A new run configuration can be set up by selecting **Run -> Run Configurations** in the C/C++ perspective as shown in [▶ Figure 43].

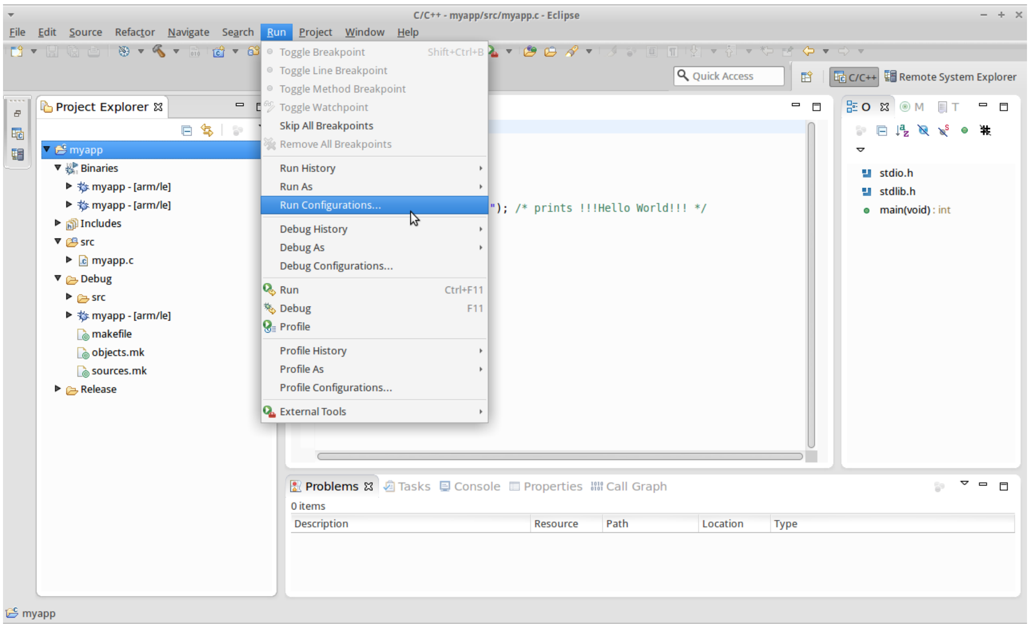


Figure 43: Creating a new run configuration

C/C++ Remote Application must be chosen as run configuration type, and a new launch configuration is added by clicking the Add symbol as shown in [▶ Figure 44].

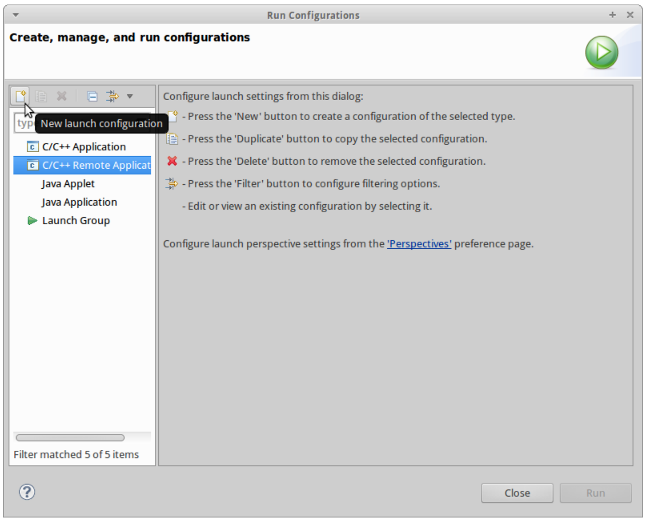


Figure 44: Adding a new launch configuration

To run the release version of myapp remotely in /usr/bin on the target system, the following dialog must be configured as shown in [▶ Figure 45].

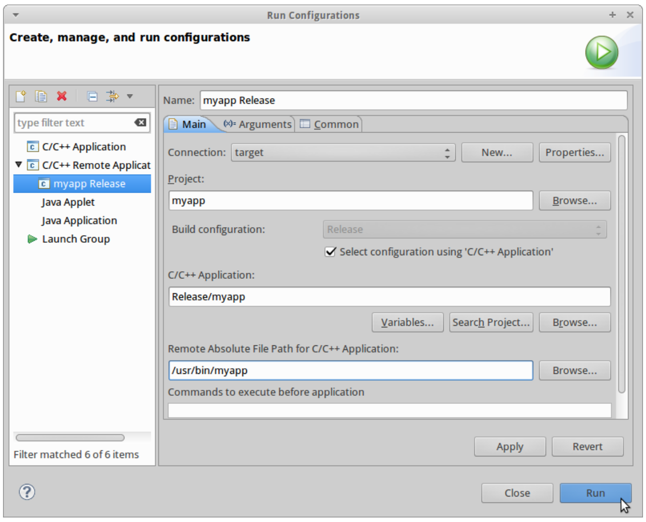


Figure 45: Run configuration settings

After pressing the **Run** button, myapp should be executed remotely on the target and a console output **!!!Hello World!!!** as shown in [▶ Figure 46] should appear.

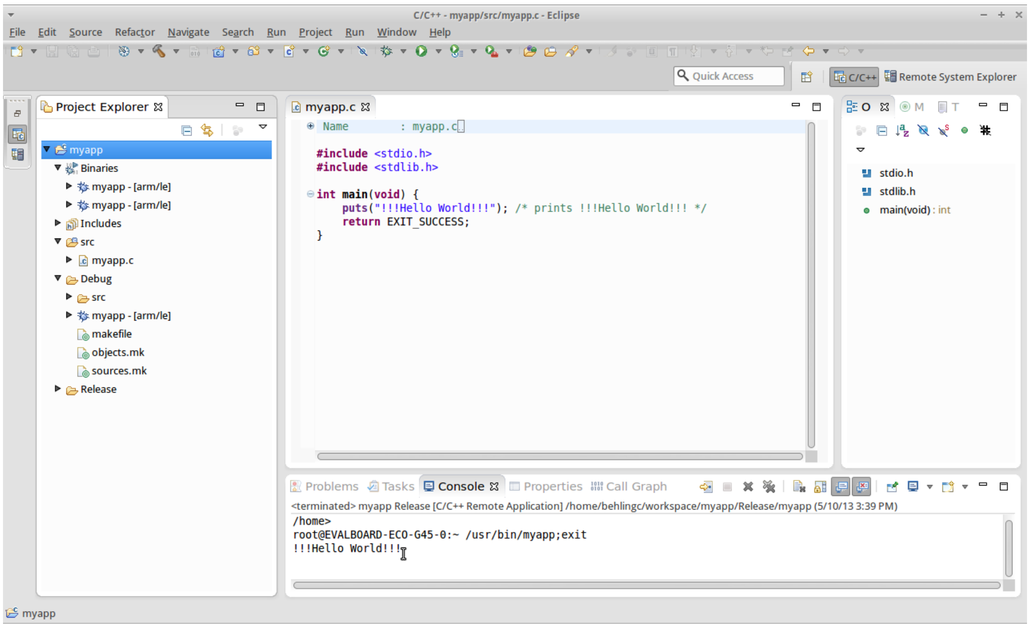


Figure 46: The console output of the remote application

8.2 Qt-based GUI user application

The GUI user applications described here will display the message **Hello World!** on the device's display.

8.2.1 Qt-based GUI user application outside from PTXDist

Create a new directory in your home directory on the host system and change to it:

```
$ cd ~
$ mkdir qt4-myapp
$ cd qt4-myapp
```

Create the empty files **main.cpp** and **build.sh** in this directory and make build.sh executable:

```
$ touch main.cpp build.sh myapp.pro
$ chmod a+x ./build.sh
```

Edit the contents of the file **main.cpp** as follows:

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);
        app.setOverrideCursor(Qt::BlankCursor);
        QPushButton hello("Hello World!");
        hello.setWindowFlags(Qt::FramelessWindowHint);
        hello.resize(800, 480);
        hello.show();
        return app.exec();
}
```

Edit the contents of the file **qt4-myapp.pro** as follows:

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

Edit the contents of the file **build.sh** as follows:

```
#!/bin/sh

TOOLCHAINDIR=/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-
    ↪ glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized-sanitized
BSP_DIR=~/OSELAS.BSP-GUF-Linux-1.44.4-0
BSP_PLATFORM=SANTARO
QT4_DIR=$BSP_DIR/platform-$BSP_PLATFORM/build-target/qt-everywhere-opensource-src
    ↪ -4.7.2-build
CROSS_DIR=$BSP_DIR/platform-$BSP_PLATFORM/sysroot-cross

export PATH=$TOOLCHAINDIR/bin:$CROSS_DIR/bin:/bin:/usr/bin

#
# Create Makefile by running qmake
#

QMAKEPATH=$BSP_DIR/platform-$BSP_PLATFORM/build-target/qt-everywhere-opensource-src
    ↪ -4.7.2 \
INSTALL_ROOT=$BSP_DIR/platform-$BSP_PLATFORM/sysroot-target \

QMAKESPEC=$BSP_DIR/platform-$BSP_PLATFORM/build-target/qt-everywhere-opensource-src
    ↪ -4.7.2/mkspecs/qws/linux-ptx-g++ \
qmake qt4-myapp.pro

#
# Run make
#
make

#
# Strip down the binary
```

```
#
arm-<arm-core>-linux-gnueabi-strip qt4-myapp

exit 0
```

Note: The above build script assumes that you have extracted the Source BSP in your home directory under ~/OSELAS.BSP-GUF-Linux-1.44.4-0. If the BSP is located in a different directory, you have to change the BSP_DIR variable accordingly.

Execute build.sh in the qt4-myapp directory:

```
$ ./build.sh
```

Now, there is the qt4-myapp executable in the qt4-myapp directory. You can transfer this application to the target system's /usr/bin directory in one of the ways described in chapter [▶ 3 Accessing the target system] and run it from the device shell.

8.2.2 Qt-based GUI user application integrated into PTXDist

Create a new directory in OSELAS.BSP-GUF-Linux-1.44.4-0/local_src on the host system and change to it:

```
$ cd local_src/common
$ mkdir qt4-myapp
$ cd qt4-myapp
```

Create the empty files main.cpp and qt4-myapp.pro in this directory:

```
$ touch main.cpp qt4-myapp.pro
```

Edit the contents of the file main.cpp as follows:

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);
        app.setOverrideCursor(Qt::BlankCursor);
        QPushButton hello("Hello World!");
        hello.setWindowFlags(Qt::FramelessWindowHint);
        hello.resize(800, 480);
        hello.show();
        return app.exec();
}
```

Edit the contents of the file qt4-myapp.pro as follows:

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

Change to the BSP base directory OSELAS.BSP-GUF-Linux-1.44.4-0 and create a new PTXdist package (you should use your own name and email address instead of the stated one):

```
$ cd ../../..
$ ptxdist newpackage target
ptxdist: creating a new 'target' package:
ptxdist: enter packet name.......: qt4-myapp
ptxdist: enter version number....: trunk
ptxdist: enter URL of basedir....: file://$(PTXDIST_WORKSPACE)/local_src/common
ptxdist: enter suffix............:
ptxdist: enter packet author.....: Carsten Behling <carsten.behling@garz-fricke.com>
ptxdist: enter package section...: project_specific
```

Change the contents of the newly created file **OSELAS.BSP-GUF-Linux-1.44.4-0/rules/myapp.in** as follows:

```
## SECTION=fixme

config QT4_MYAPP
  bool
  select QT4
  prompt "qt4-myapp"
  help
    The qt4-myapp application.
```

Change the contents of the newly created file **OSELAS.BSP-GUF-Linux-1.44.4-0/rules/myapp.make** as follows (**Note:** Use tab key for all indented lines, because it is a GNU makefile!):

```
# -*-makefile-*-
#
# Copyright (C) 2010 by Carsten Behling <carsten.behling@garz-fricke.com>
#
# See CREDITS for details about who has contributed to this project.
#
# For further information about the PTXdist project and license conditions
# see the README file.
#

#
# We provide this package
#
PACKAGES-$(PTXCONF_QT4_MYAPP) += qt4-myapp

# ----------------------------------------------------------------------------
# Paths and names
# ----------------------------------------------------------------------------

QT4_MYAPP_VERSION   := 1.0
QT4_MYAPP           := qt4-myapp
QT4_MYAPP_SUFFIX    :=
QT4_MYAPP_SRCDIR    := $(PTXDIST_WORKSPACE)/local_src/common/$(QT4_MYAPP)
QT4_MYAPP_DIR       := $(BUILDDIR)/$(QT4_MYAPP)
QT4_MYAPP_LICENSE   := unknown

# ------------------------------------------------------------------------------
# Extract
# ------------------------------------------------------------------------------

$(STATEDIR)/qt4-myapp.extract:
	@$(call targetinfo)
	@$(call clean, $(QT4_MYAPP_DIR))
	@rm -Rf $(QT4_MYAPP_DIR)
	@cp -R $(QT4_MYAPP_SRCDIR) $(QT4_MYAPP_DIR)
	@$(call patchin,QT4_MYAPP)
	@$(call touch)

# ------------------------------------------------------------------------------
# Get
```

```
# ----------------------------------------------------------------------------

$(QT4_MYAPP_SOURCE):
	@$(call targetinfo)
	@$(call get, QT4_MYAPP)

# ----------------------------------------------------------------------------
# Prepare
# ----------------------------------------------------------------------------

QT4_MYAPP_PATH  := PATH=$(CROSS_PATH)


QT4_MYAPP_ENV = \
	$(CROSS_ENV) \
	QMAKEPATH=$(QT4_DIR) \
	INSTALL_ROOT=$(SYSROOT) \
	QMAKESPEC=$(QT4_DIR)/mkspecks/qws/linux-ptx-g++

$(STATEDIR)/qt4-myapp.prepare:
	@$(call targetinfo)
	cd $(QT4_MYAPP_DIR) && \
		$(QT4_MYAPP_PATH) $(QT4_MYAPP_ENV) qmake qt4-myapp.pro
	@$(call touch)

# ----------------------------------------------------------------------------
# Compile
# ----------------------------------------------------------------------------

$(STATEDIR)/qt4-myapp.compile:
@$(call targetinfo)
cd $(QT4_MYAPP_DIR) && $(QT4_MYAPP_PATH) $(MAKE) $(PARALLELMFLAGS)
@$(call touch)


# ----------------------------------------------------------------------------
# Install
# ----------------------------------------------------------------------------

#$(STATEDIR)/qt4-myapp.install:
#	@$(call targetinfo)
#	@$(call world/install, QT4_MYAPP)
#	@$(call touch)

# ----------------------------------------------------------------------------
# Target-Install
# ----------------------------------------------------------------------------

$(STATEDIR)/qt4-myapp.targetinstall:
@$(call targetinfo)

	@$(call install_init, qt4-myapp)
	@$(call install_fixup, qt4-myapp,PRIORITY,optional)
	@$(call install_fixup, qt4-myapp,SECTION,base)
	@$(call install_fixup, qt4-myapp,AUTHOR,"Carsten Behling <carsten.
	    ↪ behling@garz-fricke.com>")
	@$(call install_fixup, qt4-myapp,DESCRIPTION,missing)

	@$(call install_copy, qt4-myapp, 0, 0, 0755, $(QT4_MYAPP_DIR)/qt4-myapp,
     /usr/bin/qt4-myapp)

	@$(call install_finish, qt4-myapp)

	@$(call touch)

# ----------------------------------------------------------------------------
```

```
# Clean
# ----------------------------------------------------------------------------

#$(STATEDIR)/qt4-myapp.clean:
#       @$(call targetinfo)
#       @$(call clean_pkg, QT4_MYAPP)

# vim: syntax=make
```

Open the PTXdist project configuration menu:

```
$ ptxdist menuconfig
```

Unselect the item Garz & Fricke demo application and select the item **qt4-myapp** as shown in [▶ Figure 47] (you can navigate between the items with the arrow keys and select/deselect an item with **[SPACE]**).

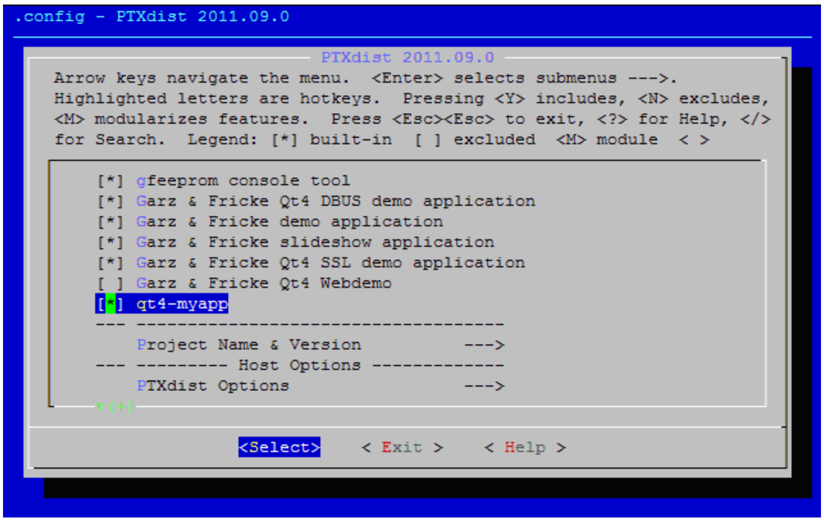


Figure 47: Selection of the created Qt application in PTXDist

To rebuild the target system with the new application, simply run PTXdist in the BSP directory **OSELAS.BSP-GUF-Linux-1.44.4-0**. If you have already built the BSP before, only the newly created application package will be built:

```
$ ptxdist go
$ ptxdist images
```

Now you can deploy the new target system like described in chapter [▶ 7 Deploying the Linux system to the target].

If you want to modify your application, e.g. change the **main.cpp** file, you can rebuild your application **qt4-maypp** by removing the state files for the **qt4-myapp** package and rebuild the system with PTXdist:

```
$ rm -f platform-SANTARO-0/state/qt4-myapp.*
$ ptxdist go
$ ptxdist images
```

Additional information of handling packages with PTXdist can be found in the PTXdist documentation from the CD/USB stick shipped with the starter kit or the Garz & Fricke FTP server in the Documentation folder (**OSELAS.BSP-Pengutronix-Generic-arm-Quickstart.pdf**).

8.2.3 Using the Qt Creator IDE

Qt Creator can be used as an IDE for Qt development with C++ for the target system. Its features make the application development for Qt more comfortable. Qt Creator allows the development and the automatic deployment of the Qt application controlled by its IDE.

To use Qt with the cross toolchain shipped with the Garz & Fricke BSP, the Qt version must be set up properly. Furthermore, the device configuration for automatic deployment must be set up properly.

To install Qt Creator the following installer from the CD / USB stick shipped with the starter kit for the SANTARO is required:

- **Tools/qt-creator-x86_64-opensource-2.5.2.bin**

It must be ensured that the installer has execution permissions:

```
$ chmod a+w ./qt-creator-x86_64-opensource-2.5.2.bin
```

Now, the installer can be executed from the Linux shell:

```
$ ./qt-creator-x86_64-opensource-2.5.2.bin
```

After executing the installer, the welcome dialog appears as shown in figure [▶ Figure 48].

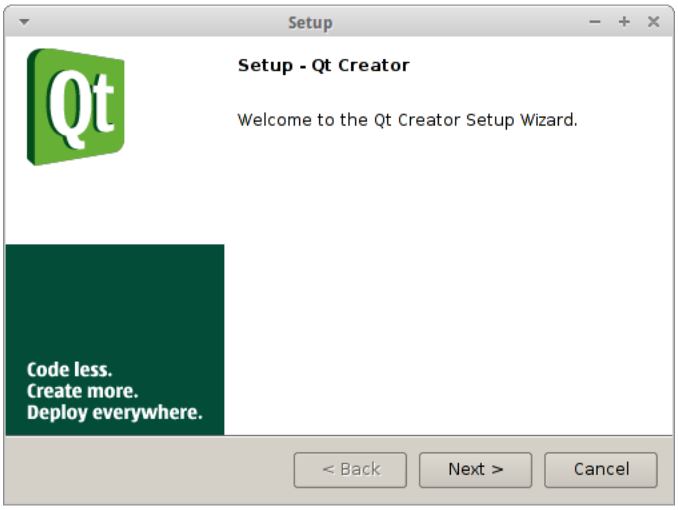


Figure 48: Welcome dialog of the Qt Creator setup

Pressing the **Next >** button leads to the license agreement dialog as shown in [▶ Figure 49].

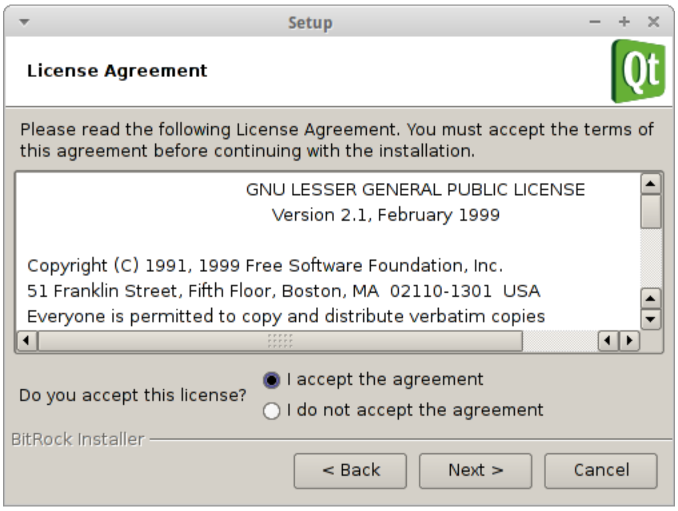


Figure 49: Qt Creator License agreement confirmation

The license agreement can be accepted by choosing the **I accept the agreement** option and pressing the **Next >** button again. In the following dialog as shown in [▶ Figure 50], the installation path of Qt Creator can be chosen. Per default Qt Creator will be installed into the user's home directory.

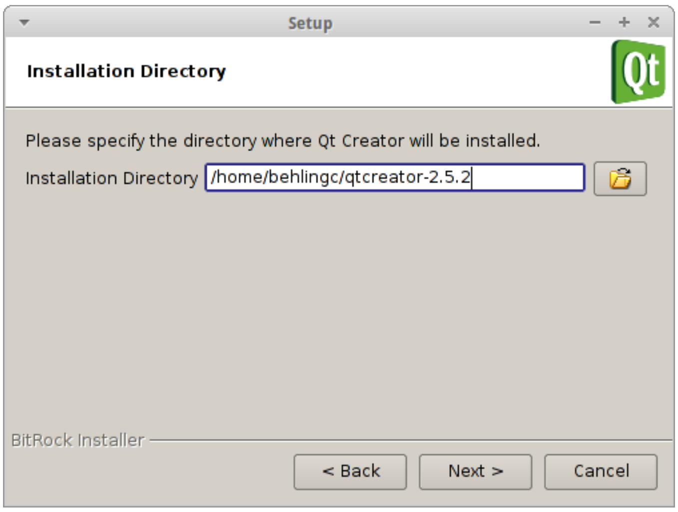


Figure 50: Qt Creator installation path selection

Pressing the **Next >** button leads to the installation start dialog as shown in [▶ Figure 51]. The purpose of this dialog is only confirming and starting the installation process. This can be done by pressing the **Next >** button again.

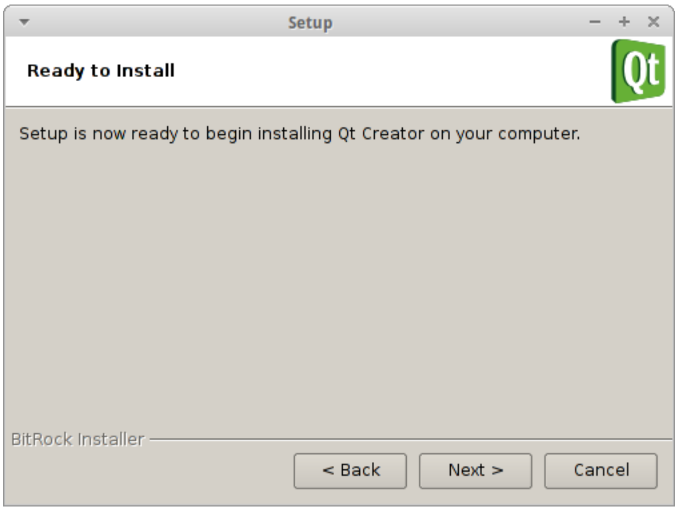


Figure 51: Qt Creator installation beginning

The progress of the installation process can be followed with the installation process dialog as shown in [▶ Figure 52].

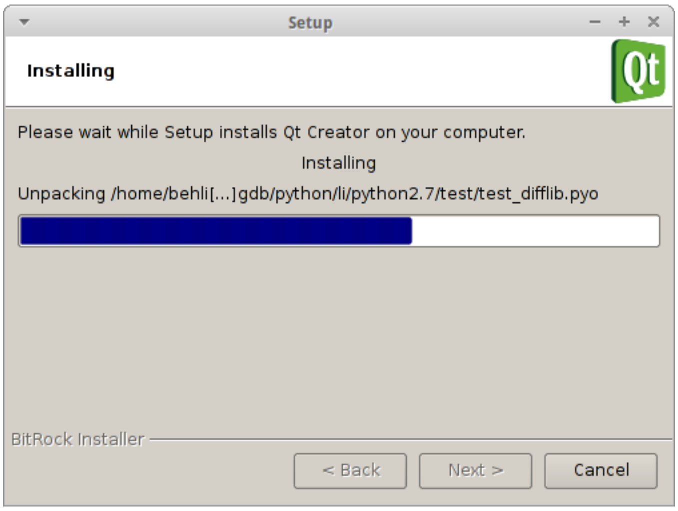


Figure 52: Qt Creator installation process

The installation process ends when the finishing dialog appears as shown in [▶ Figure 53].

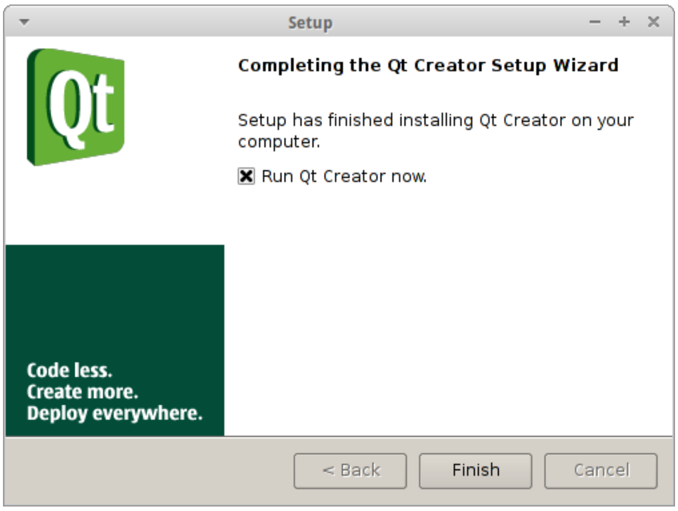


Figure 53: Finishing Qt Creator installation

If the **Run Qt Creator now** option stays selected, Qt Creator will be started after pressing the **Finish** button. This option can be left unchecked and Qt Creator can be started manually later.

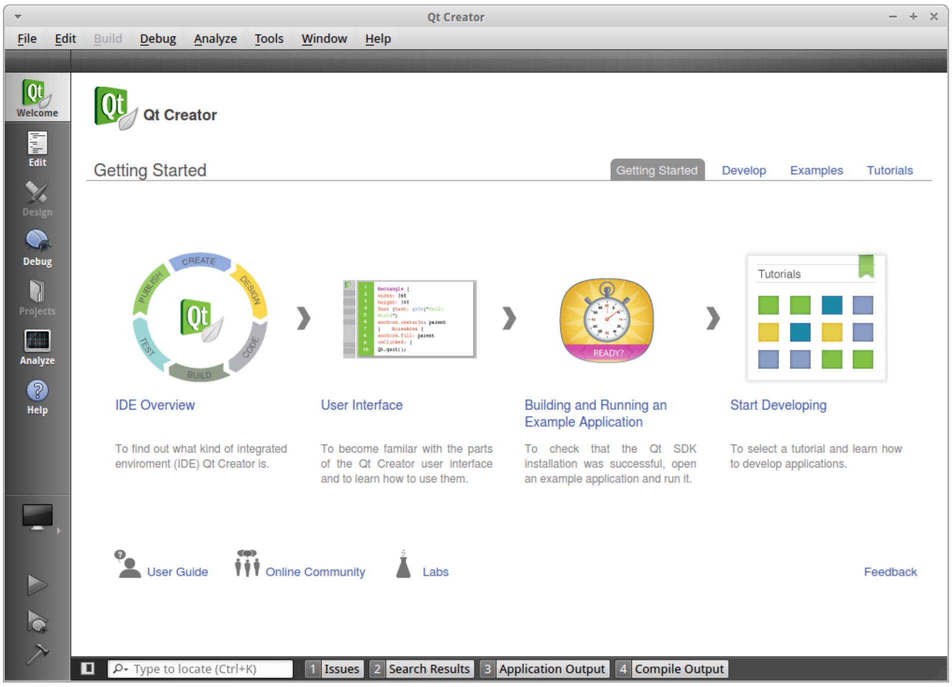


Figure 54: Qt Creator main screen

After finishing the installation process, Qt Creator can by started by pressing the desktop symbol as shown in [▶ Figure 55] or selecting the Qt Creator menu item of the desktops start menu as shown in [▶ Figure 56].

Figure 55: Qt Creator desktop icon

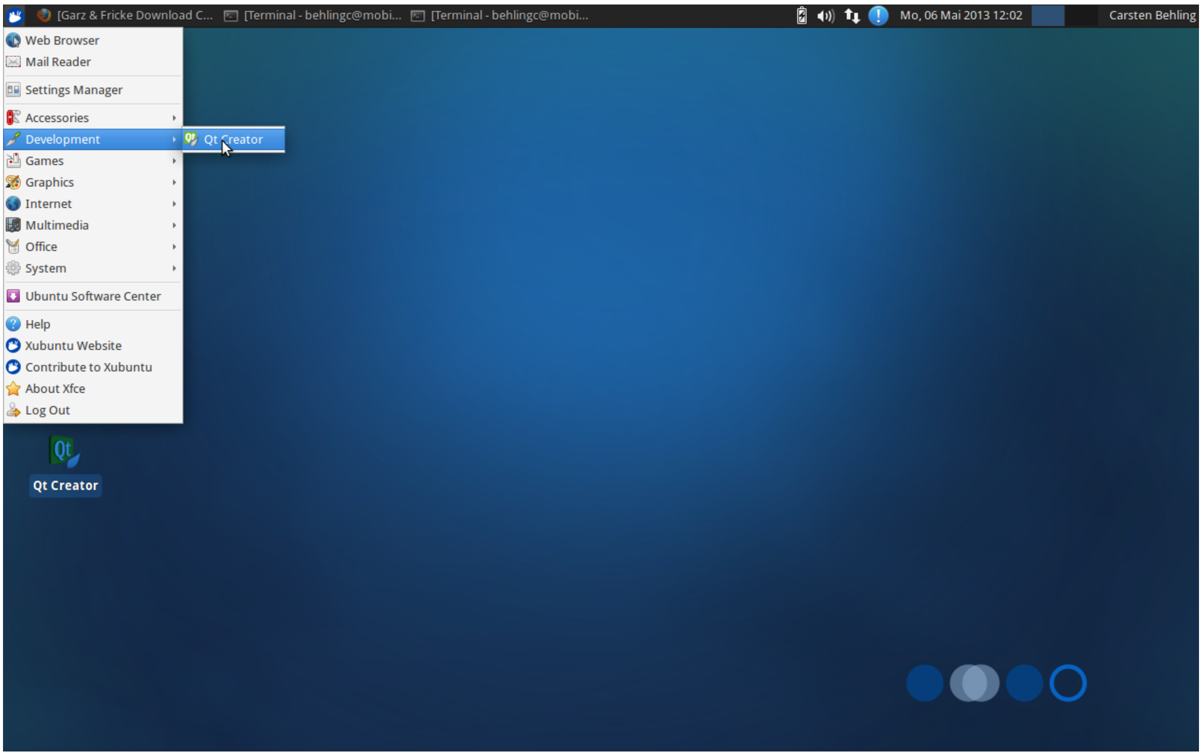


Figure 56: Qt Creator start menu item

Before creating any Qt applications for SANTARO the involved tools must be set up properly. This can be done by choosing the **Tools->Options** dialog as shown in [▶ Figure 57].

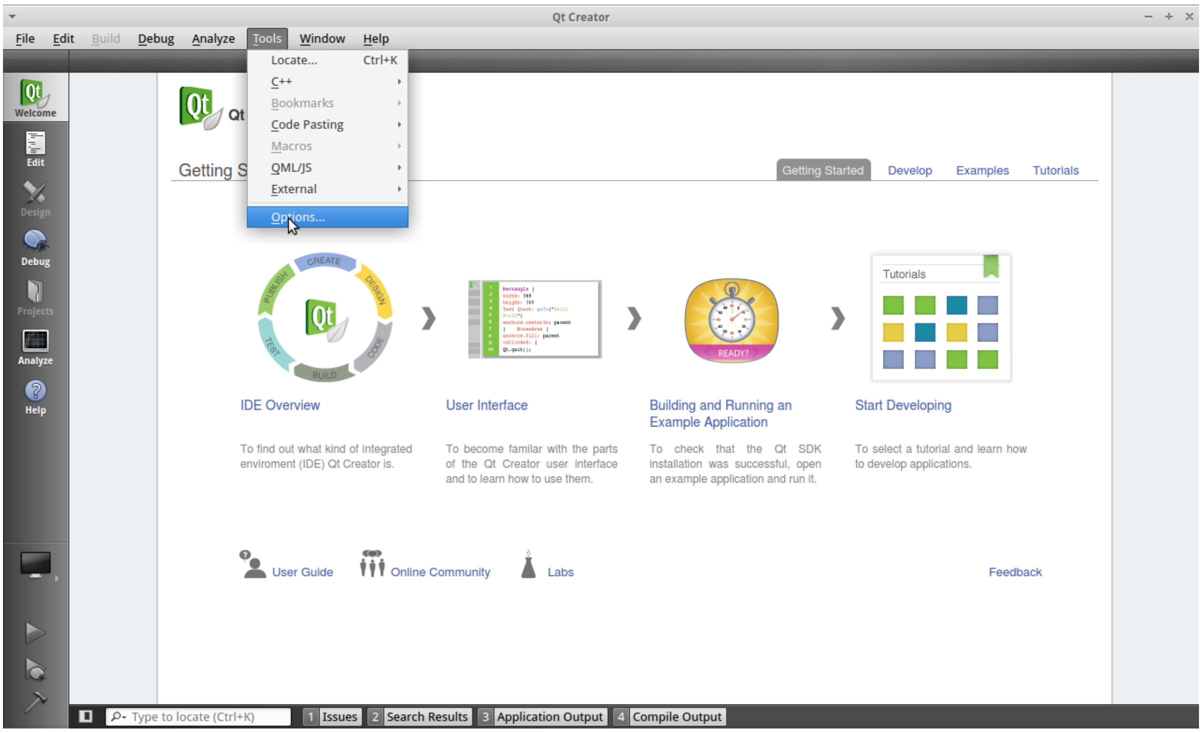


Figure 57: Qt Creator options selection

At first, it is necessary to set up the GNU cross compiler and the GNU cross debugger. This can be done by choosing the tab **Tool Chains** under **Build & Run** and selecting **Add->GCC** as shown in [▶ Figure 58].

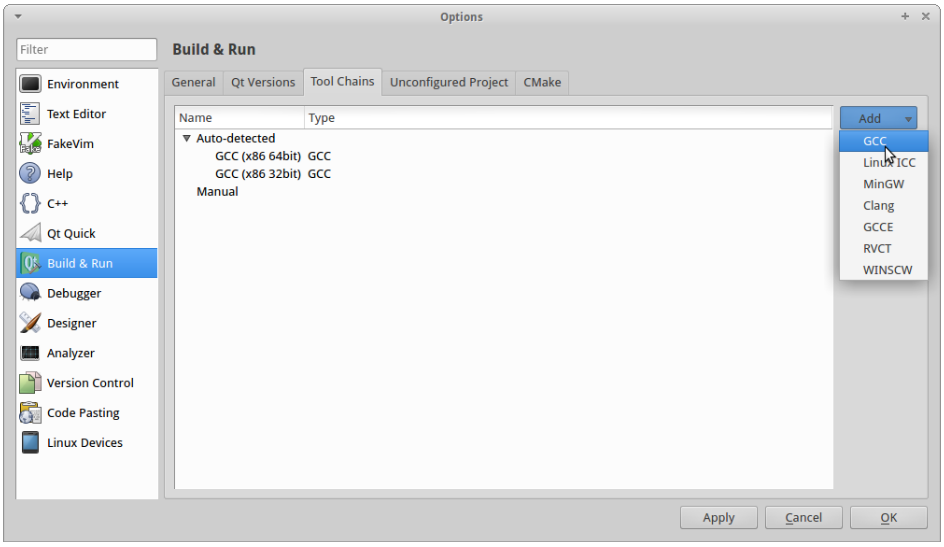


Figure 58: Qt Creator cross compiler setup

After choosing **Add->GCC** the dialog will be extended with selection fields **Compiler path:** and **Debugger:** as shown in [▶ Figure 59]. By using the **Browse ...** buttons those two paths can be browsed.

If the GNU crosstoolchain is installed in the default location like described in [▶ 6.3 Installing the GNU cross toolchain for the target architecure] the following path must be chosen for the GNU cross compiler:

- **/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-fsl-3.0.35-1.44.4-0-sanitized/bin/arm-cortexa9-linux-gnueabi-gcc**

Likewise, the GNU cross debugger path is:

- **/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-fsl-3.0.35-1.44.4-0-sanitized/bin/arm-cortexa9-linux-gnueabi-gdb**

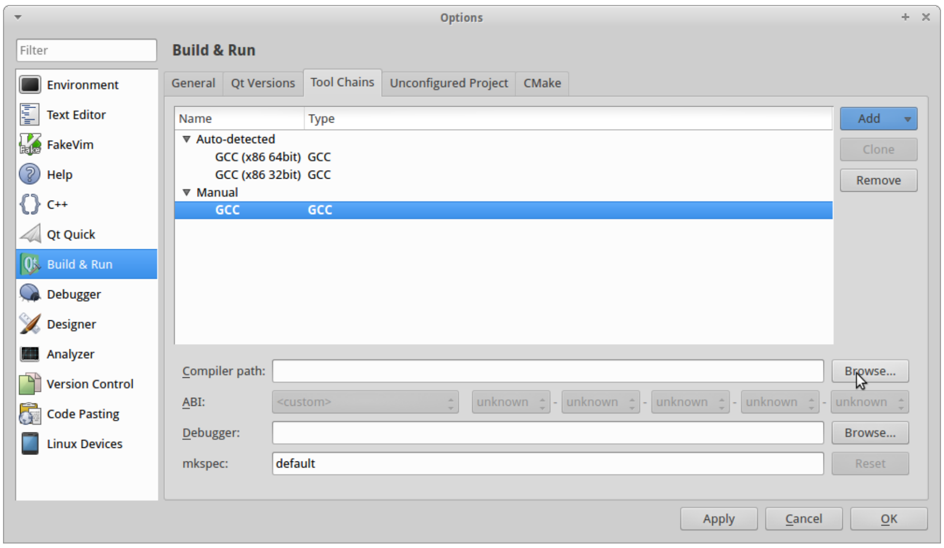


Figure 59: Qt Creator cross compiler and cross debugger selection

If the GNU cross gcc and the GNU cross debugger are choosen similar to [▶ Figure 60], the settings can be confirmed by pressing the **Apply** buton.

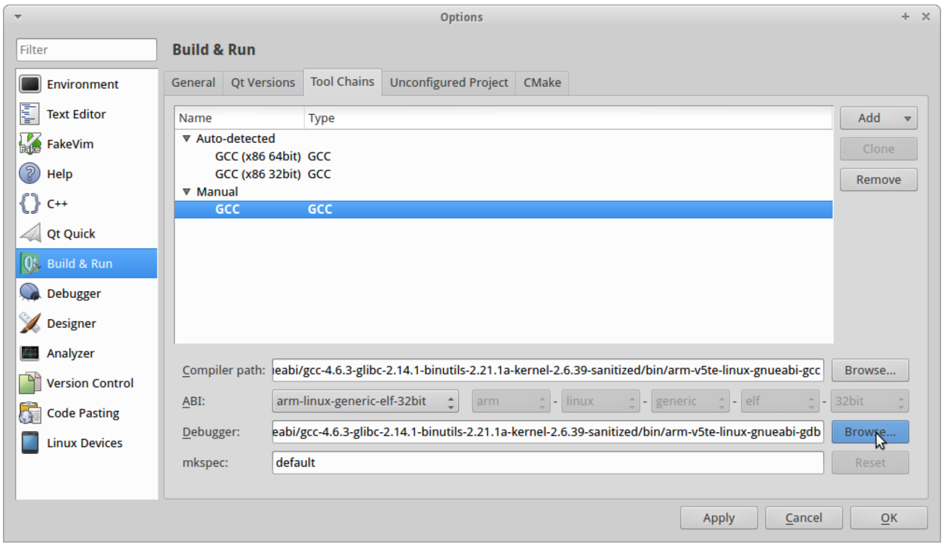


Figure 60: Selected cross compiler and cross debugger in Qt Creator

To make Qt Creator use the Qt version of the SANTARO BSP, a Qt version configuartion with the appropriate settings must be added. This can be done from the **Qt versions** tab in the Tools Options menu under **Build & Run** as shown in [▶ Figure 61].

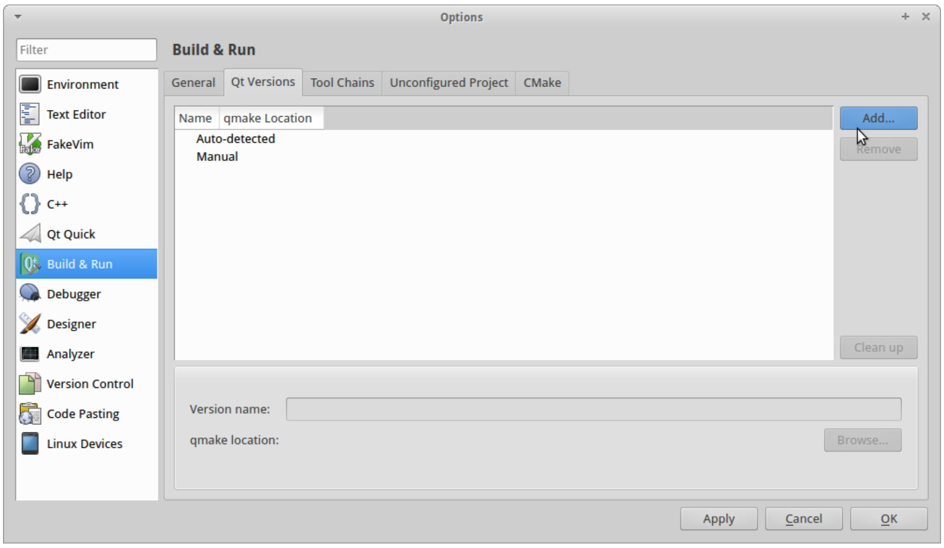


Figure 61: Qt Creator Qt version setup

After pressing the **Add...** button, the location of the **qmake** tool of the desired Qt version must be chosen as shown in figure [▶ Figure 62].

After a complete BSP build like described in chapter [▶ 6.5 Building the BSP for the target platform with PTXDist], **qmake** is present inside the BSP under the path:

- **OSELAS.BSP-GUF-Linux-1.44.4-0/platform-SANTARO/sysroot-cross/bin/qmake**

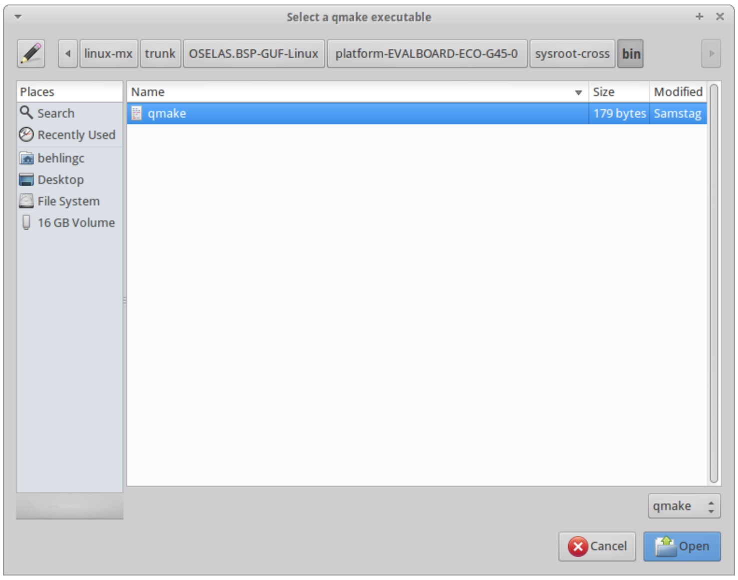


Figure 62: Qt Creator BSP cross qmake selection

By having access to the qmake tool, Qt Creator is able to query all other settings like header and libraries automatically. The current settings can be confirmed by pressing the **Apply** button as shown in [▶ Figure 63].

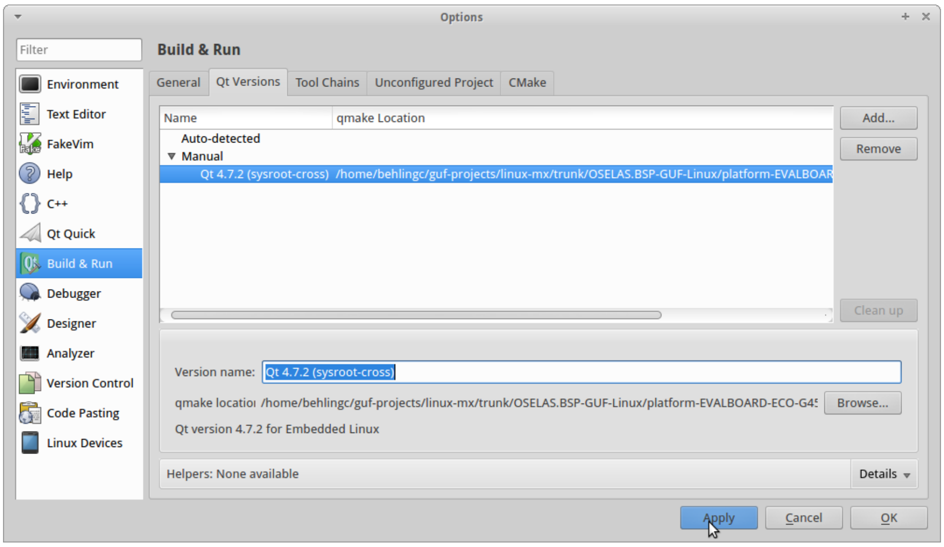


Figure 63: Selected Qt version in Qt Creator

To be able to transfer and run or debug a Qt application automatically on the SANTARO target device, a device configuration has to be added. This can be done with the **Device Configuration** tab under **Linux Devices** as shown in [▶ Figure 64] by pressing the **Add...** button.

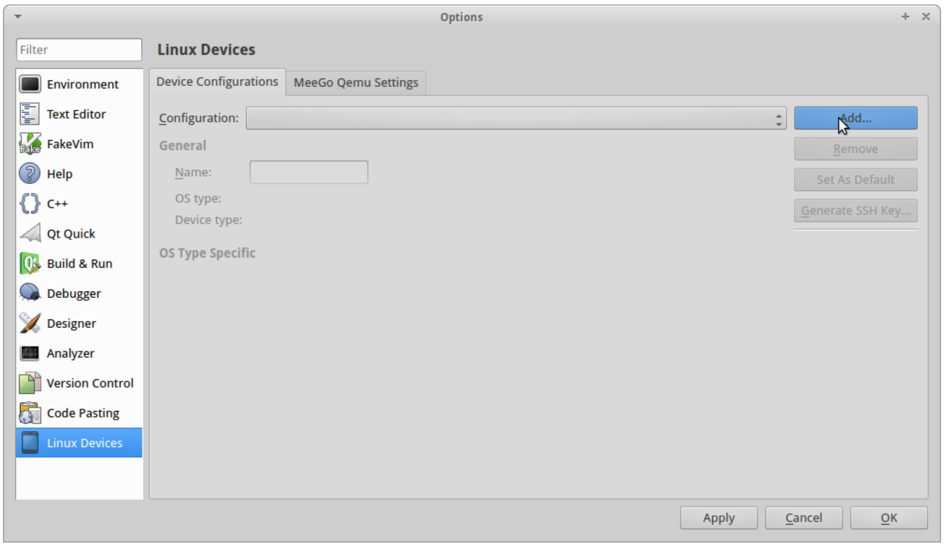


Figure 64: Adding a new Linux device configuration in Qt Creator

In the following dialog, a device configuration wizard can be selected as shown in [▶ Figure 65]. **Generic Linux Device** must be selected.

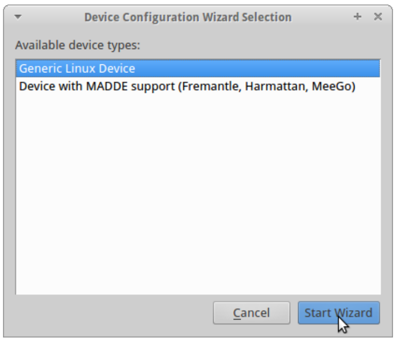


Figure 65: Selecting the Generic Linux Device wizard in Qt Creator

After pressing the **Start Wizard** button, the SSH connection data can be entered as shown in [▶ Figure 66]. Assuming that the target has the default Garz & Fricke IP configuration **192.168.1.1/255.255.255.0**, the network connection between target and host is established and no root password is set on the target system, the settings as shown in [▶ Figure 66] can be used.

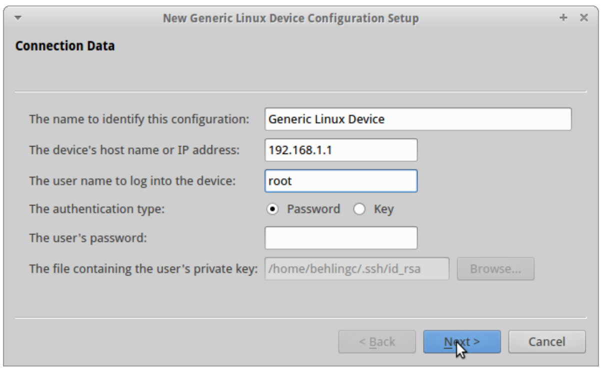


Figure 66: Device configuration setup configuration

After pressing the **Next >** button, the Setup Finished dialog appears as shown in [▶ Figure 67].

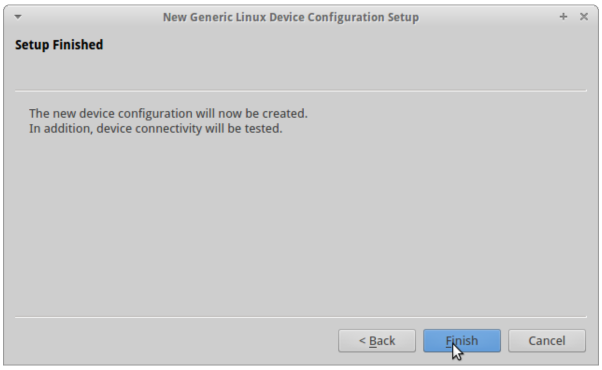


Figure 67: Setup Finished dialog

The connection is tested automatically when the **Finish** button is pressed as shown in [▶ Figure 68].

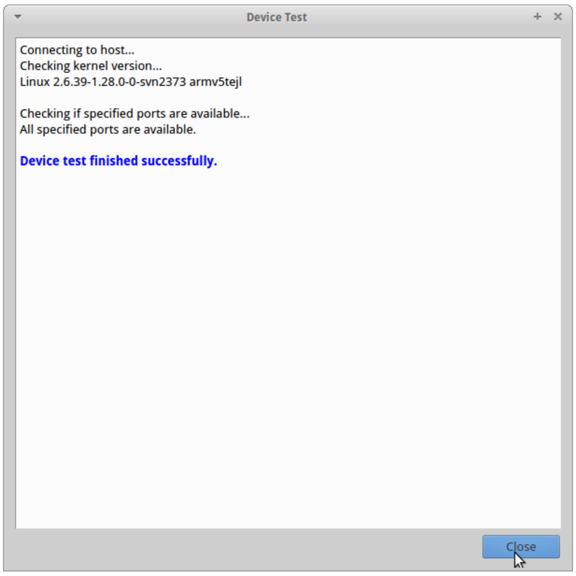


Figure 68: Qt Creator device connection test

After the test, the device configuration settings can be confirmed by pressing the **Apply** button as shown in [▶ Figure 69].

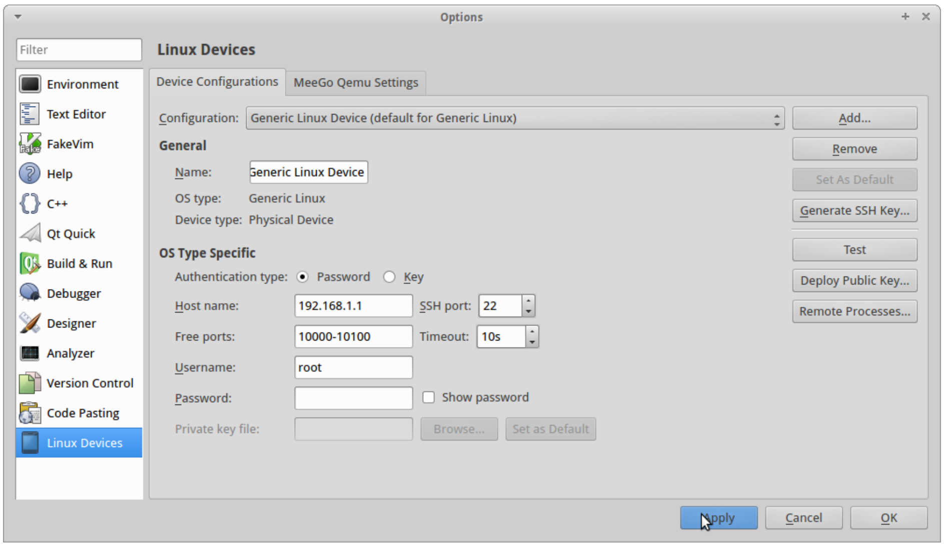


Figure 69: Qt Creator device configuration settings

By pressing the **OK** button the tools options dialog can be left.

Havin set up the toolchain and the device configuration, a simple GUI project for SANTARO can be built using Qt Creator with the apropriate cross toolchain and the Qt version for the BSP. A simple Qt GUI application skeleton can be generated with the Qt Creator wizard by selecting **File->New File or Project...** as shown in [▶ Figure 70].

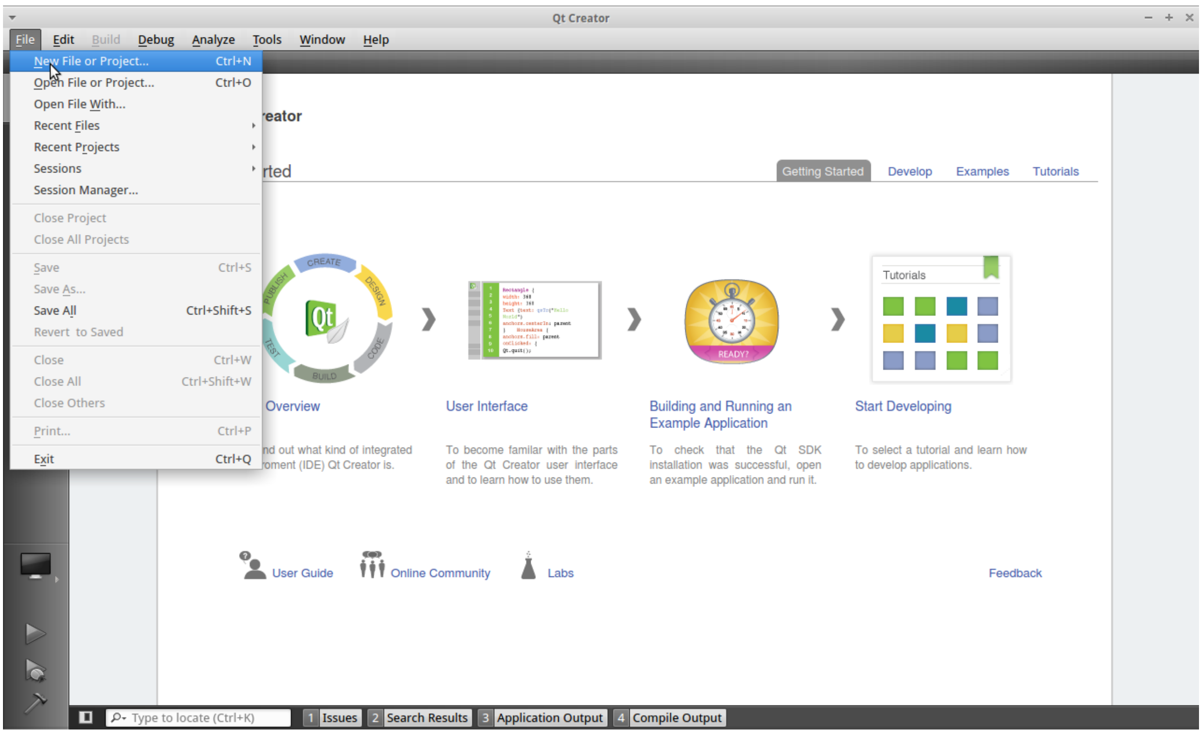


Figure 70: Creating a new project with Qt Creator

First, a template must be chosen as shown in [▶ Figure 71]. **Qt Gui Application** is selected per default. So, this wizard step can be left unmodified and this project type can be chosen by pressing the **Choose...** button.

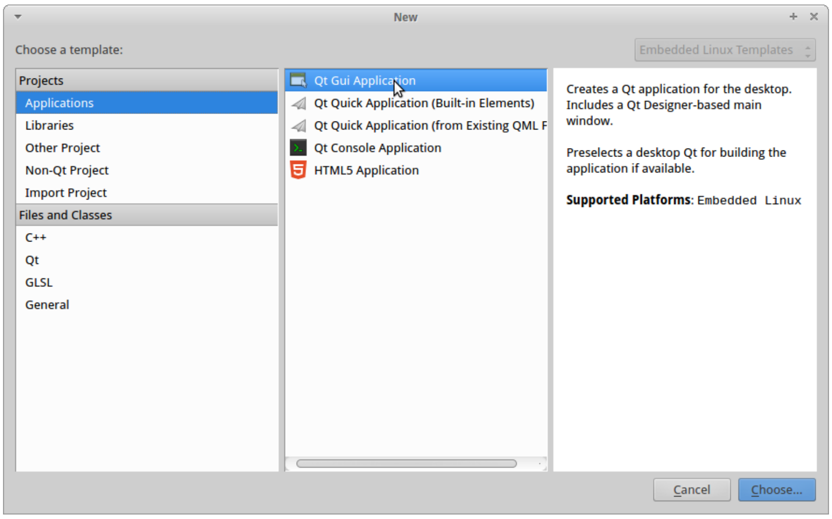


Figure 71: Choosing Qt Gui application type with the Qt Creator wizard

In the next step a project name and location must be set. In this example **qt4-myapp** and the user's home folder is chosen as shown in [▶ Figure 72].

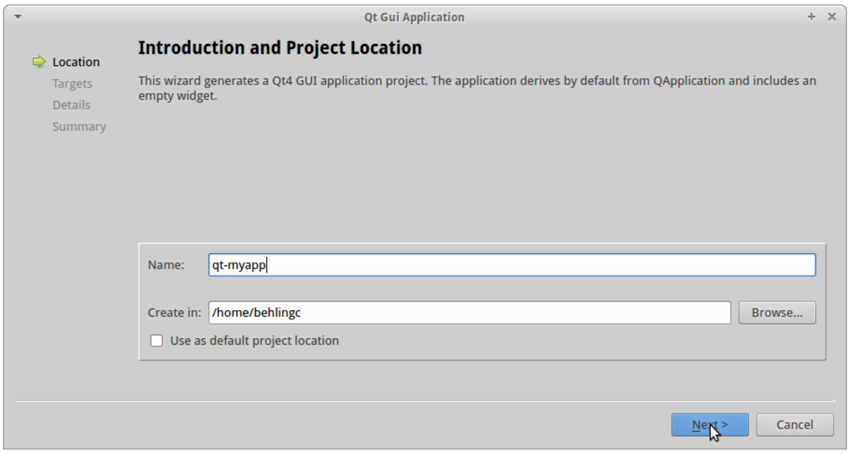


Figure 72: Project name and location selection with the Qt Creator wizard

After pressing the **Next >** button, the dialog for the target project type selection will appear as shown in [▶ Figure 60]. If no other configuration is added except for that one described above in this chapter, the only possibility to choose is **Embedded Linux**.

Again, this dialog can be left untouched and confirmed with the **Next >** button.

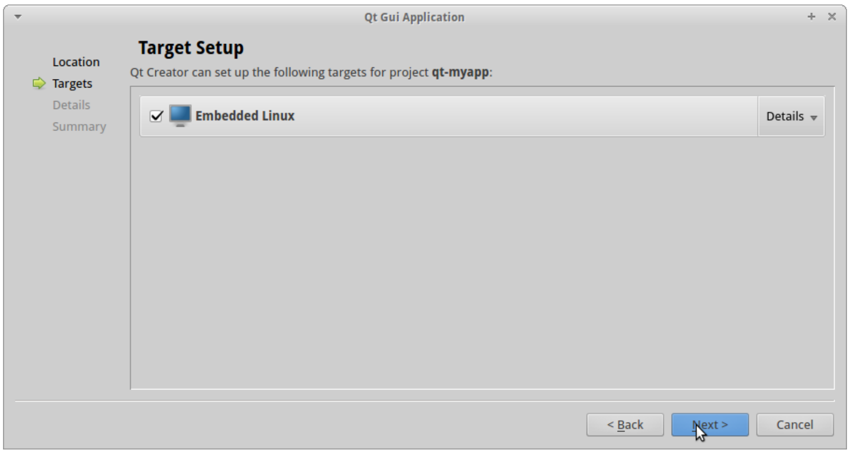


Figure 73: Project target setup with the Qt Creator wizard

With the next dialog, the skeleton class and the skeleton files can modified as shown in [▶ Figure 74]. For this example the settings will be left unmodified.

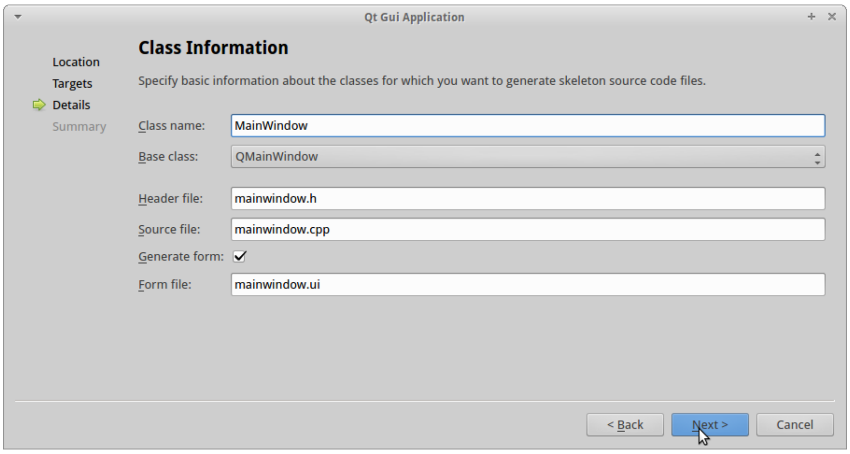


Figure 74: Project skeleton class setup with the Qt Creator wizard

After pressing the **Next >** button, the project menagement dialog will appear with a short summary as shown in [▶ Figure 75].

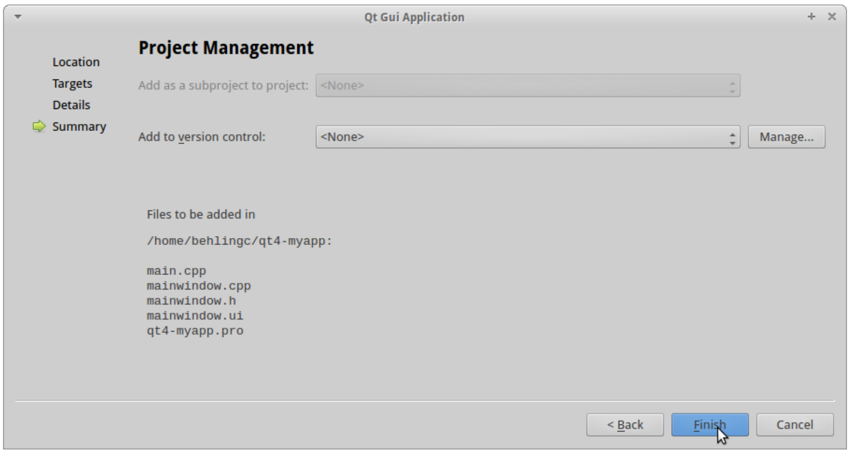


Figure 75: Project management setup with the Qt Creator wizard

By presing the **Finish** button, the project will be created by the Qt Creator wizard and the project screen will appear as shown in [▶ Figure 76].

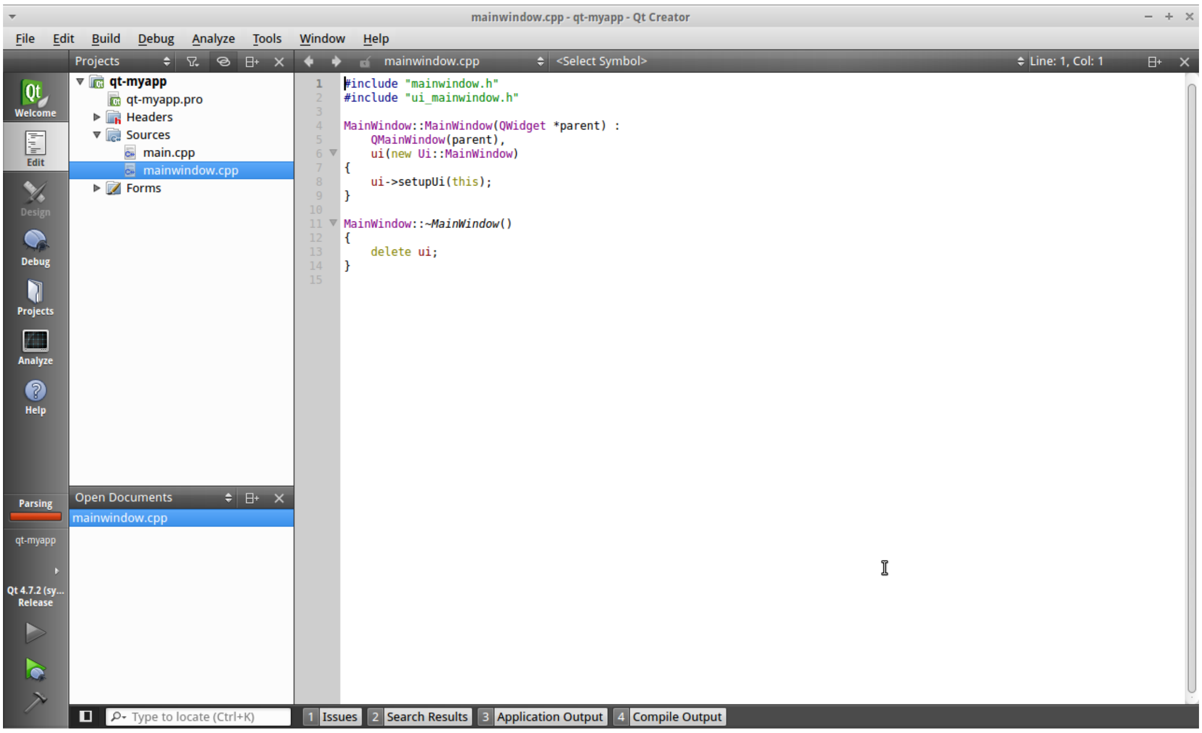


Figure 76: Project screen after project skeleton generation

The project can now be built by selecting **Build->Build All** from the Qt Creator menu as shown in [▶ Figure 77].

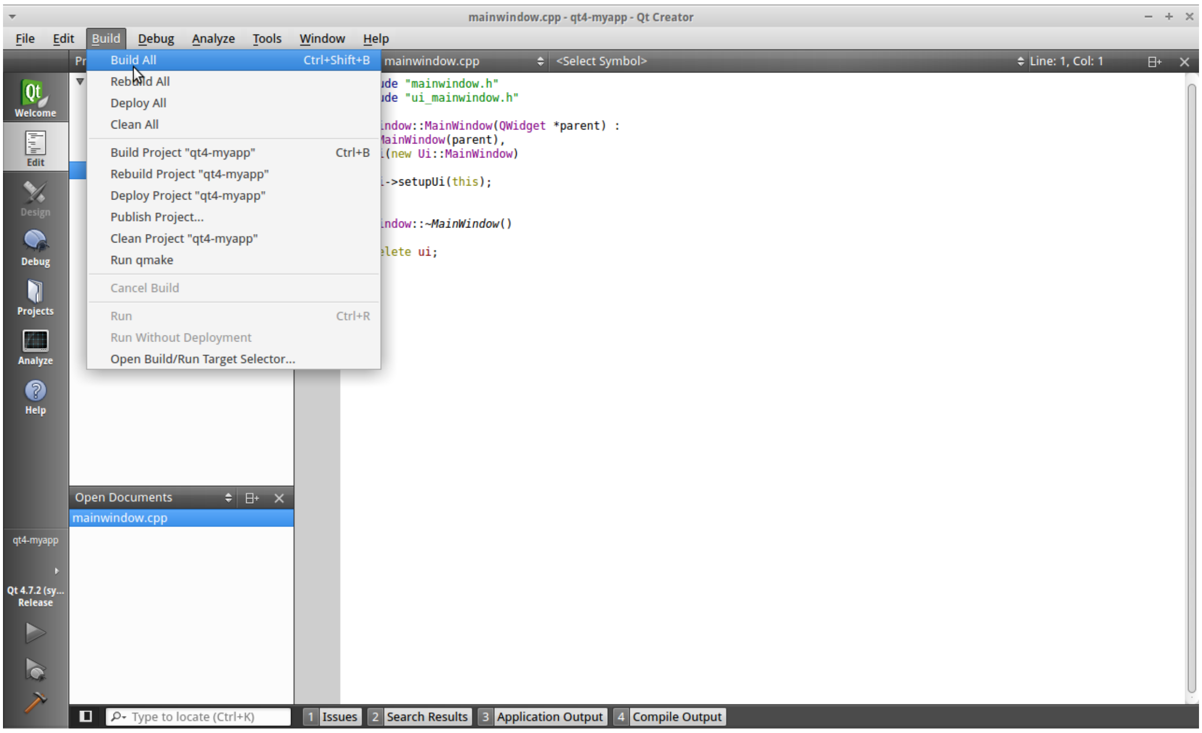


Figure 77: Building the generated project

The executable for is created in a folder parallel to the above selected project folder called:

- /home/<user>/qt4-myapp-build-embedded-Qt_4_7_2__sysroot-cross_Release/qt4-myapp

To enable Qt Creator to transfer und execute the application automatically to the target device, the run settings of the project must be set up. This can be done by selecting the **Run Settings** tab under **Projects** as shown in [▶ Figure 78].

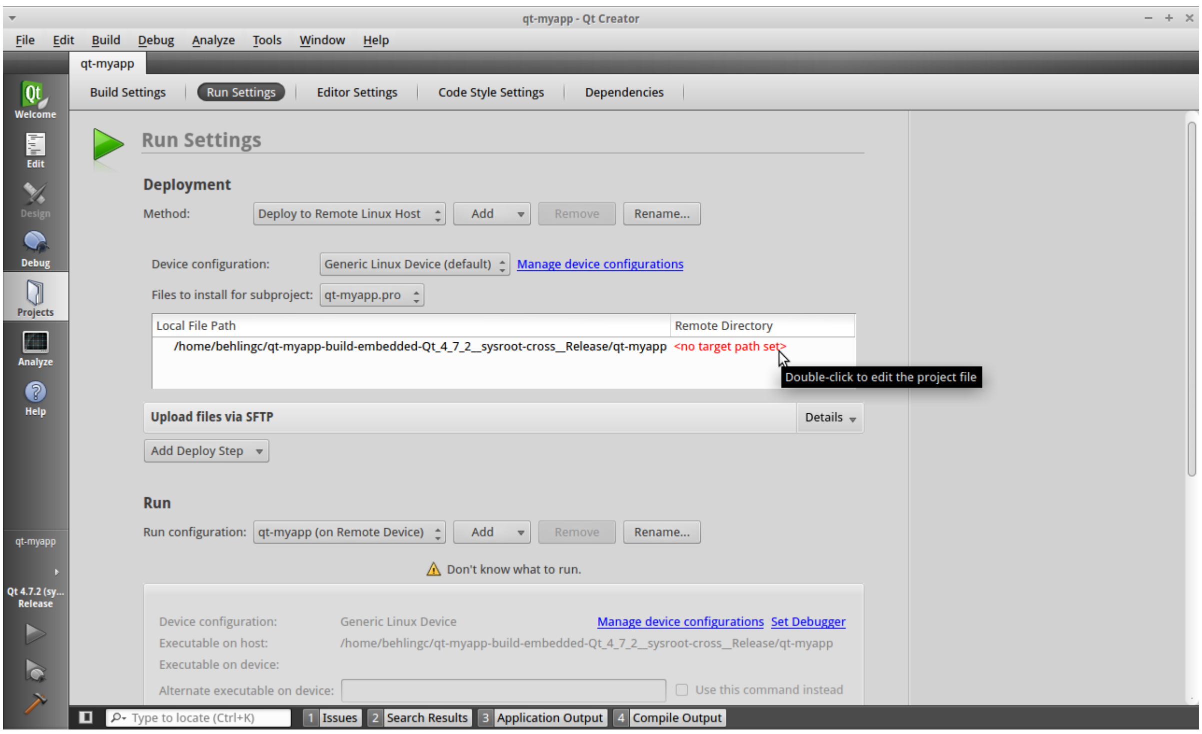


Figure 78: Qt Creator project run settings

Qt Creator complains per default that no target path is set for the **Remote Directory** for the target device as also shown [▶ Figure 78].

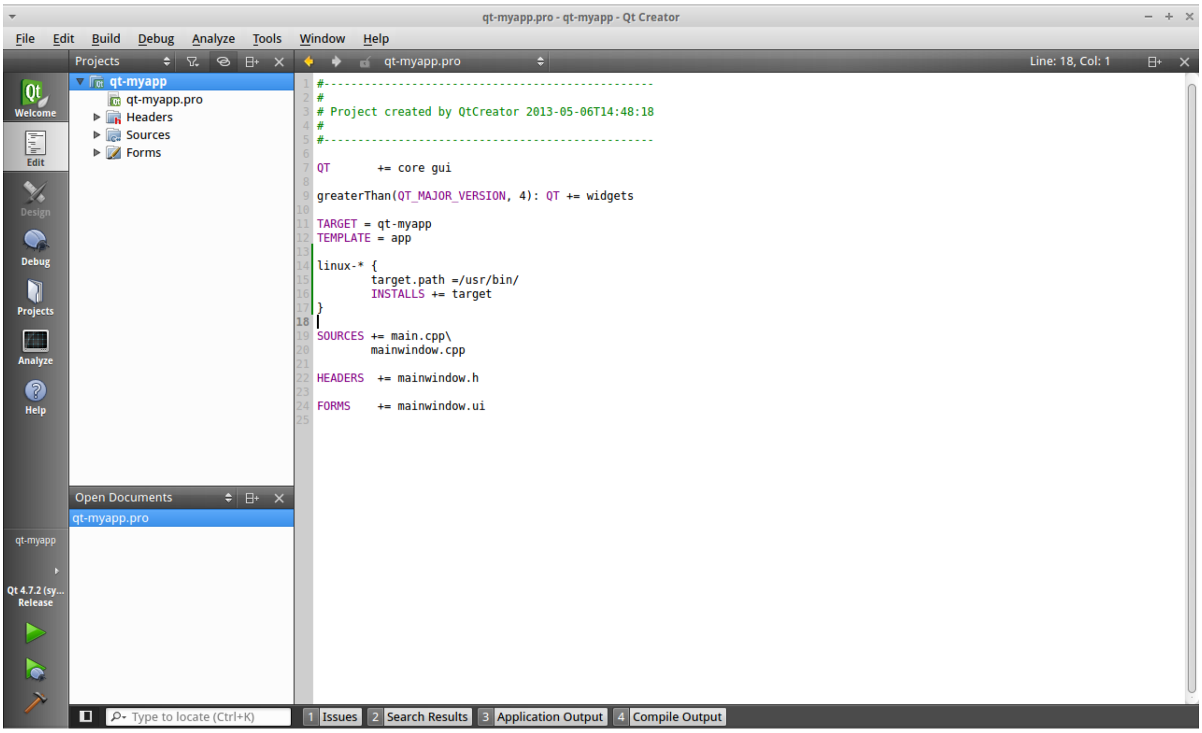


Figure 79: Adding a target install path to the Qt project file

An install target path can be added by double-clicking on **<no target path set>**. The editor should then appear with the Qt project file as shown in [▶ Figure 79]. The following lines must be added:

```
...
linux-* {
        target.path = /usr/bin
                INSTALLS += target
}
...
```

The complete Qt project file is depicted in [▶ Figure 79].

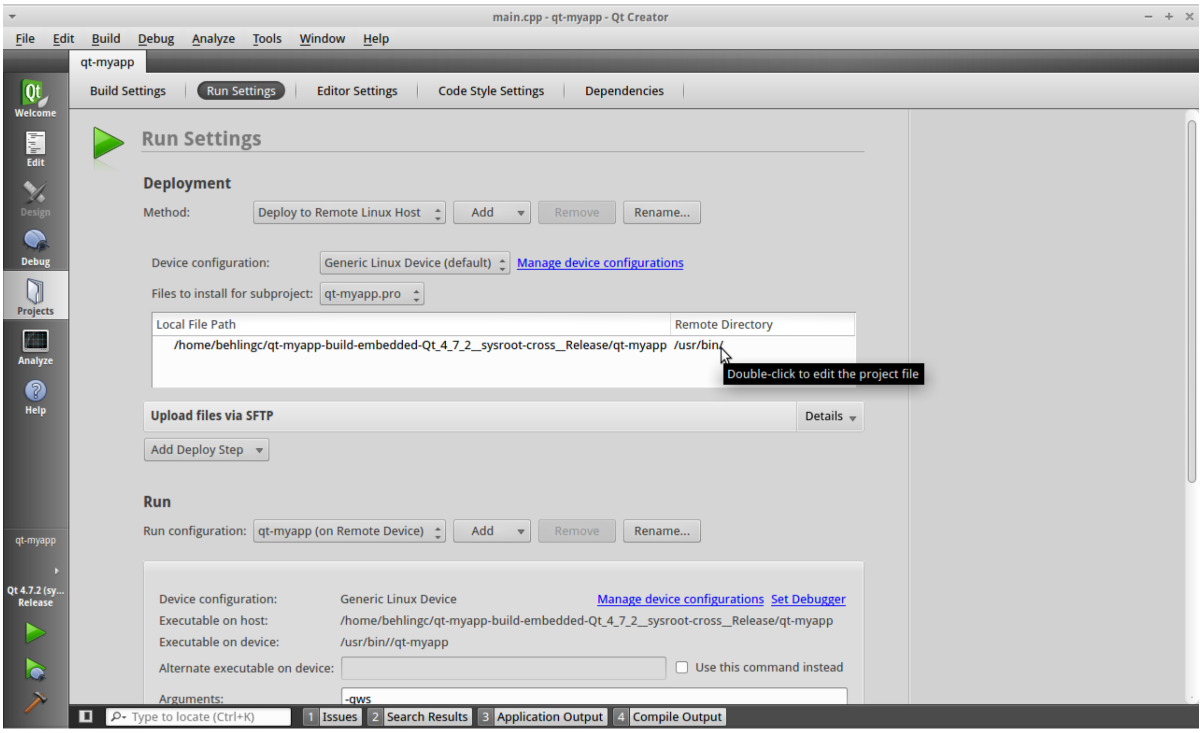


Figure 80: Porperly setup of the remote directory

With this modification, the **Remote Directory** should be set up properly now as shown in [▶ Figure 80].

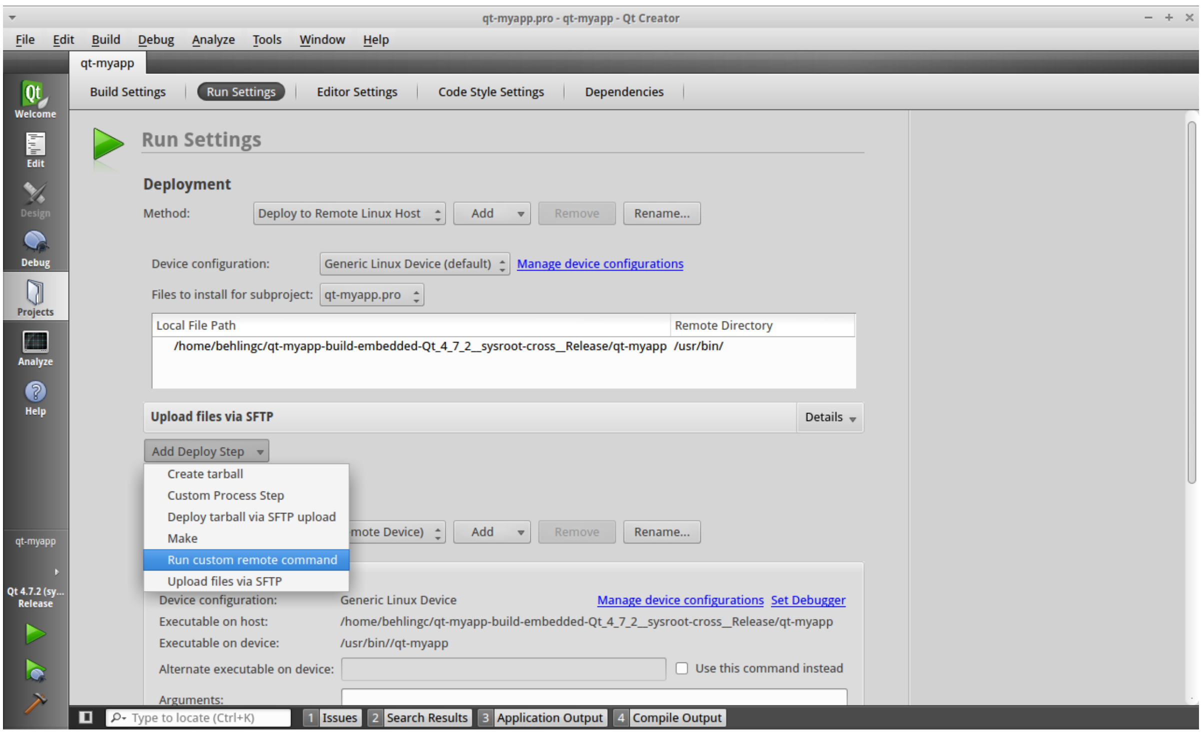


Figure 81: Adding a remote command as deploy step

Because the Garz & Fricke demo application is running by default, a command that kills this demo application must be added as a deploy step. Otherwise, both applications will overwrite the screen. This can be done by selecting **Add Deploy Step** and **Run custom remote command** as shown in [▶ Figure 81].

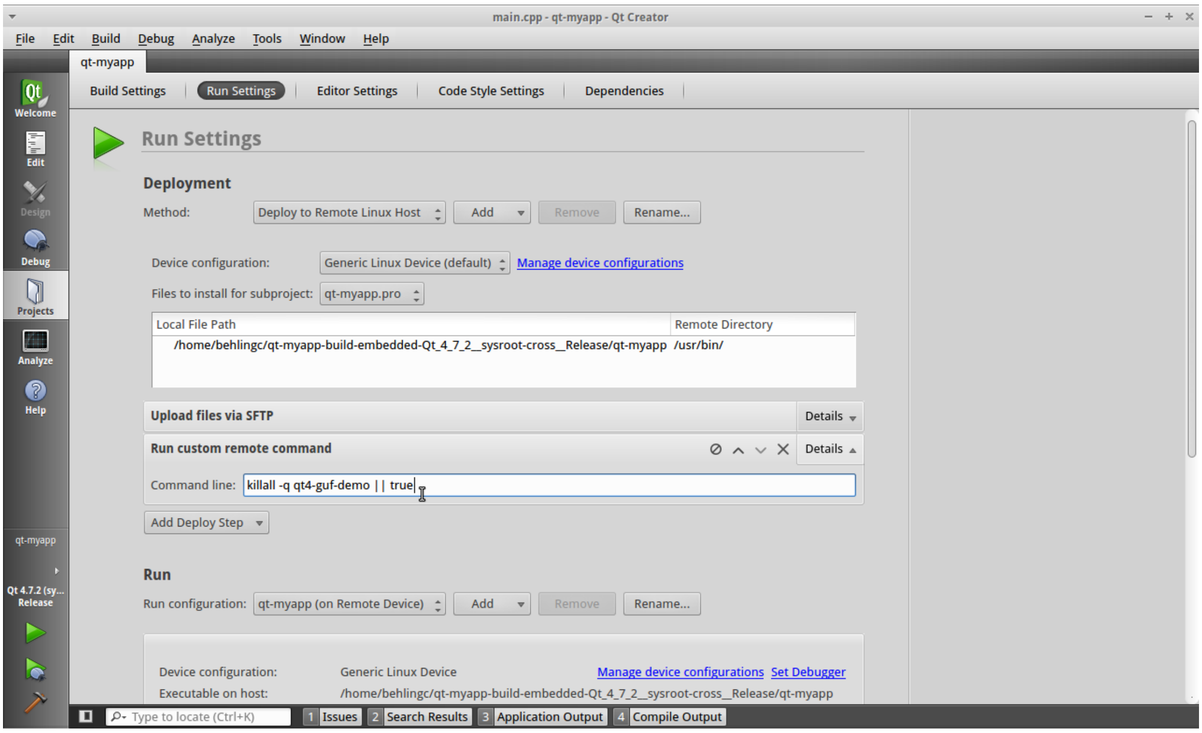


Figure 82: Adding the kill command for the demo application to the deploy step

The following command will kill the demo application before the project application is started remotely:

```
killall -q qt4-guf-demo || true
```

This command has to be entered in the **Command line** of the deploy step as shown in [▶ Figure 82].

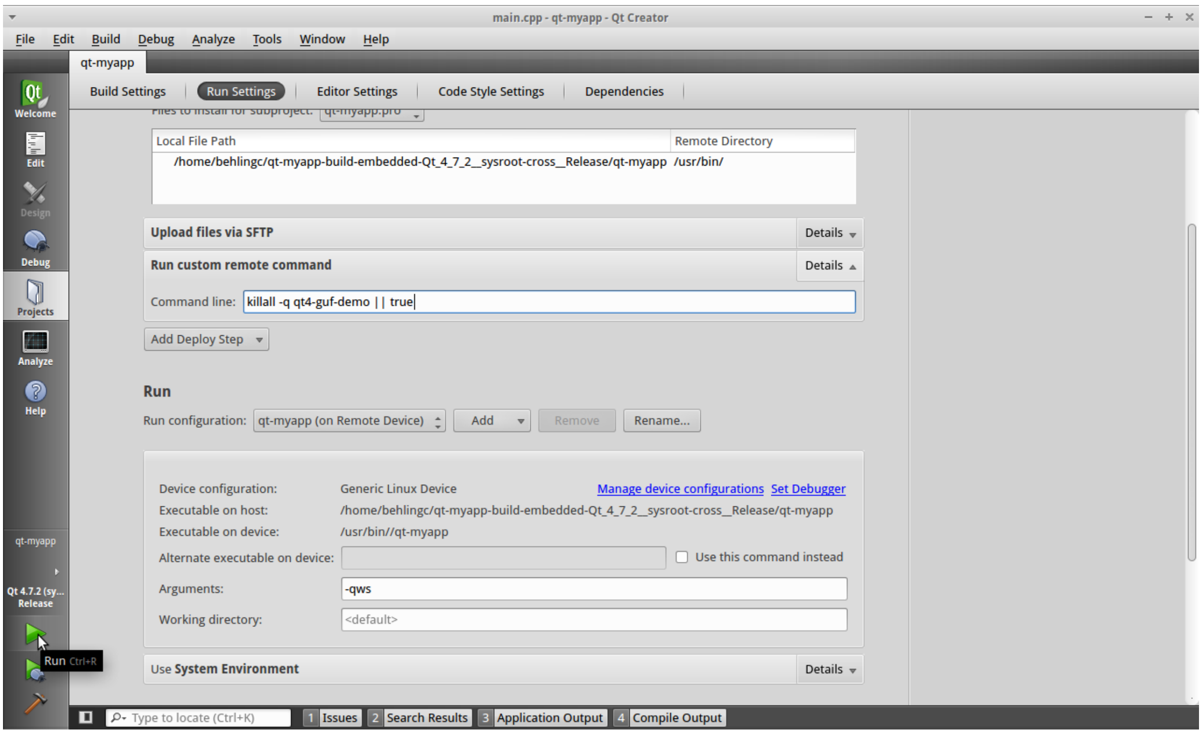


Figure 83: Adding the -qws command line argument and start the deployment

Finally, the **-qws** command line argument must be configured for the project application as shown in [▶ Figure 83].

The application can be executed on the target now either by pressing the green **Play** button as shown in [▶ Figure 83] or by selecting **Build->Run** in the Qt Creator menu as shown in [▶ Figure 84].

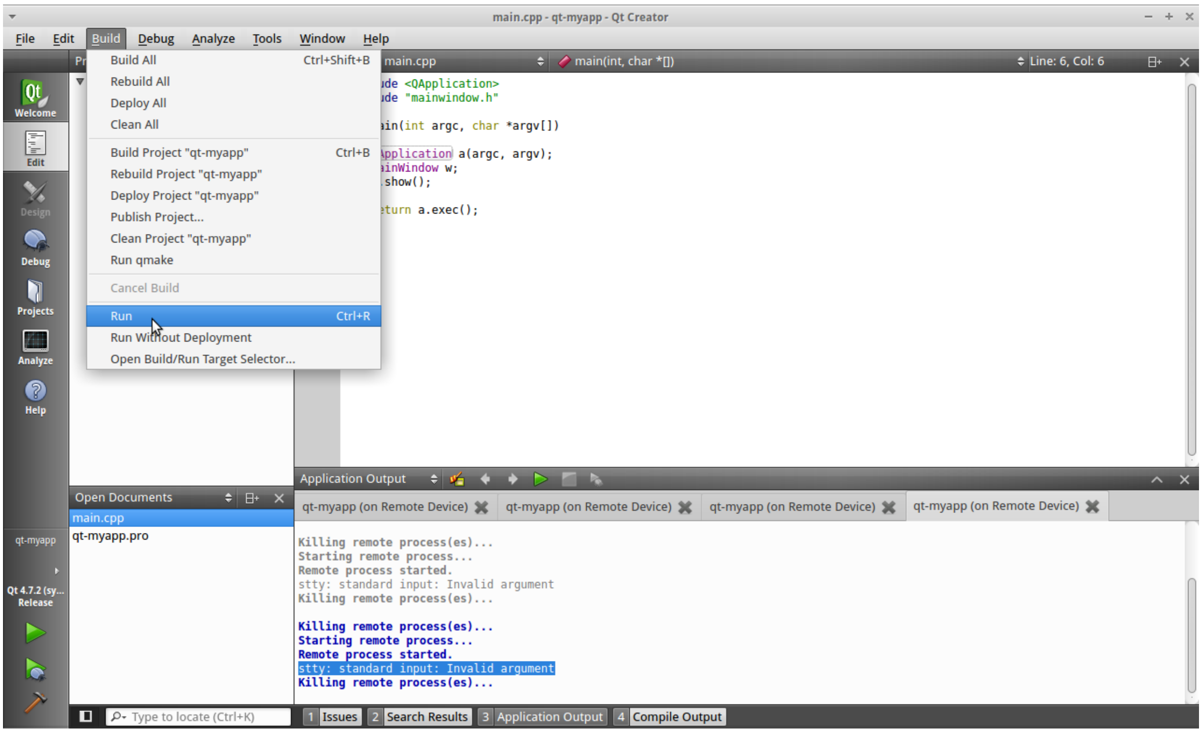


Figure 84: Running the project application remotely from the Qt Creator IDE

8.3 Autostart mechanism for user applications

In order to make the target system start your application automatically during the boot process you have to create a start/stop script in the **/etc/init.d** directory. As described in chapter [▶ 4.1 Services], this directory contains a number of scripts for various services on your system. Each script will be run as a command of the following form:

```
root@SANTARO:~ /etc/init.d/<COMMAND> <OPTION>
```

Where **COMMAND** is the actual command to run and **OPTION** can be one of the following:

- start
- stop

The command can be called by hand to start or stop a specific service. In order to start a service automatically during system boot, a link to the script has to be created in the **/etc/rc.d** directory. In this directory, the filename of each link starts with an **S**, followed by a two-digit number representing the execution order.

For your demo application, create a new script at **/etc/init.d/myapp** on the target system:

```
root@SANTARO:~ touch /etc/init.d/myapp
```

Edit **/etc/init.d/myapp** with the editor nano on the target system by typing:

```
root@SANTARO:~ nano /etc/init.d/myapp
```

Change the contents of this file as follows:

```
#!/bin/sh

. /etc/profile

case "$1" in
start)
     start-stop-daemon -m -p /var/run/myapp.pid -b -a /usr/bin/myapp -S
     ;;
stop)
     start-stop-daemon -p /var/run/myapp.pid -K
     ;;
*)
     echo "Usage: /etc/init.d/myapp {start|stop}" >&2
     exit 1
;;
esac
```

Save the changes by pressing **Ctrl+O** and accept the target file name as suggested by pressing **[RETURN]**. Leave the nano editor by pressing **Ctrl+X**.

Create a startlink in **/etc/rc.d/**:

```
root@SANTARO:~ ln -s /etc/init.d/myapp /etc/rc.d/S95myapp
```

If the Garz & Fricke demo application is installed on your device, its startlink should be deleted so that your application is the only application automatically started:

```
root@SANTARO:~ rm -f /etc/rc.d/S95qt4-guf-demo
```

After system reboot your application will start automatically.

8.4 Configuring the Qt Webkit demo

The Linux BSP for SANTARO contains a small Qt Webkit demo application, which simply displays a website over the whole screen. You can start this demo using its start/stop script in **/etc/init.d**:

```
root@SANTARO:~ /etc/init.d/qt4-guf-webdemo start
```

Without any modifications, the demo displays the local HTML page **/home/guf/site/index.htm**, as configured in the script itself:

```
#!/bin/sh

case "$1" in
start)
     start-stop-daemon -m -p /var/run/qt4-guf-webdemo.pid -b -a \
        /usr/bin/qt4-guf-webdemo -S -- -qws --no-scrollbars /home/guf/site/index.htm
     ;;
stop)
     start-stop-daemon -p /var/run/qt4-guf-webdemo.pid -K
     ;;
*)
     echo "Usage: /etc/init.d/qt4-guf-webdemo {start|stop}" >&2
     exit 1
     ;;
esac
```

For displaying your own HTML page, either load your page into the local default path (and overwrite the file **index.html**), or change the path in line 6 of the script, e.g. using nano:

```
root@SANTARO:~ nano /etc/init.d/qt4-guf-webdemo
```

Per default, scrollbars are disabled in the Webkit browser. If you want to enable scrollbars, remove the **--no-scrollbars** parameter preceeding the webpage path.

For having the webdemo automatically started on system startup, use the autostart mechanism described in the precedent chapter [▶ 8.3 Autostart mechanism for user applications].

9 Garz & Fricke Support Libraries

This chapter provides the support libraries Garz & Fricke offers you to support your application development.

10 Related documents and online support

This document contains product specific information. Additional documentation is available for the use of embedded operating systems, the related tool chain and the bootloader (BIOS).

Title	File Name	Description
RedBoot User Manual	GF_RedBoot_User_Manual_Rnn.pdf	Contains relevant information about BIOS, boot logo, display settings, etc. in the case that RedBoot is used as BIOS.
U-Boot User Manual	GF_U-Boot_Manual_SANTARO-1.44.4-0.pdf	Contains relevant information about BIOS, boot logo, etc. in the case that U-Boot is used as BIOS.
Windows OS Manual	GF_WindowsCE_Manual_Vn.n.pdf	Contains information about Windows Embedded CE, the tool chain, the development environment Visual Studio, Garz & Fricke tools, etc..
Linux OS Manual	GF_Linux_Manual_SANTARO-1.44.4-0.pdf	Contains information about Linux BSP, the tool chain, Qt, etc..
SAM-BA User Manual	GF_SAM-BA_Manual_SANTARO-1.44.4-0.pdf	Contains relevant information about the usage on ATMEL's SAM-BA tool with Garz & Fricke devices in the case that an AT91SAM based platform is used.

Support for your Garz & Fricke embedded device is available on the Garz & Fricke website. You may find a list of the documents available, as well as their latest revision and updates for your system:

- http://www.garz-fricke.com/santaro-download

A GNU General Public License v2

Version 2, June 1991

A.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD

THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.3 END OF TERMS AND CONDITIONS

A.3.1 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it
starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
```

```
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.