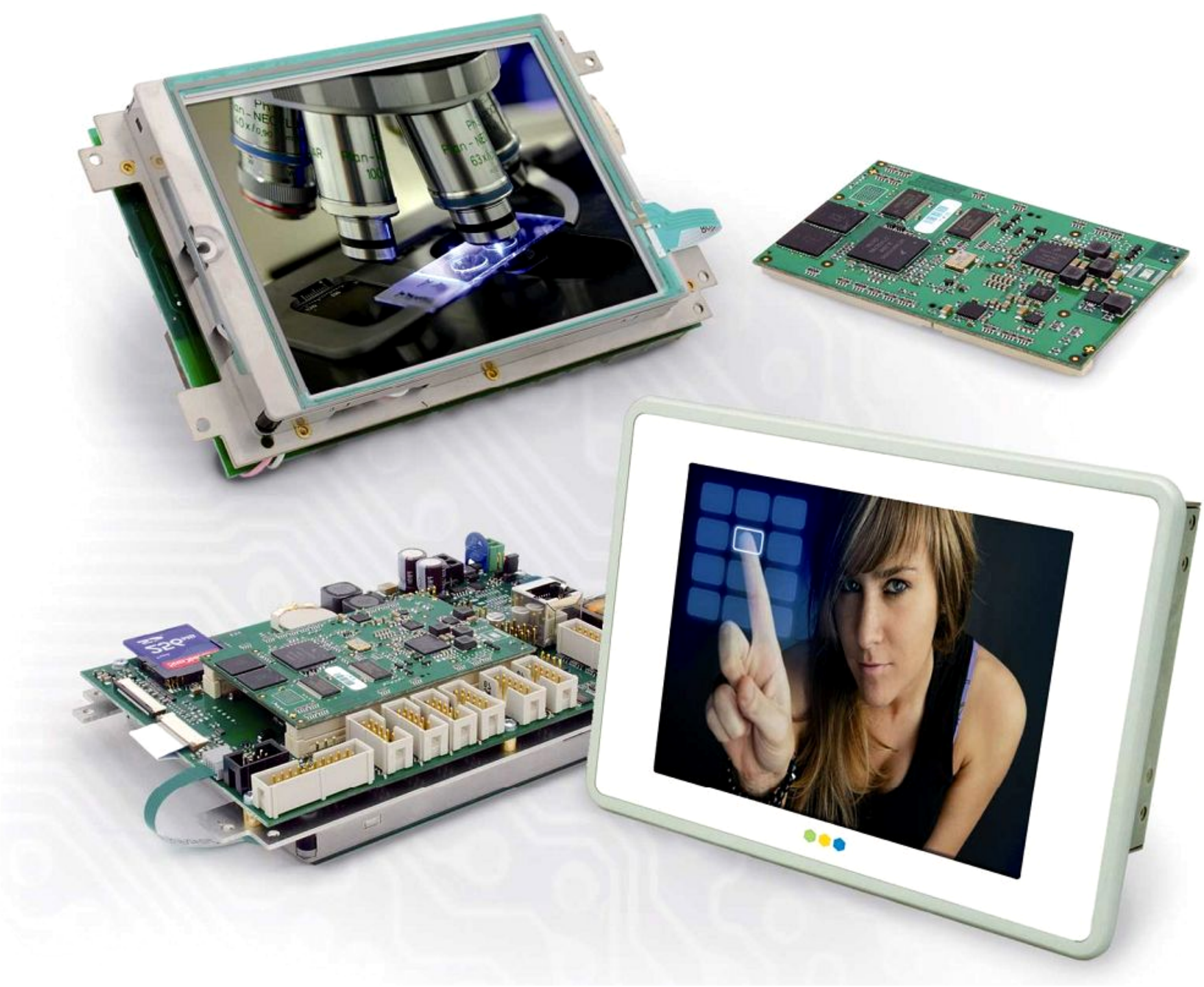
GARZ FRICKE

Embedded Systems

# Garz & Fricke

Embedded Computer Systems



# Yocto · User Manual

GUF-Yocto-21.0-r5061-0 · SANTARO/SANTOKA

Built on 19.06.2015

freescale Enabled

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 [▶ 11 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/SANTOKA**, 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/Linux-Yocto/Releases/

It does not include the complete source code to all packages. Instead, several external packages are downloaded from third party online sources and from the Garz & Fricke packages mirror during the build process: If third party souces are not available anymore at the former location there should be a backup available at:

- http://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. 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/SANTOKA 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/SANTOKA is equipped with **Qt5**.

## 2.4 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/SANTOKA 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/SANTOKA 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

## 2.5 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

Information about the Yocto Project can be found at:

- https://www.yoctoproject.org

Documentation of the Yocto Project can be found at:

- https://www.yoctoproject.org/documentation/current

# 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
- **SFTP** 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. There are two way to connect the serial console:

- RS232 on connector X13
- Virtual console over USB

To use the RS232 connection, connect the first RS-232 port of your target system using to a COM port of your PC or a USB-to-RS232 converter using a null modem cable.

For a working connection, 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 1], [▶ Figure 2] below. If you received your system as part of a **starter kit**, this kit should also contain a cable to be used for this connection.

| 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) |

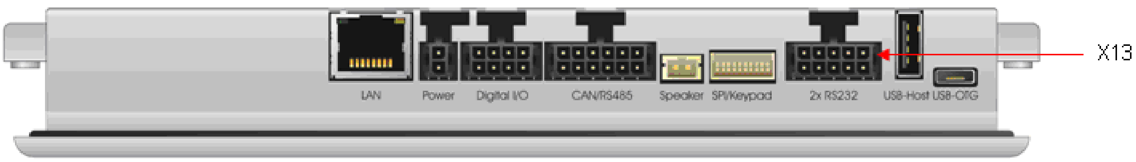



Figure 1: Location of the X13 connector

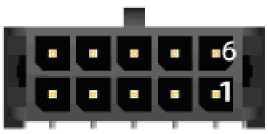



Figure 2: Pinning of the X13 connector

To use the serial console provided over USB, connect a Micro-USB cable to the USB-OTG connector of the target. When this USB cable is connected to a Windows PC, a driver is installed and a new COM port is created. Its name can be seen in the device manager.

**Note:** Although the serial connection over USB is easy to setup, there are some disadvantages over the RS232 connection: The output of the bootloader and the boot messages are not shown. The first thing you see is the login shell. This way it is not usable for system updates.

With the serial connection set up start 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

If you are using the RS232 connection, you should see debug messages in the terminal from the very first moment when the target is powered. After the boot process has finished, you will see the Linux login shell:

```
Garz & Fricke Yocto BSP (based on Poky) @VERSION@ santaro /dev/ttymxc0

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:

```
root@santaro:~# 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/tftp** 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 /srv/tftp/
```

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 SFTP

You can exchange files between the host system and the target system using an SFTP (Secure FTP) client on the host system. Simply choose your favourite SFTP client (e.g. FileZilla) 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 Yocto Linux systems are usually started with start scripts. This is a very common technique on Linux systems. Yocto uses the **/etc/init.d/rc** script for this purpose. This script is run by the **init** process after parsing the **/etc/inittab** file:

```
[...]
# The default runlevel.
id:5:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
[...]
```

As the comments in the file tell, the first script to be run on boot is **/etc/init.d/rcS**, which executes all start scripts in **/etc/rcS.d**. Afterwards, the default runlevel (5) is entered, which makes the start scripts in **/etc/rc5.d** being executed.

All scripts starting with the character **S** are executed with the argument **start** appended, while all scripts starting with the character **K** are executed with the argument **stop** appended. Furthermore, the naming convention states that the **S**/**K** character is followed by a number which determines 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/rcS.d/S04udev -> /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

### 4.1.2 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.

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

- **/etc/rc5.d/S02dbus-1 -> /etc/init.d/dbus-1**

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

### 4.1.3 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/rc5.d/S09sshd -> /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

### 4.1.4 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/rcS.d/S05modutils.sh -> /etc/init.d/modutils.sh**

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.

### 4.1.5 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.

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 [▶ 4.1.6 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 [▶ 4.1.7 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.

### 4.1.6 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 [▶ 4.1.5 Network initialization]) and touch configuration.

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

- /etc/rcS.d/S24sharedconf -> /etc/init.d/sharedconf

### 4.1.7 Garz & Fricke system configuration

The /etc/init.d/sharedconf script (see [▶ 4.1.6 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 | list | <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 a 2.Ethernet is present, it may be configured via serial2, mac2, etc.
```

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.

The 'name' set with sconfig is also used as hostname for the device. It defaults to GFMM<serial number>.

**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
```

### 4.1.8 Kernel command line

The kernel command line can be used to change some kernel features.

**Note:** Be careful changing the command line, as it can easily break the booting process of your device. If booting fails after those changes, you will need to boot into **Flash-N-Go System** and correct the settings. In this case, please refer to the **Flash-N-Go System** manual.

To change the kernel command line, the boot partition needs to be mounted.

```
mount /dev/mmcblk0p2 /mnt
```

Calling

```
nano /mnt/boot.cfg
```

will bring up the editor with the boot configurations.

The boot configuration normally looks similar to this:

```
load -b 0x12000000 -p config config.xml
load -b 0x13040000 -o logo.png
devtree -b  0x05:0x61:0x13000000 imx6-santoka-dl.dtb
devtree -b  0x07:0x61:0x13000000 imx6-santaro-dl.dtb
devtree -b  0x07:0x63:0x13000000 imx6-santaro-q.dtb
load linuximage
exec "console=ttymxc0,115200 root=/dev/mmcblk0p3 xmlram=0x12000000 logo=0x13040000"
```

The last line is the kernel command line. Options can be added to the end.

**Example 1:** Enable the SPI interface on the keypad connector:

```
exec "console=ttymxc0,115200 root=/dev/mmcblk0p3 xmlram=0x12000000 logo=0x13040000
    ↪ keypad=gpio,spi"
```

# 5 Add-On Packages

Starting with Garz & Fricke Yocto Linux BSP Release 19 some packages are not integrated into the prebuilt OS-images provided by Garz & Fricke but instead shipped separately as installable RPM-packages. This is done to not burden the prebuilt OS-images with larger packets and to allow choices, which particular version of a package to install on a specific device.

Such RPM packages can be transfered to the device using any protocol or storage-medium supported by the system and afterwards installed using the following command:

```
$ rpm -i  <name of RPM-package-file>
```

Please note that RPM-packages, like the prebuilt OS-images, are provided for each Garz & Fricke system separately and can be installed on only the particular system they are intended for, e.g. an RPM-package provided for VINCELL cannot be installed on a SANTARO or vice versa.

Deinstallation of RPM-packages can be performed likewise:

```
$ rpm -e  <name of RPM-package>
```

For deinstallation you can usually provide abbreviated package names instead of filenames or package names including full version numbers.

The following sections assume that RPM-packages are stored on your TFTP-server and will be uploaded to the device using TFTP. You may use other protocols or storage-media to transfer the RPM-packages to your device, of course.

## 5.1 Chromium

Starting with Yocto Release 19 Garz & Fricke provides the open-source Chromium browser as installable RPM-package.

This package is intended to be used for HTML-based applications starting automatically during system boot, which won't display the regular browser GUI, i.e. URL-bar, navigation buttons, etc.

### 5.1.1 Installation

The prebuilt Yocto OS-images provided by Garz & Fricke usually start a small QT-based demo-application automatically on system boot. As the Chromium package will install the browser to start automatically on system boot, as well, our default demo application must be disabled first:

```
$ update-rc.d -f qt4-guf-demo remove
```

Note: This will only remove the symlinks responsible for starting the qt4-guf-demo during system boot. The application itself still remains on the device and may still be started from the command-line.

Now the Chromium package may be uploaded and installed on the device. The following example assumes that you use a TFTP-Server on your Host-PC and have copied the RPM-package to its TFTP-Root folder, already. Of course you may use FTP, SFTP, USB-Sticks, SD-Cards or other protocols or storage-media instead. You must change the IP-address in the following example to the IP-address of your Host-PC and may have to change the name of the RPM-package you want to install (Garz & Fricke will eventually supply different Chromium versions for each platform).

```
$ export TFTP=172.20.9.107
$ curl tftp://$TFTP/GUF-Yocto-21.0-r5061-0-SANTARO/SANTOKA-chromium-42.0.2311.90-r0.
    ↪ rpm >/tmp/chromium.rpm
$ rpm -i /tmp/chromium.rpm
$ sync
```

This may take a moment and shortly afterwards Chromium should already start automatically. After installation Chromium will always start automatically on each system boot.

Note: Installing multiple different versions at the same time on a system is currently not supported, i.e. if you want to install a different version than you already may have installed on your device, you must deinstall the previous version first.

#### 5.1.2 Deinstallation

Chromium can easily be deinstalled using rpm. Please note that it must be currently running on the device otherwise the rpm-command will fail.

```
$ rpm -e chromium
```

#### 5.1.3 Manual Start/Stop of Chromium

You can start and stop Chromium during runtime with the following commands:

```
$ /etc/init.d/chromium stop
```

```
$ /etc/init.d/chromium start
```

#### 5.1.4 "Kiosk"-Mode

For normal application use Chromium is configured to start in so-called "kiosk"-mode. This means that it runs in full-screen mode without displaying the regular browser GUI, e.g. navigation buttons, URL-bar, access to the Chromium menu, etc. Only your webpage and, if necessary, scroll-bars are displayed.

For development or testing purposes you may want to use Chromium in normal-mode showing the navigation bar. You can temporarily do so by first stopping Chromium, as described above, and then calling it manually with any URL(s) and command-line options you like, e.g.:

```
$ /etc/init.d/chromium stop
$ Chromium http://www.google.com
```

This will temporarily re-start Chromium with the full GUI. Note that upon reboot of the system it will automatically start in Kiosk-mode again.

#### 5.1.5 Configuration

**URL:** The Chromium packages shipped by Garz & Fricke load the Garz & Fricke webpage by default. This is intended for first demonstration purposes only, of course, and may be changed to a different URL of your choice with the following command:

```
$ echo "<Your URL>" >/etc/chromium.conf
$ sync
```

E.g. to re-configure the Garz & Fricke website as default URL to be opened, execute:

```
$ echo "http://www.garz-fricke.com" >/etc/chromium.conf
$ sync
```

**Command-Line Options:** If you want Chromium to start with different command-line options than we have configurated by default, please modify the file **/etc/init.d/chromium** accordingly, e.g. you may remove the **–kiosk** command-line option if you want Chromium to automatically start in normal-mode with the navigation-bar enabled.

Supplying command-line options to the **/etc/init.d/chromium start** call is not supported. You may test command-line options by starting Chromium manually, though, e.g.:

```
$ /etc/init.d/chromium stop
$ Chromium --show-fps --kiosk --incognito http://www.garz-fricke.com
```

A list of all Chromium command-line options is available here:

- http://peter.sh/experiments/chromium-command-line-switches/

Not all command-line options are supported by all versions or builds of the Chromium browser, though, and Chromium does not complain about command-line options it does not recognize, but just ignores them silently, instead.

**Chromium Settings:** If you want to change Chromium settings, install Chromium extensions, use the Chromium developer tools, etc. you must (temporarily) start Chromium in normal-mode. These features and tools are not available in kiosk-mode

**Chromium Apps and Extensions:** When started in normal-mode you may install Chromium extensions via the Chromium menu. Please note, though, that the Chromium-Webstore, which is used to find, download and install extensions, is shared with the Chrome-Webstore you may know from the Desktop Google Chrome browser.

Chromium is the open-source part of Chrome and does not support all functions of the full Chrome browser, which unfortunately is not available for Linux-ARM systems. Due to the larger market-share of the regular PC-Chrome browser, most apps and extensions will probably be tested only on this browser and platform and may not work on the Chromium browser of Garz & Fricke devices, e.g. while the "Virtual Keyboard"-extension from xonTAB.com works on Garz & Fricke devices and is in fact currently installed by default in the Garz & Fricke Chromium package (see below), e.g. the "Google Input Tools" do not work properly on our devices.

Therefore please understand that some apps and extensions you may want to install, may fail to work on our devices.

To install the same set of extensions (or Chromium preferences) on multiple devices, the easiest way is to configure one single device as desired and then take a "snapshot" of the device's **/home/root/.chromium**-folder, which can then be unpacked on any other device desired.

### 5.1.6 Soft-Keyboard

As already briefly mentioned, the Chromium RPM-packages provided by Garz & Fricke currently come with the free "Virtual Keyboard"-extension from xonTAB.com pre-installed, because there is currently no system-wide soft-keyboard available in our Yocto-BSP, which can be used in the Chromium kiosk-mode.

More information on the xonTAB.com "Virtual Keyboard" Chromium extension can be found on the manufacturer web-page: ▶ http://xonTAB.com.com/Apps/VirtualKeyboard

This extension allows you to try regular web-pages without modification. This soft-keyboard will not work in the Chromium URL-bar or in any Chromium dialogs that you may open, though. It is a Javascript-based extension only intended to enter text into forms of web-pages (but may even have issues with some web-pages).

To still easily allow opening different web-pages for testing, the keyboard is configured to include an URL-button, by default. If you want to use this soft-keyboard with your HTML-application, you may want to disable this URL-button on the options-page of the extension. You can also enable/disable different international keyboard layouts (by default English and German layouts are enabled), change the soft-keyboard to a floating window, etc.

For your final HTML-based application you may want to replace the xonTAB.com "Virtual Keyboard" with a different Chromium extension providing a soft-keyboard, use other third-party Javascript-based keyboards embedded directly in your web-page without any Chromium extensions being installed, or provide your own soft-keyboard via Javascript, HTML5, etc.

Quite a lot of free third-party Javascript-based soft-keyboard to be integrated in web-pages may be found here:
- http://mottie.github.io/Keyboard

Using the Chromium URL-bar or text-input fields in Chromium dialog boxes is only supported via external USB-keyboards.

# 6 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.

## 6.1 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.

## 6.2 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/SANTOKA:

| TTY device | Hardware function |
|---|---|
| /dev/ttymxc0 | RS-232 #1 (X13) |
| /dev/ttymxc1 | RS-232 #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 **sources/meta-guf/meta/recipes-guf/ltp-guf-tests/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 **sources/meta-guf/meta/recipes-guf/mdbtest** for a full source code example.

## 6.3 Ethernet

If all network devices are properly configured as described in [▶ 4.1.7 Garz & Fricke system configuration] and [▶ 4.1.5 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

## 6.4 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
```

## 6.5 Keypad connector

The so called keypad connector is a general purpose connector. It can be used in different modes. The mode is selected using the kernel command line. See [▶ 4.1.8 Kernel command line] how it can be modified.

The actual pin mapping is described in the hardware guide for your device.

There are three available functions.

**GPIO:**

```
keypad=gpio
```

All pins are available as gpios. You can access them in usermode using the sysfs entries:

```
/sys/class/gpio/keypad_pin<3-18>
```

See [▶ 6.1 Digital I/O] for usage.

**I2C:**

```
keypad=i2c
```

The i2c 1 interface is mapped to the keypad connector pins 11 and 12. It is available in usermode as:

```
/dev/i2c-1
```

See [▶ 6.7 I2C] for usage.

**SPI:**

```
keypad=spi
```

The spi interface is mapped to the pins 12 till 18. The interface is available in usermode as:

```
/dev/spidev0.0
/dev/spidev0.1
/dev/spidev0.2
```

See [▶ 6.6 SPI] for usage.

**Combination:** The options can be combined to use multiple functions at the same time, for example:

```
keypad=gpio,i2c,spi
```

will enable all three functions. This leads to the following mapping:

- pins 3-10 are used as gpios
- 11 and 12 are used for i2c

- 13-18 are used for spi

**Default:**

The default setting is:

```
keypad=gpio,i2c
```

## 6.6 SPI

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

- Writing a kernel SPI device driver and accessing this driver from user space by using its interface.
- Accessing the SPI bus via the Linux kernel's **spidev** API directly.

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

- **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 kernel directory under:

- **Documentation/spi/spidev**

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

- **Documentation/spi/spidev_test.c**

The header for spidev is available under:

- **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.

## 6.7 I2C

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

- Writing a kernel I2C device driver and accessing this driver from user space by using its interface.
- Accessing the I2C bus via the Linux kernel's **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. It is located in the kernel directory under:

- **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 kernel directory under:

- **Documentation/i2c/dev-interface**

The header for i2c-dev is available under:

- **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.

## 6.8 CAN

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/SANTOKA 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 **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.

## 6.9 USB

There are two general types of USB devices:

- **USB Host**: the Linux platform device is the host and controls 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.

### 6.9.1 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.

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.

### 6.9.2 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**.

## 6.10 Display power

The display can be powered on or off by software. The value is exported as a virtual file in the sysfs under **/sys/class/graphics/fb0/blank**. It can be accessed using the standard file operations open(), read(), write() and close().

**Example 1:** Turn display off:

```
root@santaro:~# echo 4 > /sys/class/graphics/fb0/blank
```

**Example 2:** Turn display on:

```
root@santaro:~# echo 0 > /sys/class/graphics/fb0/blank
```

Please note that this value is not persistent, i.e. it gets lost when the device is rebooted.

## 6.11 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.0/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.0/brightness
255
root@santaro:~# echo 100 > /sys/class/backlight/pwm-backlight.0/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.

## 6.12 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/mmcblk1**, its single partitions are located under **/dev/mmcblk1pX** 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/mmcblk1p1
```

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 **udev** service. SD card partitions are mounted to mount point **/media/mmcblk1pX** with X being a positive number, and USB mass storage devices are mounted to mount point **/media/sdXY** with X=a..z and Y being 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/mmcblk1p1 /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
```

## 6.13 Temperature Sensor

Most Garz & Fricke systems are equipped with an on-board hardware temperature sensor. The sensor is a Texas Instruments LM73 connected via $I^2C$. To poll the currently measured temperature you can read the corresponding sysfs file in **/sys/class/hwmon/hwmon0/device/**.

For example:

```
root@santaro:~# cat /sys/class/hwmon/hwmon0/device/temp1_input
+38.0
```

The sensor generates an alert when the measured temperature exceeds the maximum temperature defined in **temp1_max**. The alert flag will be set to 0 (active low) and an interrupt is generated. The interrupt will trigger an event on the **temp1_alrt** sysfs entry, that can be catched using the **poll()** function.

Read the temperature alert flag (active low):

```
root@santaro:~# cat /sys/class/hwmon/hwmon0/device/temp1_alrt
1
```

To reset the temperature alert flag the currently measured temperature needs to be below the maximum temperature value. Then the flag can be reset by writing 1 to the **temp1_alrtrst** sysfs entry.

Reset the temperature alert flag:

```
root@santaro:~# echo 1 > /sys/class/hwmon/hwmon0/device/temp1_alrtrst
```

## 6.14 Touchscreen

The touchscreen device is used by the application framework (e.g. Qt) via the **Linux input subsystem** kernel API, i.e. its device node **/dev/input/event0**.

For resistive touchscreens, which require a calibration, the **tslib** library is used as an inter-layer. Garz & Fricke provides optimized signal filtering 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.

## 6.15 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 'PCM' 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
```

## 6.16 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
```

# 7 Building and running a user application

There are two general ways of building your own native application for the target: it can be built either manually, using the cross toolchain of the SDK, or integrated into the BSP, using Yocto and bitbake as a build system. Integrating the application into the BSP build process is the more complex way and useful only if you have modified the BSP and want to deploy a complete root file system image to the target anyway (see [▶ 8 Building a Garz & Fricke Yocto Linux system from source]). For a single application, using the BSP in its original state as provided by Garz & Fricke, the SDK is recommended to be used. The following sections describe how to build a simple **Hello Wold!** application using the SDK.

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 of this chapter.

## 7.1 SDK installation

The SDK contains the cross-toolchain and several files like headers and libraries necessary to build software for SANTARO/SANTOKA. It is available as download from the Garz & Fricke support website. The sdk was tested with Ubuntu 14.04 Desktop (amd64), other destributions might work but 32bit systems will not. In that case you have to build the sdk yourself using the bsp as described in [▶ 8.3 Building the BSP for the target platform with Yocto]

▶ http://support.garz-fricke.com/projects/Santaro/Linux-Yocto/Releases/Yocto-@VERSION@/sdk

The SDK file is a self-extracting archive that is supposed to run on Linux based machines. It is named something like **GUF-Yocto-21.0-r5061-0-sdk.sh**. Run this file on your development PC:

```
$sh <SDK location>/GUF-Yocto-21.0-r5061-0-sdk.sh
```

The installer will ask you if you want to install the SDK into a subfolder in **/opt**. Supposed this is what you want press the **y** key.

Example output:

```
Enter target directory for SDK (default: /opt/guf/GUF-Yocto-21.0-r5061-0-sdk):
You are about to install the SDK to "/opt/guf/GUF-Yocto-21.0-r5061-0-sdk". Proceed[Y/
    ↪ n]?
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

Now that the SDK is installed you may proceed and write your first application for the embedded device.

## 7.2 Simple command-line application

We will create a simple C++ "Hello World!" application that uses a Makefile and the supplied SDK. 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:

```
myapp: main.cpp
        $(CXX) ${CXXFLAGS} -o $@ $<
        $(STRIP) $@

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

It is necessary to setup your build environment so that the complier, headers and libraries can be found. This is done by "sourcing" a build environment configuration file. If the toolchain is installed in the default directory, this example compiles for the target system by typing

```
$ source
    ↪ /opt/guf/GUF-Yocto-21.0-r5061-0-sdk/environment-setup-santaro-santoka-guf-linux-gnueabi
$ make
```

in the **myapp** directory. The first line is needed only once as it configures the current shell and sets the necessary environment variables.

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.

You can find further information about how to build applications for Yocto-based platforms at:

- https://www.yoctoproject.org/docs/current/adt-manual/adt-manual.html

## 7.3 Qt-based GUI user application

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

```
$ cd ~
$ mkdir myqtapp
$ cd myqtapp
```

Create the empty files **main.cpp** and **myqtapp.pro**.

```
$ touch main.cpp myqtapp.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 **myqtapp.pro** as follows:

```
TEMPLATE = app
TARGET = myqtapp

QT = core gui widgets

SOURCES += \
    main.cpp
```

Setup the build environment.

```
$ source
    ↪ /opt/guf/GUF-Yocto-21.0-r5061-0-sdk/environment-setup-santaro-santoka-guf-linux-gnueabi
```

**Note:** The above command assumes that you have extracted the SDK in the default directory under **/opt/guf/GUF-Yocto-21.0-r5061-0-sdk**. If the SDK is located in a different directory, you have to change the directory accordingly.

Execute the following command to create a **Makefile** and build the binary in the **myqtapp** directory:

```
$ qmake myqtapp.pro
$ make
```

Now, there is the **myqtapp** executable in the **myqtapp** 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.

## 7.4 Using the Qt Creator IDE

Apart from compiling Qt applications on the command line, Qt Creator can be used as a comfortable IDE for developing, building and deploying applications for the target system. This section describes how to set up Qt Creator and how to compile and deploy a sample application.

### 7.4.1 Configuring Qt Creator

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.

Our tests were performed using a virtual machine installation of Ubuntu 14.04 Desktop (amd64). The following examples consider a Ubuntu 14.04 Desktop (amd64) installation. To install the Qt Creator from the terminal type

```
$ sudo apt-get install qtcreator qtcreator-plugin-cmake qtcreator-plugin-qnx
```

SFTP is used to deploy your program to the target device, thus SSH has to be installed as well:

```
$ sudo apt-get install ssh
```

After installation, Qt Creator needs some environment variables set. It is the same process as in the chapters [▶ 7.2 Simple command-line application] and [▶ 7.3 Qt-based GUI user application]. Open a console and type

```
$ source
    ↪ /opt/guf/GUF-Yocto-21.0-r5061-0-sdk/environment-setup-santaro-santoka-guf-linux-gnueabi
```

Now that this console session is prepared, start the Qt Creator:

```
$ qtcreator &
```

Now you need to configure the Qt Creator to use the correct toolchain and to deploy to the correct device. Open the **Tools->Options** dialog. We will configure the target device first.

On the left pane of the dialog open the **Devices** view. Add a new **Generic Linux Device** and configure IP and credentials according to your target settings. The IP address depends on the configuration, factory default is the static adress 192.168.1.1 but dhcp is also possible. Make sure that you can access the target as described in [▶ 3 Accessing the target system] with ssh or telnet before.

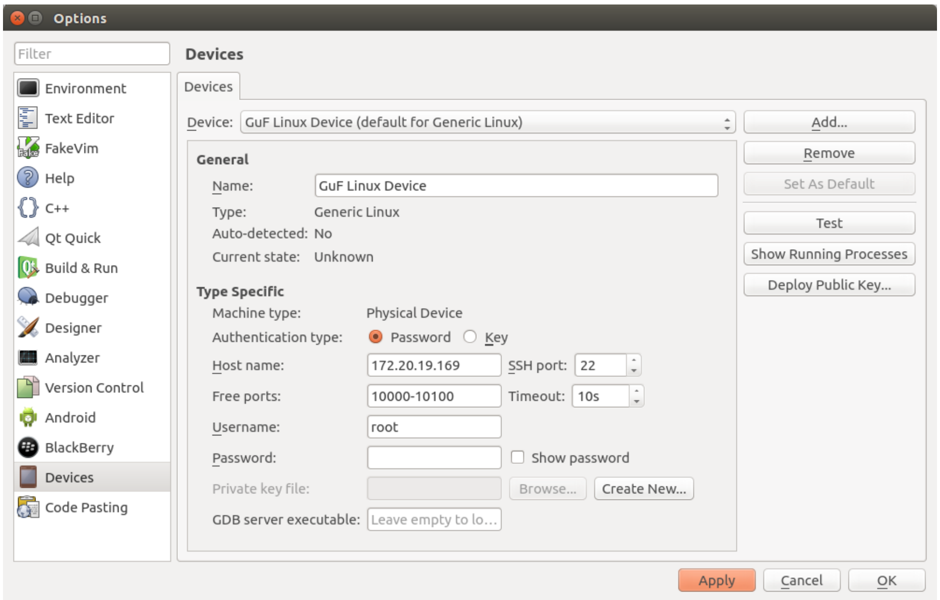


Figure 3: Qt Creator device options

You can press the **Test** button to test your configuration. The test dialog should display:

```
Device test finished successfully.
```

Now that the device is configured we need to set up the toolchain. This is done in the **Build & Run** pane. The first thing we want to add is the cross compiler in the **Compilers** section. The **Compiler path** is everything that is needed here.

```
/opt/guf/GUF-Yocto-21.0-r5061-0-sdk/sysroots/x86_64-gufsdk-linux/usr/bin/arm-guf-
    ↪ linux-gnueabi/arm-guf-linux-gnueabi-g++
```

The dialog should look similar to [▶ Figure 4].

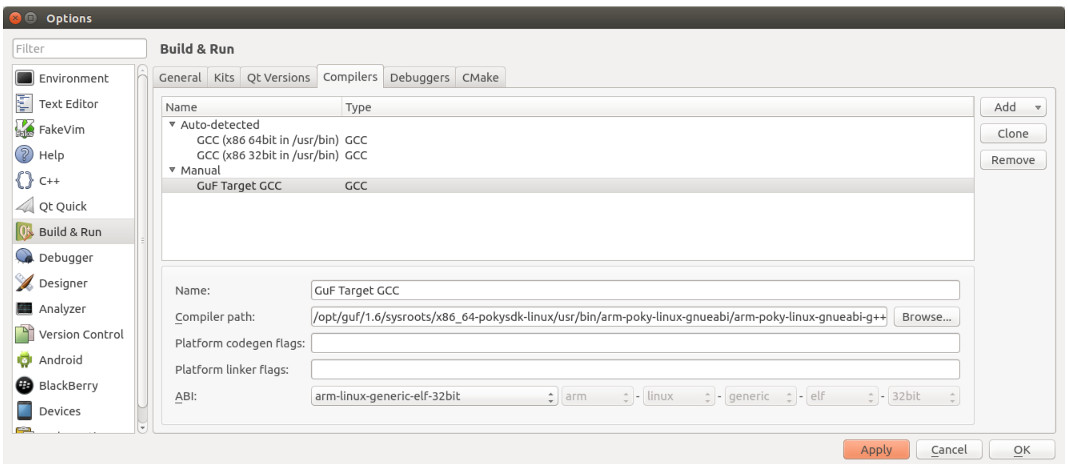


Figure 4: Qt Creator compiler options

The next step is checking the version the **Qt Versions** section. It should be set to

```
/opt/guf/GUF-Yocto-21.0-r5061-0-sdk/sysroots/x86_64-gufsdk-linux/usr/bin/qt5/qmake
```

as seen in [▶ Figure 5].

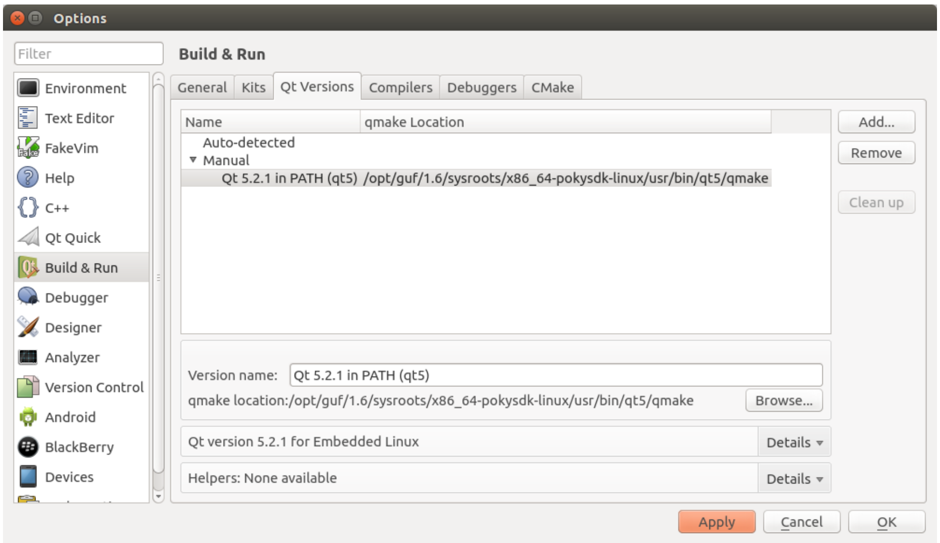



Figure 5: Qt Creator Qt Versions

In the last configuration step we combine the previous configuration steps to a kit in the **Kits** section. Add a new **Generic Linux Device** and set the options so that the previously created settings are used. The **Sysroot** setting needs to be set to something like

```
/opt/guf/GUF-Yocto-21.0-r5061-0-sdk/sysroots/santaro-santoka-guf-linux-gnueabi
```

and the **Qt mkspec** is set to

```
linux-oe-g++
```

Please note that the **Qt mkspec** will hide after setting.

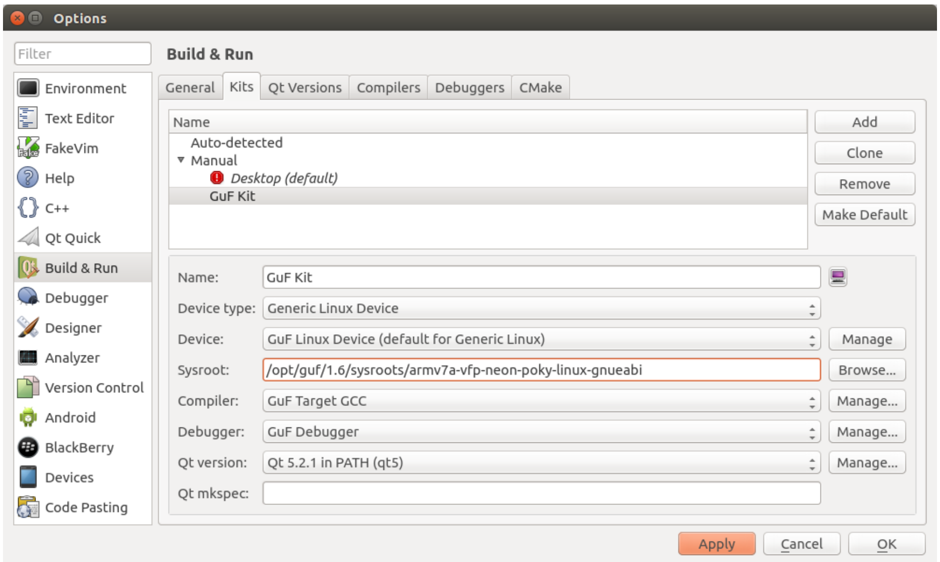



Figure 6: Qt Creator kits options

You can now begin to develop a Qt Application using the Qt Cerator.

### 7.4.2 Developing with Qt Creator

In this section we will create and deploy a simple Qt Quick Application. The application will be the default sample application that comes with the Qt Creator IDE.

To create a new project select **File->New File or Project...**. Make sure that in the top right corner of the wizard dialog the option **Embedded linux Templates** is selected. Choose an **Applications** project and select **Qt Quick Application**.

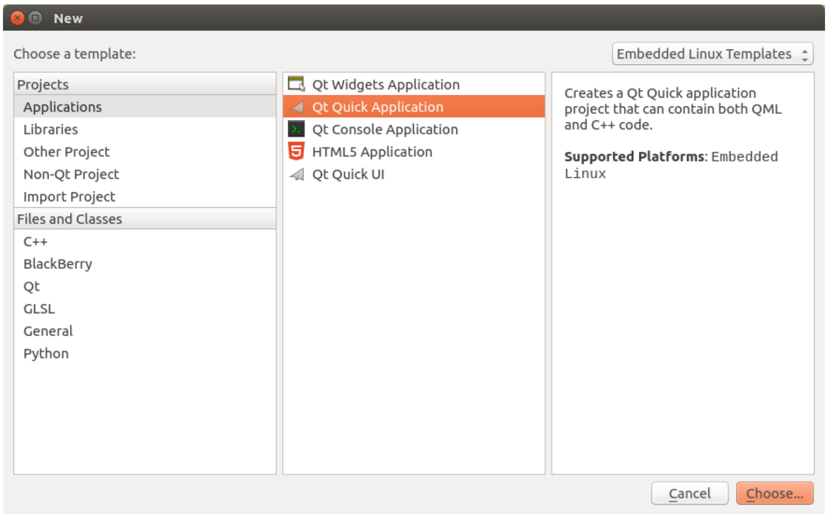



Figure 7: Qt Creator new project screen

Click on **Choose** and give your application a name. After a click on the **Next** button you can choose which component set you want to use. For this example we select **Qt Quick 2.0**. Another click on the **Next** button shows the Kits selection. You only need the kit that you created in the previous section.

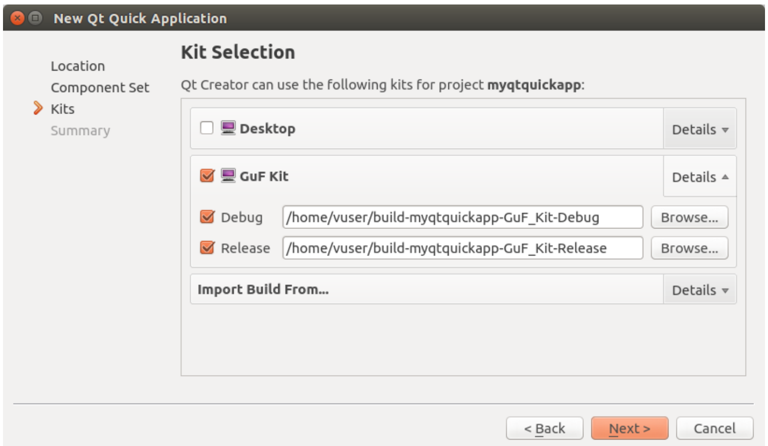



Figure 8: Qt Creator kit selection

After finishing the wizard you should see the opened **main.qml** and the project files.

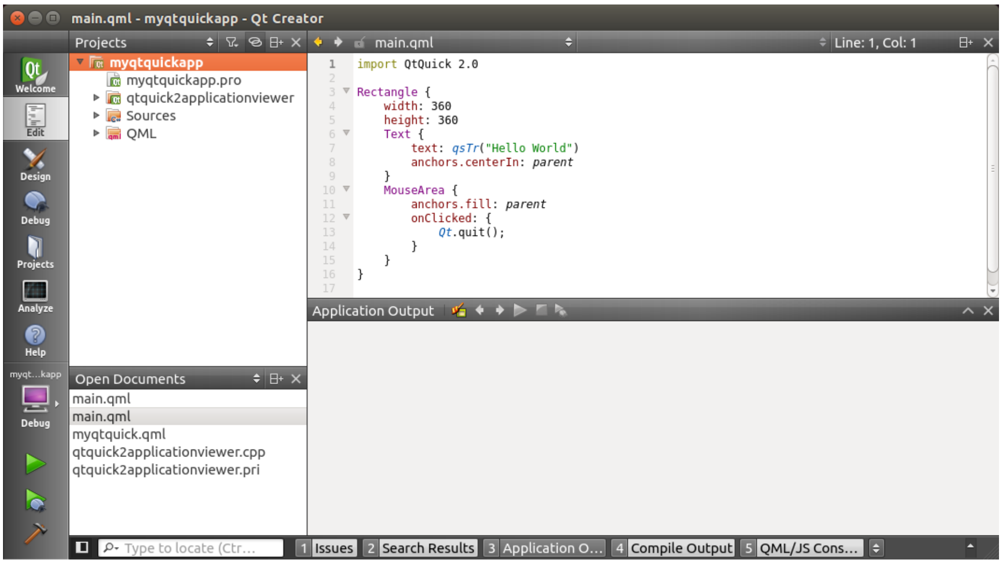



Figure 9: The first Qt Quick application in Qt Creator

You can now build and deploy the application by clicking the **Run** button (the play button in the left bottom corner) or using the shortcut **Ctrl-R**. Now the Qt Creator builds your application and automatically deploys it to the device. On the device, you should see an application showing **Hello World** now.

For further information on how to use Qt Creator and how to program Qt applications see the online Qt documentation:

- http://qt-project.org/doc/qtcreator-3.0/index.html
- http://qt-project.org/doc/

## 7.5 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:~# nano /etc/init.d/myapp
```

Change the contents of this file as follows:

```
#!/bin/sh

[ -f /etc/profile.d/tslib.sh ] && . /etc/profile.d/tslib.sh
[ -f /etc/profile.d/qt5-touch-config.sh ] && . /etc/profile.d/qt5-touch-config.sh

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**.

Make **/etc/init.d/myapp** executable:

```
root@santaro:~# chmod a+x /etc/init.d/myapp
```

Create startlinks in **/etc/rc*.d/**:

```
root@santaro:~# update-rc.d myapp defaults 95 5
```

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:~# update-rc.d -f qt4-guf-demo remove
```

After system reboot your application will start automatically.

## 7.6 Configuring the Qt Webkit demo

The Linux BSP for SANTARO/SANTOKA 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 -- --no-scrollbars file:///usr/share/website/
		  ↪ 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 [▶ 7.5 Autostart mechanism for user applications].

# 8 Building a Garz & Fricke Yocto Linux system from source

This chapter describes how to build a Yocto based 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.

## 8.1 General information about Garz & Fricke Yocto Linux systems

Garz & Fricke uses the **Yocto Project** build system for building embedded Linux systems for their platforms by providing a Board Support Package (BSP). The Yocto Project is lead by the Linux foundation with the aim to produce tools and processes to create embedded Linux distributions.

The Yocto project includes a configurable build system specializing in building embedded Linux systems. This chapter contains information about the handling of Linux with **Yocto** and Yocto based toolchains for Garz & Fricke systems. For further information regarding the **Yocto Project** please refer to the official **Yocto website**:

- https://www.yoctoproject.org

Documentation regarding several Yocto topics can be found at:

- https://www.yoctoproject.org/documentation/current

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

- autoconf
- automake
- build-essential
- dblatex
- docbook-utils
- fop
- libglib2.0-dev
- libsdl1.2-dev
- libtool
- make
- xmlto
- xsltproc
- xterm
- git
- texinfo
- chrpath
- python-dev
- python3-dev

See also:

- https://www.yoctoproject.org/docs/1.6/ref-manual/ref-manual.html#required-packages-for-the-host-development-system .

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

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

To install all the above packages, type:

```
$ sudo apt-get install autoconf automake dblatex docbook-utils fop libglib2.0-dev
    ↪ libsdl1.2-dev libtool make xmlto xsltproc xterm git texinfo chrpath python-dev
    ↪  python3-dev
```

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. When building a BSP with Yocto the toolchain will be supplied and built alongside with the target system. There is no need to install a GNU Compiler Collection (GCC) host and cross toolchain separately.

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 two steps, covered in the following chapters:

- Downloading and Installing a Yocto BSP [▶ 8.2 Download and install the Garz & Fricke Yocto BSP]
- Building the BSP for the target platform [▶ 8.3 Building the BSP for the target platform with Yocto]

## 8.2 Download and install the Garz & Fricke Yocto BSP

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

- **GUF-Yocto-21.0-r5061-0.tar.bz2**

This archive can also be found on the Garz & Fricke support website:

▶ http://support.garz-fricke.com/projects/Santaro/Linux-Yocto/Releases/

This archive contains the necessary files to build a cross toolchain and a Yocto based target image. To install the Garz & Fricke BSP, simply extract the file.

For example:

```
$ cd ~
$ mkdir yocto
$ cd yocto
$ cp /media/sda1/Tools/GUF-Yocto-21.0-r5061-0.tar.bz2 .
$ tar -xvf GUF-Yocto-21.0-r5061-0.tar.bz2
```

If everything went right, we have a **GUF-Yocto-21.0-r5061-0** directory now, so we can change into it:

```
$ cd GUF-Yocto-21.0-r5061-0
```

## 8.3 Building the BSP for the target platform with Yocto

In the Yocto directory, the following command selects the platform to be built:

```
$ MACHINE=santaro-santoka source setup-environment build-santaro-santoka
```

After that the shell should have changed the current working directory to the platform specific build directory **build-santaro-santoka**. To build packages and complete images, the Yocto build tool **bitbake** is used. It is documented in the official Yocto documentation and online in the **OpenEmbedded Bitbake Manual**:

▶ http://www.yoctoproject.org/docs/1.6.1/bitbake-user-manual/bitbake-user-manual.html

Yocto builds the images from build descriptions called **recipes**. The recipe to build the Garz & Fricke Linux image is called **guf-image**. To build this image, call:

```
$ bitbake guf-image
```

This step automatically downloads all necessary parts from the web, builds the native toolchains as well as the target binaries. As this step is fairly complex, and many packages will be created and compiled, it takes quite some time. On our development machines, a complete build takes approximately 60 minutes.

After the build has finished, the images will be located in your build directory under **tmp/deploy/images/santaro-santoka**. In this directory, several files should be located. The most important ones are the last kernel build (**uImage-santaro-santoka.bin**) , the devicetree files (**uImage-imx6-santaro-q.dtb, uImage-imx6-santaro-dl.dtb,**

**uImage-imx6-santoka-dl.dtb**) and the last target root file system (**guf-image-santaro-santoka.tar.bz2**). The latter is also often called **rootfs**.

Rather than files itself, these are symbolic links to the former build artifacts in the same directory. Every successive build of the image creates new artifacts with a recent timestamp in its name.

# 9 Deploying the Linux system to the target

A Garz & Fricke Yocto image can be installed on the system's internal eMMC flash memory using the Garz & Fricke **Flash-N-Go System**. This is a small RAM-disk-based Linux which is installed on your **SANTARO/SANTOKA** in parallel to the regular operating system. This chapter describes how to boot your device into **Flash-N-Go System** and how to use it to install your Yocto image.

## 9.1 Booting Flash-N-Go System

There are two ways of booting your device into **Flash-N-Go System**. If the device already has a working Yocto image installed, you can switch to **Flash-N-Go System** by issuing the following commands on the device console:

```
root@santaro:~# bootselect alternative
root@santaro:~# reboot
```

The device will reboot and show the **Flash-N-Go System** splash screen on the display. On the serial console (see [▶ 3.1 Serial console]), the command prompt should appear:

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

FLASH-N-GO:/
```

The change of the bootmode using the **bootmode** command is permanent, i.e. the next boot of the device will start **Flash-N-Go System** again, until the bootmode is set back to regular operation:

```
FLASH-N-GO:/ bootselect regular
```

Alternatively, the bootmode can be switched temporarily by pressing down and holding the bootmode switch **SW2** while the device is powered on. The location of **SW2** is shown in [▶ Figure 10].


Figure 10: Location of the SW2 switch

This method changes the bootmode only for a single boot. The next boot of the device (without **SW2** pressed) will boot the regular operating system again.

For more detailed information concerning the Garz & Fricke **Flash-N-Go System** please consult the **Flash-N-Go System Manual**.

## 9.2 Installing a Yocto image on the device

Garz & Fricke provides a shell script for installing a Yocto image on the device, called **fng-install.sh**. The files can be installed either locally on the device (e.g. a USB drive or an SD card) or remotely via TFTP. Regardless which solution you prefer, you will need the following files from your **deploy** folder (see [▶ 8.3 Building the BSP for the target platform with Yocto]):

- fng-install.sh
- uImage-santaro-santoka.bin
- guf-image-santaro-santoka.tar.bz2
- uImage-imx6-santaro-q.dtb, uImage-imx6-santaro-dl.dtb, uImage-imx6-santoka-dl.dtb

- boot-santaro-santoka.cfg

If you have not built the image yourself but rather want to install a Garz & Fricke prebuilt Yocto image, the filenames will differ slightly. The file extensions should be equal, though, so it should not be difficult to determine the correct files.

**Note:** The installation via **fng-install.sh** removes any previously installed regular operating system. If the installation fails for some reason, the device will always boot into **Flash-N-Go System** afterwards.

### 9.2.1 Over the network via TFTP

During development, the most comfortable way of installing the images on the device is by loading them over the network via TFTP. For this purpose, a TFTP server is needed on your development host machine (please see [▶ 3.4 Uploading files with TFTP] on how to install and configure it). The TFTP server directory has to be set to your **deploy** folder, e.g. **/home/user/yocto/GUF-Yocto-21.0-r5061-0/build-santaro-santoka/tmp/deploy/images/santaro-santoka/**. The ethernet connection on the device has to be configured as described in [▶ 4.1.7 Garz & Fricke system configuration], so that it can access the TFTP server.

The script can be loaded to the device and executed there via the **Flash-N-Go System** shell. Assuming, that your TFTP host has the IP address 192.168.1.100, type:

```
FLASH-N-GO:/ export TFTP=192.168.1.100; curl tftp://$TFTP/fng-install.sh > /tmp/a.sh;
    ↪  sh /tmp/a.sh
```

The above command loads the **fng-install.sh** script from your TFTP server to the **/tmp** directory of the **Flash-N-Go System** and executes it from there. During execution of the script, the Yocto image files will be loaded from the TFTP server and written directly to the eMMC flash memory.

The installation procedure will take some minutes. You can observe the output messages of the process on the terminal console. After successful installation the script returns to the Flash-N-Go prompt:

```
Update successful
FLASH-N-GO:/
```

### 9.2.2 From a local folder using an external storage device

If you do not have a network connection to your device, the **fng-install.sh** can be copied to an external storage device, e.g. a USB driver or an SD card, along with the Yocto images. Simply put all files into the same folder and insert the storage device into your SANTARO/SANTOKA.

The TFTP environment variable must not be set. Usually the variable is not set, so you do not have to worry about this. If you have tried using TFTP before, though, it probably contains your TFTP server IP address and has to be unset explicitly:

```
FLASH-N-GO:/ unset TFTP
```

To start the installation, simply call the script from the shell:

```
FLASH-N-GO:/ sh /mnt/mstick1/fng-install.sh
```

The installation procedure will take some minutes. You can observe the output messages of the process on the terminal console. After successful installation the script returns to the Flash-N-Go prompt:

```
Update successful
FLASH-N-GO:/
```

**Note:** The installation from a local folder requires Flash-N-Go System 4.0 or higher.

# 10 Securing the device

The default configuration of a Garz & Fricke device can be described as "developer friendly". This means, all services are available and activated. Depending on the final application, this might be either helpful or a security risk. Once the development has been finalized, we recommend a review of the required services and to disable all services and features which are not used.

**Note:** For the following list of security risks, no claim of completeness can be made. There may arise other risks or - on the opposite - limitations in the design of your application by enabling or disabling single services.

## 10.1 Networking (netfilter/iptables)

By default all network communication is allowed. Linux can be configured to block certain IP packets depending on its header (e.g. by port or by protocol) using **iptables**, which is basically a firewall. As this mechanism is very powerful and complex it is not documented here in detail. Please take a look at the following link for a basic introduction:

- http://help.ubuntu.com/community/IptablesHowTo

## 11 Related documents and online support

This document contains product and OS specific information. Additional documentation is available for the use of the bootloader.

| Title | File Name | Description |
| --- | --- | --- |
| GUF Yocto Release Notes | GUF-Yocto-21.0-r5061-0-SANTARO-release-notes.html | Contains details about the Yocto release: change history, known restrictions, performed tests, etc. |
| Flash-N-Go System User Manual | GF_Flash-N-Go_Manual-<version>.pdf | Contains relevant information about the BIOS, boot logo, display settings, etc. in the case that Flash-N-Go Boot is used as the bootloader. |
| RedBoot User Manual | GF_RedBoot_User_Manual_<revision>.pdf | Contains relevant information about the BIOS, boot logo, display settings, etc. in the case that RedBoot is used as the bootloader. |

Support for your Garz & Fricke embedded device is available on the Garz & Fricke website. You can find a list of the available documents, as well as their latest revision and updates for your system under the following link:

- 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.