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 prior to 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 [► 9 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

© Copyright 2012 by Garz & Fricke GmbH. All rights are reserved. Copies of all or part of this manual or translations into a different language may only be made with the prior written approval.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important hints</td>
<td>2</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2 Overview</td>
<td>5</td>
</tr>
<tr>
<td>2.1 The bootloader</td>
<td>5</td>
</tr>
<tr>
<td>2.2 The Linux kernel</td>
<td>5</td>
</tr>
<tr>
<td>2.3 The root file system</td>
<td>5</td>
</tr>
<tr>
<td>2.4 The partition layout</td>
<td>6</td>
</tr>
<tr>
<td>2.5 Further information</td>
<td>6</td>
</tr>
<tr>
<td>3 Accessing the target system</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Serial console</td>
<td>7</td>
</tr>
<tr>
<td>3.2 SSH console</td>
<td>7</td>
</tr>
<tr>
<td>3.3 Telnet console</td>
<td>8</td>
</tr>
<tr>
<td>3.4 Uploading files with TFTP</td>
<td>8</td>
</tr>
<tr>
<td>3.5 Uploading files with FTP</td>
<td>9</td>
</tr>
<tr>
<td>4 Services and utilities</td>
<td>10</td>
</tr>
<tr>
<td>4.1 Services</td>
<td>10</td>
</tr>
<tr>
<td>4.1.1 Udev</td>
<td>10</td>
</tr>
<tr>
<td>4.1.2 D-Bus</td>
<td>10</td>
</tr>
<tr>
<td>4.1.3 SSH service</td>
<td>11</td>
</tr>
<tr>
<td>4.1.4 Module loading</td>
<td>11</td>
</tr>
<tr>
<td>4.1.5 Network initialization</td>
<td>11</td>
</tr>
<tr>
<td>4.1.6 Garz &amp; Fricke shared configuration</td>
<td>11</td>
</tr>
<tr>
<td>4.1.7 Garz &amp; Fricke system configuration</td>
<td>12</td>
</tr>
<tr>
<td>5 Accessing the hardware</td>
<td>13</td>
</tr>
<tr>
<td>5.1 Digital I/O</td>
<td>13</td>
</tr>
<tr>
<td>5.2 Serial interfaces (RS-232 / RS-485 / MDB)</td>
<td>14</td>
</tr>
<tr>
<td>5.3 Ethernet</td>
<td>14</td>
</tr>
<tr>
<td>5.4 Real Time Clock (RTC)</td>
<td>15</td>
</tr>
<tr>
<td>5.5 SPI</td>
<td>15</td>
</tr>
<tr>
<td>5.6 I2C</td>
<td>15</td>
</tr>
<tr>
<td>5.7 CAN</td>
<td>15</td>
</tr>
<tr>
<td>5.8 USB</td>
<td>17</td>
</tr>
<tr>
<td>5.8.1 USB Host</td>
<td>17</td>
</tr>
<tr>
<td>5.8.2 USB Device</td>
<td>17</td>
</tr>
<tr>
<td>5.9 Display backlight</td>
<td>18</td>
</tr>
<tr>
<td>5.10 SD cards and USB mass storage</td>
<td>18</td>
</tr>
<tr>
<td>5.11 Touchscreen</td>
<td>18</td>
</tr>
<tr>
<td>5.11.1 tslib</td>
<td>19</td>
</tr>
<tr>
<td>5.12 Audio</td>
<td>19</td>
</tr>
<tr>
<td>5.13 SRAM</td>
<td>19</td>
</tr>
<tr>
<td>6 Building a Garz &amp; Fricke Yocto Linux system from source</td>
<td>21</td>
</tr>
<tr>
<td>6.1 General information about Garz &amp; Fricke Yocto Linux systems</td>
<td>21</td>
</tr>
<tr>
<td>6.2 Download and install the Garz &amp; Fricke Yocto BSP</td>
<td>22</td>
</tr>
<tr>
<td>6.3 Building the BSP for the target platform with Yocto</td>
<td>22</td>
</tr>
<tr>
<td>7 Deploying the Linux system to the target</td>
<td>23</td>
</tr>
<tr>
<td>7.1 Release deployment</td>
<td>23</td>
</tr>
<tr>
<td>8 Building a user application for the target system</td>
<td>25</td>
</tr>
<tr>
<td>8.1 SDK installation</td>
<td>25</td>
</tr>
<tr>
<td>8.2 SDK installation procedure</td>
<td>25</td>
</tr>
<tr>
<td>8.3 Non-GUI user application</td>
<td>25</td>
</tr>
<tr>
<td>8.3.1 Non-GUI user application outside from Yocto</td>
<td>26</td>
</tr>
<tr>
<td>8.4 Building a Qt based application</td>
<td>26</td>
</tr>
</tbody>
</table>
8.4.1 Qt-based GUI user application outside from Yocto 27
8.4.2 Configuring the Qt Creator IDE 28
8.4.3 Develop with the Qt Creator IDE 30

9 Related documents and online support 33

A GNU General Public License v2 34
A.1 Preamble 34
A.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 34
A.3 END OF TERMS AND CONDITIONS 37
A.3.1 How to Apply These Terms to Your New Programs 37

Document history concerning all platforms 39
1 Introduction

Garz & Fricke systems based on Freescale i.MX53 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 VINCELL, 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/Vincell/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 sources are not available anymore at the former location there should be a backup available at:

▶ 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).

VINCELL uses the bootloader RedBoot.

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 or GNOME. However, in practice most embedded Linux Systems are running only one GUI application and a desktop system generates useless overhead.

VINCELL 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 partitions of the backing-storage. The backing-storage type of VINCELL is NAND Flash. 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 VINCELL platform is:

- RedBoot: bootloader binary image
- FIS directory: XML configuration parameters
- Redundant FIS: mirrored XML configuration parameters
- logo.png: boot logo in PNG format
- kernel: kernel binary image
- rootfs: root file system

RedBoot can start the following Linux kernel image types:

- zImage: compressed image
- Image: uncompressed image

2.5 Further information

For readers who are not familiar 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:


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 U-Boot can be found at:

► http://www.denx.de/wiki/U-Boot

Information about the RedBoot can be found at:


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 download over Ethernet
- **FTP**: file upload and download over Ethernet

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

### 3.1 Serial console

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

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

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

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

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@VINCELL>:
```

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

```
$ exit
```

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

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

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

```bash
$ 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 '^]'.
gufboard login: root
Password: [Enter password]
root@VINCELL:~
```

### 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:

```bash
$ 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:

```bash
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:

```bash
$ 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:

```bash
$ sudo service xinetd restart
```
From now on, you can access files in this directory from the target.

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

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

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

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

### 3.5 Uploading files with FTP

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

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

4.1 Services

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

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

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

4.1.1 Udev

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

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

Additionally, udev is in charge of loading firmware 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/S03udev -> /etc/init.d/udev`

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

More information about udev can be found at:


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. For Qt, Garz & Fricke provides an example in its BSP under `local_src/common/qt4-guf-dbus`.

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

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

More information about dbus can be found at:


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

- [http://qt-project.org/doc/qt-4.7/intro-to-dbus.html](http://qt-project.org/doc/qt-4.7/intro-to-dbus.html)

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

4.1.3 SSH service

The ssh service allows the user to log in on the target system. Furthermore, 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/S04modutils.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@VINCELL:~ sconfig
```

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

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

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

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

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

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

```
root@VINCELL:~ sconfig dhcp off
root@VINCELL:~ sconfig ip 192.168.1.123
```
5 Accessing the hardware

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

5.1 Digital I/O

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

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

<table>
<thead>
<tr>
<th>sysfs file</th>
<th>Valid values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0, 1</td>
<td>The current level of the GPIO pin. The <code>active_low</code> flag (see below) has to be taken into account for interpretation.</td>
</tr>
<tr>
<td>direction</td>
<td>in, out</td>
<td>The direction of the GPIO pin.</td>
</tr>
<tr>
<td>active_low</td>
<td>0, 1</td>
<td>Indicates if the pin is interpreted active low.</td>
</tr>
</tbody>
</table>

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@VINCELL:~ cat /sys/class/gpio/keypad_pin7/active_low
0
root@VINCELL:~ echo out > /sys/class/gpio/keypad_pin7/direction
root@VINCELL:~ 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@VINCELL:~ cat /sys/class/gpio/keypad_pin12/direction
in
root@VINCELL:~ cat /sys/class/gpio/keypad_pin12/active_low
1
root@VINCELL:~ cat /sys/class/gpio/keypad_pin12/value
1
```

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

```c
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.

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

Most of the serial interfaces are exported as TTY devices and thus accessible via their device nodes under `/dev/ttytmxc<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 VINCELL:

<table>
<thead>
<tr>
<th>TTY device</th>
<th>Hardware function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/ttytmxc0</td>
<td>RS-232 No. 1 (X13)</td>
</tr>
<tr>
<td>/dev/ttytmxc1</td>
<td>RS-232 No. 2 / MDB (X13)</td>
</tr>
<tr>
<td>/dev/ttytmxc2</td>
<td>RS-485 (X39)</td>
</tr>
<tr>
<td>/dev/ttytmxc3</td>
<td>internal UART (X11)</td>
</tr>
</tbody>
</table>

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:

```c
#include <termios.h>

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

    /* Open port */
    fd = open("/dev/ttytmxc2", O_RDWR | O_SYNC);

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

    close(fd);
}
```

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

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

### 5.3 Ethernet

If all network devices are properly configured as described in [4.1.7 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](http://pubs.opengroup.org/onlinepubs/9699919799/functions/contents.html)
5.4 Real Time Clock (RTC)

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

```
root@VINCELL:~ hwclock --show
Fri Jun 1 14:51:14 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@VINCELL:~ date 2010.09.09-16:50:00
Thu Sep 9 16:50:00 UTC 2010
root@VINCELL:~ hwclock --systohc
```

5.5 SPI

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

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

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.

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

5.6 I2C

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

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

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.

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

5.7 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 VINCELL 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@VINCELL:~ canconfig can0 bitrate 125000
root@VINCELL:~ 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@VINCELL:~ ip link set can0 type can bitrate 125000
root@VINCELL:~ 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@VINCELL:~ cansend can0 -i 0x20 0xca 0xbe 0xca 0xbe
```

Example 6 shows how all CAN messages are read on `can0` using the `candump` tool:
A more detailed documentation of the CAN bus handling in the Linux kernel can be found in the documentation directory of the Linux kernel source tree.

5.8 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.

5.8.1 USB Host

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

- USB Mass Storage
- USB Keyboard

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

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

Further information about libusb can be found under:

▶ [http://libusb.sourceforge.net/api-1.0](http://libusb.sourceforge.net/api-1.0)

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

5.8.2 USB Device

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

- USB Serial Gadget

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

```
root@VINCELL:~ modprobe g_serial
```

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

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

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

```
root@VINCELL:~ cat /sys/class/backlight/pwm-backlight.0/brightness
255
root@VINCELL:~ 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@VINCELL:~ xconfig addattribute -p /configurationFile/variables/display/backlight
↪   -n level_ac -v 100
```

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

5.10 SD cards and USB mass storage

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

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

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

If the first partition on an SD card or a USB mass storage device already contains a file system when it is plugged into the device, it is mounted automatically by the udev service. SD card partitions are mounted to mount point /media//dev/mmcblk0 with X being a positive number, and USB mass storage devices are mounted to mount point /mnt/mstickX with X 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@VINCELL:~ mkdir /my_sdcard
root@VINCELL:~ mount /dev/mmcblk0p1 /my_sdcard
```

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

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

5.11 Touchscreen

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

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

Garz & Fricke ensured that signal filtering is already optimized for the touchscreens that are shipped with their products by choosing a suitable set of filters with suitable parameters in tslib. The filter configuration can be altered in the configuration file /etc/ts.conf in the target’s root filesystem. This should only be done if the user is familiar 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 configuration file or executing the `sync` command. Otherwise, the changes may get lost during a hard reset.

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

5.12 Audio

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

> http://www.alsa-project.org

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

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

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

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

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

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

```
root@VINCELL:~ amixer sset 'Playback' 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 conjunction 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 `adiotestssrc` and `alsasink` plugins:

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

5.13 SRAM

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

```
root@VINCELL:~ cat /proc/mtd | grep SRAM
```
Per default, the SRAM is located at /dev/mtd0.

A very common use of the SRAM in conjunction 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

```bash
root@VINCELL:~ flash_eraseall /dev/mtd0
root@VINCELL:~ mkfs.jffs2 /dev/mtdblock0
root@VINCELL:~ mount /dev/mtdblock0 -t jffs2 /mnt
```
6 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.

6.1 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 process to create embedded Linux distributions.

The Yocto project employs 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](https://www.yoctoproject.org)

Documentation regarding several Yocto topics can be found at:

▶ [https://www.yoctoproject.org/documentation/current](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
- libdl1.2-dev
- libtool
- make
- xsltproc
- xterm

See also ▶ [https://www.yoctoproject.org/docs/1.6/ref-manual/ref-manual.html#required-packages-for-the-host-development-system](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:

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

To install all the previous listed packages type:

```bash
$ sudo apt-get autoconf automake dblatex docbook-utils fop libglib2.0-dev libdl1.2-dev libtool
make xsltproc xterm
```

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:
6.2 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 VINCELL have to be extracted:

- Tools/GUF-Yocto-<version>-0.tar.bz2

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

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

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

For example:

```
$ cd ~
$ mkdir yocto
$ cd yocto
$ cp /<mountpoint of USB mass storage>/Tools/GUF-Yocto-<version>-0.tar.bz2 .
$ tar -xvf GUF-Yocto-<version>-0.tar.bz2
```

If everything went right, we have a GUF-Yocto-<version>-0 directory now, so we can change into it:

```
$ cd GUF-Yocto-<version>-0
```

6.3 Building the BSP for the target platform with Yocto

The in the Yocto directory select the platform and distribution. This is done using the following line

```
$ DISTRO=guf MACHINE=vincell source setup-environment build-vincell
```

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 compile it takes quite some time. In our development machines it takes 3 hours for a complete build.

After this build has finished the images will be located in the tmp/images/vincell directory. In this directory several files should be located.

The most important ones are the last kernel build and the last target root file system. The latter is also often called rootfs.

The kernel for VINCELL is named zImage-vincell.bin and the rootfs is named guf-image-vincell.ubi.

Rather than files itself these are symbolic links to the formerly build artifacts in the same directory. Every successive build of the image creates new artifacts with a recent timestamp the it’s name.
7 Deploying the Linux system to the target

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

- Development deployment
- Release deployment

7.1 Release deployment

When the development phase has finished, the kernel and the root file system can be written into the system’s NAND flash memory and executed from there.

In chapter [▶ 6.3 Building the BSP for the target platform with Yocto] we created two image files:

- `zImage-vincell.bin`
- `guf-image-vincell.ubi`

The same must be done with the `zImage-vincell.bin` and the `guf-image-vincell.ubi` files to your TFTP directory:

```bash
$ cp tmp/deploy/images/vincell/zImage-vincell.bin /tftpboot
$ cp tmp/deploy/images/vincell/guf-image-vincell.ubi /tftpboot
```

To write the images to the system flash memory, re-power your target and interrupt the boot loader RedBoot with Ctrl+C for having access to its console:

```bash
RedBoot>
```

You can list the current partition layout by typing:

```bash
RedBoot> fis list
... Read from 0x800c5008-0x800d5008 at 0xe0080000:

<table>
<thead>
<tr>
<th>Name</th>
<th>FLASH addr</th>
<th>Mem addr</th>
<th>Length</th>
<th>Entry point</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBoot</td>
<td>0xE0000000</td>
<td>0xE0000000</td>
<td>0x00080000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Redundant FIS</td>
<td>0xE0080000</td>
<td>0xE0080000</td>
<td>0x00040000</td>
<td>0x00000000 current</td>
</tr>
<tr>
<td>FIS directory</td>
<td>0xE00C0000</td>
<td>0xE00C0000</td>
<td>0x00040000</td>
<td>0x00000000 backup</td>
</tr>
<tr>
<td>logo.png</td>
<td>0xE0100000</td>
<td>0x80100000</td>
<td>0x00020000</td>
<td>0x80100000</td>
</tr>
<tr>
<td>kernel</td>
<td>0xE0120000</td>
<td>0x80100000</td>
<td>0x00A00000</td>
<td>0x80100000</td>
</tr>
<tr>
<td>rootfs</td>
<td>0xE0B20000</td>
<td>0x80100000</td>
<td>0x0BA00000</td>
<td>0x80100000</td>
</tr>
</tbody>
</table>
```

Remove the old kernel and root file system partition if a previous Linux system is installed:

```bash
RedBoot> fis delete kernel
[...]
RedBoot> fis delete rootfs
[...]
```

If a Windows CE was pre-installed remove the OS and FlashDisk partition:

```bash
RedBoot> fis delete OS
[...]
RedBoot> fis delete FlashDisk
[...]
```

There may exist other partitions, which most probably should be removed before installing Linux. The only essential partitions which must not be deleted are RedBoot, FIS directory, Redundant FIS and logo.png.

Now we can load the kernel image from the TFTP server to the target RAM:

```bash
RedBoot> load -r -v -b 0x80100000 -h 192.168.1.100 linuximage
```

If the image file has successfully been loaded we can create a kernel flash partition out of it:
RedBoot> fis create -b 0x80100000 -l 0x600000 -e 0x80100000 -r 0x80100000 kernel

If the system asks you to increment the length of the image due to the occurrence of bad blocks then select \textit{y}.

The same can be done with the root file system image:

RedBoot> load -r -v -b 0x80100000 -h 192.168.1.100 root.ubi

The creation of the root partition is a little different: After loading the file to the target RAM you have to determine the file size of the image. The address range is shown in the console output, e.g.:

\begin{verbatim}
 [...]
 Raw file loaded
 Entry point: 0x80100000, address range: 0x80100000-0x81ebffff
 [...]
\end{verbatim}

You can make sure that you have enough flash memory left using the command

RedBoot> fis free

\begin{verbatim}
 ... Read from 0x800c5da0-0x800d5da0 at 0xe0080000: 
. 0xe05e0000 .. 0x00000000
\end{verbatim}

The output can be read we have free buffer from address \texttt{0xE05E0000} to \texttt{0x00000000} - 1 = \texttt{0xFFFFFFFF}. Thus there should be \texttt{0x1FA20000} = 530710528 Bytes free memory left on the NAND flash memory. For Yocto images you can use the most of the free memory to for the root file system.

However, some space should be left for additional bad-blocks. For the length of the rootfs you may choose \texttt{0x1F000000}.

To create the new root file system partition on the NAND flash use:

RedBoot> fis create -b 0x80100000 -l 0x1F000000 -e 0x80100000 rootfs

If the system asks you to increment the length of the image due to the occurrence of bad blocks then select \textit{y}.

Finally, RedBoot must be configured to execute the images from flash memory. Change the RedBoot boot script as follows. You do not have to enter all the other settings again. Type a dot (~) after the timeout question to skip all following settings and confirm the update of the configuration:

RedBoot> fconfig

\begin{verbatim}
 Run script at boot: true
 Boot script:
 .. net
 .. load -r -v -b 0x80100000 -h <server_ip_address> linuximage
 .. exec -c "console=ttymx0 root=/dev/nfs rw nfsroot=192.168.1.111:/rootfs"
 Enter script, terminate with empty line
 >> fis load kernel
 >> exec -c "console=ttymx0 ubi.mtd=rootfs root=ubi0:vincell-rootfs rootfstype=ubifs"
 >> [ENTER]
 Boot script timeout (1000ms resolution): 1.
 Update RedBoot non-volatile configuration - continue (y/n) y
\end{verbatim}

If the images are written correctly and the configuration is changed accordingly, the system should boot from the flash memory now.
8 Building a user application for the target system

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

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

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

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

In all cases it is necessary to install the SDK supplied with the Yocto release for the Garz & Fricke platforms. The SDK includes necessary files like headers and libraries to build software for embedded devices. It also includes the toolchain to build binaries for ARM-based platforms.

The SDK is available as download from the Garz & Fricke support website.

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

8.1 SDK installation

The SDK should include everythin neccessary to build software for the for the Garz & Fricke devices. The following section will describe the installation process for the SDK.

8.2 SDK installation procedure

The SDK file is a self extracting archive that is supposed to run on Linux based machines. The installation process is therefore pretty easy. It should be located in the sdk directory of the release and named somthing like GUF-Yocto-<version>-VINCELL-sdk.sh.

Run this file on your development PC.

$ <SDK location>/GUF-Yocto-<version>-VINCELL-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/1.6):
You are about to install the SDK to "/opt/guf/1.6". 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.

8.3 Non-GUI user application

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

We will now 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:

```cpp
#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 set up your build environment so that the compiler, 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/1.6/environment-setup-armv7a-vfp-neon-poky-linux-gnueabi
$ make
```

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

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


8.4 Building a Qt based application

The GUI user applications described here will display the message Hello World! on the device’s display.
8.4.1 Qt-based GUI user application outside from Yocto

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

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

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

```bash
$ touch main.cpp myqtapp.pro
```

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

```cpp
#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:

```bash
TEMPLATE = app
TARGET = myqtapp
QT = core gui
SOURCES += \\
    main.cpp

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

Setup the build environment.

```bash
$ source /opt/guf/1.6/environment-setup-armv7a-vfp-neon-poky-linux-gnueabi
```

**Note:** The above command assumes that you have extracted the SDK in the default directory under `/opt/guf/1.6`. 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:

```bash
$ cmake 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.
8.4.2 Configuring the Qt Creator IDE

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

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

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

- Tools/qt-creator-x86_64-opensource-3.0.1.bin

Our tests performed were done 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
```

To deploy your program to the target device SFTP us used. For this it is sufficient to install SSH.

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

After installation QtCreator needs some environment variables set. It is the same process as in the chapters [8.3.1 Non-GUI user application outside from Yocto] and [8.4.1 Qt-based GUI user application outside from Yocto].

Open a console and type

```
$ source /opt/guf/1.6/environment-setup-armv7a-vfp-neon-poky-linux-gnueabi
```

Now that this console session is prepared open the qtcreator

```
$ qtcreator &
```

You now need to configure the QtCreator 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.

![Figure 1: Qt Creator device options](image)
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 setup 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. It is only be neccessary to add the Compiler path.

```
/opt/guf/1.6/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++
```

and should look similar to Figure 2.

![Figure 2: Qt Creator compiler options](image)

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

```
/opt/guf/1.6/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake
```

as seen in Figure 3.

![Figure 3: Qt Creator Qt Versions](image)
In the last configuration step we combine the previous configuration 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 something like

```
/opt/guf/1.6/sysroots/armv7a-vfp-neon-poky-linux-gnueabi
```

and the Qt mkspec is set to

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

Please note that the Qt mkspec will hide after setting.

![Figure 4: Qt Creator kits options](image)

Yo can now begin to develop a Qt Application using the Qt Creator.

### 8.4.3 Develop with the Qt Creator IDE

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 on top right corner of the wizzard dialog the option Embedded linux Templates is selected. Choose a Applications project and select Qt Quick Application.
Click on Choose and give your application an 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 kit that you created in the previous section.

After finishing the wizzard you should see the opened main.qml and the project files.
You can now build and deploy 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 now see an application showing Hello World.

For further information how to use Qt Creator and how to program Qt applications see the online Qt documentations.

▶ http://qt-project.org/doc/qtcreator-3.0/index.html
▶ http://qt-project.org/doc/
9 Related documents and online support

This document contains product specific information. Additional documentation is available for the use of embedded operating systems, the related tool chain and the bootloader (BIOS).

<table>
<thead>
<tr>
<th>Title</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBoot User Manual</td>
<td>GF_RedBoot_User_Manual_Rnn.pdf</td>
<td>Contains relevant information about BIOS, boot logo, display settings, etc. in the case that RedBoot is used as BIOS.</td>
</tr>
<tr>
<td>U-Boot User Manual</td>
<td>GF_U-Boot_Manual_VINCELL-6.0.pdf</td>
<td>Contains relevant information about BIOS, boot logo, etc. in the case that U-Boot is used as BIOS.</td>
</tr>
<tr>
<td>Windows OS Manual</td>
<td>GF_WindowsCE_Manual_Vn.n.pdf</td>
<td>Contains information about Windows Embedded CE, the tool chain, the development environment Visual Studio, Garz &amp; Fricke tools, etc.</td>
</tr>
<tr>
<td>Linux OS Manual</td>
<td>GF_Linux_Manual_VINCELL-6.0.pdf</td>
<td>Contains information about Linux BSP, the tool chain, Qt, etc.</td>
</tr>
<tr>
<td>SAM-BA User Manual</td>
<td>GF_SAM-BA_Manual_VINCELL-6.0.pdf</td>
<td>Contains relevant information about the usage on ATMEL's SAM-BA tool with Garz &amp; Fricke devices in the case that an AT91SAM based platform is used.</td>
</tr>
</tbody>
</table>

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

▶ http://www.garz-fricke.com/vincell-download
A GNU General Public License v2

Version 2, June 1991

Copyright ©1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

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 copyright-related 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.
## Document history concerning all platforms

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.12.2013</td>
<td>0.0.1</td>
<td>Initial document release.</td>
</tr>
</tbody>
</table>