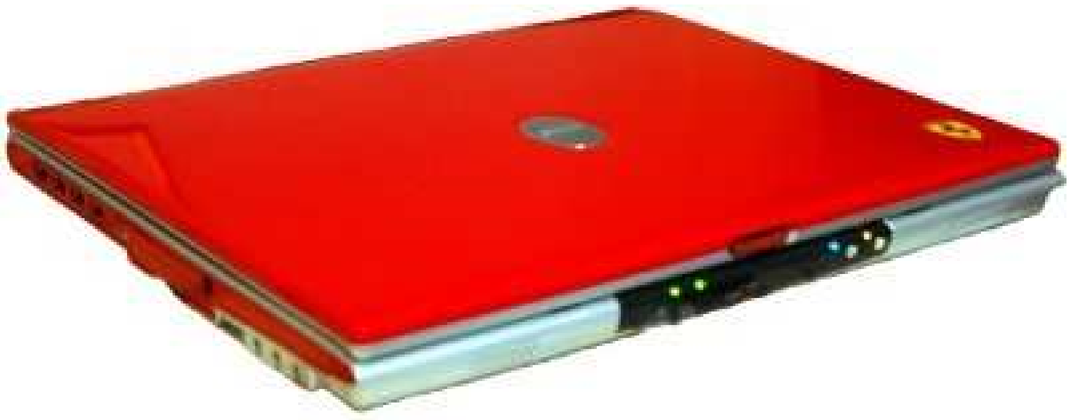
# FC4-x86_64 on the Acer Ferrari 3400LMi

by

Sven-Göran Bergh

sgb@systemasis.se

## Table of Contents



# 1 Introduction

This document is primarily designed for my own records for future use. However, it is always nice if it may help others to get started with Linux on their laptops. Here I describe the steps I have taken to get things to play on the Acer Ferrari 3400 LMi. However, general procedures may be useful for other hardware as well.

Likewise I have chosen to install Fedora Core 4 x86_64 and some details may be specific for that distribution, while most stuff should be similar on other distributions as well.

! ! ! WARNING ! ! !

However, I must warn those of you that still enjoy the luxury of an independent mind and free will. Stay away! Do not ever lay your hands on the Ferrari.

The same warning applies to the GNU/Linux software platform in general. It was several years ago my self defense was totally broken.

The combination is truly devastating! I have noticed strange things happening to me since I got *My Precious...*

Please feel free to comment on any topic or possible improvements in this document.

## 1.1 Quick reference guide

I've got most hardware working. Although, somethings need some extra tweaks. A summary may be found in the table below

| ***Hardware*** | ***Status*** | ***Details*** | ***Notes*** |
|---|---|---|---|
| CPU | OK | Mobile AMD Athlon 64 3000+ | No configuration needed. Even freqency scaling works out of the box. |

| ***Hardware*** | ***Status*** | ***Details*** | ***Notes*** |
|---|---|---|---|
| PCI Bridge | OK | VIA VT8237 [K8T800/K8T890 South] | No configuration needed. |
| Display | OK | 15” SXGA TFT (1400x1050) | No configuration needed. |
| Graphics card | OK | ATI Mobility Radeon 9700 128 MB, 8x AGP | For performance use ATI driver, see below. |
| RAM | OK | 512 MB DRAM (extended to 2GB) | No configuration needed. |
| Hard drive | OK | 80 GB Ultra ATA 100, IC25N080ATMR04-0, 4200 rpm | No configuration needed. |
| NIC (wired) | OK | Broadcom NetXtreme BCM5788 Gigabit Ethernet | No configuration needed. |
| NIC (wireless) | OK | Broadcom BCM4306 802.11b/g W-LAN | Use NdisWrapper, see comments below. |
| Modem | OK | VIA AC'97 56k Modem. | Works with minor issue, see below for comments. |
| DVD drive | OK | Matshita DVD-RAM UJ-825S (DVD/CD +/- R/W, DVD-RAM) | No configuration needed. |
| Sound | OK | VIA VT8233/A/8235/8237 AC97 Audio | No configuration needed. |
| Touchpad | OK | Synaptics SynPS/2 with 4 multi-buttons | No configuration needed. Multi-buttons works as well. |
| Special buttons | OK | Mail, web, P1, P2, volume, mute, etc. | See below for configuration. |
| PC-card | OK | Texas Instruments PCI4510 PC card/Cardbus | No configuration needed. |
| Bluetooth | OK | Cambridge Silicon Radio | No configuration needed, see comment below. |
| IEEE 1394 Firewire | OK | Texas Instruments PCI4510 IEEE-1394 | See below for configuration. |
| USB | OK | VIA, 4xUSB 2.0 | No configuration needed. |
| Infrared | OK | | See below for configuration. |

| ***Hardware*** | ***Status*** | ***Details*** | ***Notes*** |
|---|---|---|---|
| Card reader | OK | 5-in-1 (MMC, SM, SD, MS [Pro]) | No configuration needed. |

# 2 Installation

No special procedure is needed during the core installation of FC4 x86_64. Partition the hard drive as desired and install the components that you like. However, some packages will make life easier when configuring your new laptop. These are mentioned in the corresponding sections below and may be installed afterwards.

If you want to use the graphical mode of the installer (and who doesn't?) you need to disable the frame buffer by starting the installation with:

```
# linux nofb
```

Otherwise you will loose the display shortly after the installation enters graphical mode.

# 3 Graphics

The graphical hardware is properly identified and setup during the installation, so you will enjoy X11 right from the start. Another nice thing is that the dim display button, Fn-F6, works without any configurations. The ordinary Xorg X11 Mesa package (driver radeon) handles the graphics without any problems and will be sufficient for ordinary desktop use. However, this driver does not utilize the 3D hardware acceleration provided by your graphical card.

## 3.1 Install ATI driver

In order to optimize your graphical performance you need to install the proprietary “FireGL” driver provided by ATI. I grabbed the file `fglrx64_6_8_0-8.18.8-1.x86_64.rpm` from ATIs web site. The version might differ as time goes by, but you get the picture. Notice that ATI provides two similar linux drivers, one for Xfree86 and one for Xorg. I do not know whether the Xfree86 driver will work, but go for the Xorg driver, since that is what ships with FC4.

During the installation of this driver some files (`/usr/X11R6/lib/libGL.so.1.2` and `/usr/X11R6/lib64/libGL.so.1.2`) will conflict with the xorg-x11-Mesa-libGL package so you will need to force rpm to replace some files. Before you do so it might be a good idea to backup these files first.

```
#rpm -Ivh --replacefiles fglrx64_6_8_0-8.18.8-1.x86_64.rpm
```

Restart X and you should see something similar to the message below in your `/var/log/Xorg.0.log` file.

```
...
(II) LoadModule: "fglrx"
(II) Loading /usr/X11R6/lib64/modules/drivers/fglrx_drv.o
(II) Module fglrx: vendor="FireGL - ATI Technologies Inc."
        compiled for 6.8.0, module version = 8.18.8
        Module class: X.Org Video Driver
        ABI class: X.Org Video Driver, version 0.7
...
(II) ATI Proprietary Linux Driver Version Identifier:8.18.8
(II) ATI Proprietary Linux Driver Release Identifier: LGDr8.18g2
(II) ATI Proprietary Linux Driver Build Date: Oct 25 2005 10:32:21
(II) ATI Proprietary Linux Driver Build Information: autobuild-rel-
r6-8.18.1-driver-lnx-x86_64-221930
(--) Chipset MOBILITY RADEON 9600/9700 (M10/M11 4E50) found
```

You may use the ATIs configuration utility `fglrxconfig` to help you setup your `xorg.conf` file. It will ask you a whole bunch of detailed questions, but most default values work well.

Once X is configured and restarted you may test your success with the two utilities `fglrxinfo` and `fgl_glxgears`.

```
# fglrxinfo
display: :0.0  screen: 0
OpenGL vendor string: ATI Technologies Inc.
OpenGL renderer string: MOBILITY RADEON 9700 Generic
OpenGL version string: 1.3.5395 (X4.3.0-8.18.8)
```

Finally, in `/etc/yum.conf` add the line `exclude=xorg-x11-Mesa-libGL` to avoid conflicts when updating your system with yum.

## 3.2 TV-out

Works as expected, but remember that your external device must be connected before X starts. Otherwise it is not recognized and TV out will be disabled.

I found it most convenient to configure my xorg.conf file with a single ServerLayout section and run the external device as a clone of my main desktop. Otherwise, you always have the option of creating a second ServerLayout that is dedicated to running TV-out. In that case you need to supply `startx` with the desired layout, `startx -- -layout <ServerLayout>`.

You may find my `xorg.conf` in Appendix A.

It should be mentioned that the GATOS project is developing a TV-out driver for ATI Radeon cards. It is still in an early stage, but you may want to check it out at http://gatos.sourceforge.net/theater_out.php

# 4 Hard drive

No hassle what so ever, but my own reflection is that the hard drive does not match the “high end gear” profile of this laptop. When the laptop was released 120 MB drives was the latest of the greatest and 100MB drives were off the shelf goods in most stores. However, a smaller drive would have been ok at a higher speed, at least 5400rpm.

I am addicted to VMware and want extra of everything, size, speed, RAM, etc. Right now I am waiting to get my hands on the announced Seagate Momentus 160 GB (5400rpm).

# 5 Wireless NIC

Unfortunately there are no native drivers for the Broadcom BCM4306 802.11b/g WLAN chip on the Ferrari. The solution is NdisWrapper. As its name imply it is a wrapper for NDIS drivers, meaning that you may use an appropiate Windows driver instead.

## 5.1 Install & configure WLAN

1. First of all we need to make sure that the `wireless-tools` package is installed on our system:

   ```
   # rpm -q wireless-tools
   wireless-tools-28-0.pre4.3
   ```

   If not we need to install it:

   ```
   yum install wireless-tools
   ```

   or

   ```
   rpm -Uvh wireless-tools-28-0.pre4.3.x86_64.rpm
   ```

2. The next step is to download the latest version of NdisWrapper from `http://ndiswrapper.sourceforge.net`.
3. Unpack and install NdisWrapper according to the installation notes. The condensed procedure for me was pretty straightforward:

   ```
   # tar zxf ndiswrapper-1.5.tar.gz
   # cd ndiswrapper-1.5
   # make distclean
   # make
   # make install
   ```

4. Before we install the Windows driver into NdisWrapper we need to find a suitable driver. In order to do that we need to identifying the vendor ID and device ID:

```
# lspci
...
00:09.0 Network controller: Broadcom Corporation BCM4306
802.11b/g Wireless LAN Controller (rev 03)
...
# lspci -ns 00:09.0
00:09.0 Class 0280: 14e4:4320 (rev 03)
```

The vendor:device ID of the WLAN controller on this laptop is 14e4:4320. Armed with this information go to the list of known working devices on the NdisWrapper site. “Broadcom” and “14e4:4320” will guide you. Also remember that you are looking for a 64-bit driver, the ordinary 32-bit driver will not do. If you cannot find anything suitable, you may download the driver I use here (http://ferrari.databa.se/64-bit_broadcom_54g_drivers.zip)

5. Unpack the driver and install it into NdisWrapper:

```
# ndiswrapper -i netbc564.inf
```

6. Check that the installed driver is happy:

```
# ndiswrapper -l
Installed ndis drivers:
netbc564               driver present, hardware present
```

7. Looks great so add an alias for NdisWrapper in the file `/etc/modprobe.conf`:

```
alias wlan0 ndiswrapper
```

8. Since NdisWrapper act as a kernel module you need to rebuild your `/lib/modules/`uname -r`/modules.dep` file:

```
# depmod -a
```

9. Now you are ready to load your new kernel module:

```
# modprobe ndiswrapper
```

Take a look in the system log and you should see:

```
...ndiswrapper: driver netbc564 (,10/01/2002,3.70.17.5) loaded
...ndiswrapper: using irq 209
...wlan0: vendor: ''
...wlan0: ndiswrapper ethernet device 00:0b:6b:4c:42:17 using
driver netbc564, 14E4:4320.5.conf
...wlan0: encryption modes supported: WEP; TKIP with WPA; AES/CCMP
with WPA
```

10. Now the WLAN interface should show up:

```
# iwconfig
...
wlan0 IEEE 802.11g  ESSID:off/any
```

```
      Mode:Managed  Frequency:2.462 GHz  Access Point:
00:00:00:00:00:00
      Bit Rate=54 Mb/s   Tx-Power:25 dBm
      RTS thr:off   Fragment thr:off
      Encryption key:off
      Power Management:off
      Link Quality:0  Signal level:0  Noise level:0
      Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
      Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Most likely the Wireless Extensions version of your kernel and the `wireless-tools` package will differ since they evolve on parallel tracks. Normally it is safe to ignore the warning about different version.

11. OK, time to find out what's in the air:

```
# iwlist wlan0 scan
wlan0 Scan completed :
Cell 01 - Address: 00:12:DA:BF:AF:00
          ESSID:"Secret Net"
          Protocol:IEEE 802.11g
          Mode:Managed
          Frequency:2.447 GHz (Channel 8)
       Quality:0/100 Signal level:-84 dBm Noise level:-256 dBm
          Encryption key:on
          Bit Rate:1 Mb/s
          Bit Rate:2 Mb/s
          Bit Rate:5.5 Mb/s
          Bit Rate:6 Mb/s
          Bit Rate:9 Mb/s
          Bit Rate:11 Mb/s
          Bit Rate:12 Mb/s
          Bit Rate:18 Mb/s
          Bit Rate:24 Mb/s
          Bit Rate:36 Mb/s
          Bit Rate:48 Mb/s
          Bit Rate:54 Mb/s
          Extra:bcn_int=100
          Extra:atim=0
Cell 02 - Address: 00:E0:63:50:9A:31
          ESSID:"LSN"
          Protocol:IEEE 802.11b
          Mode:Managed
          Frequency:2.462 GHz (Channel 11)
       Quality:0/100 Signal level:-63 dBm Noise level:-256 dBm
          Encryption key:off
          Bit Rate:1 Mb/s
          Bit Rate:2 Mb/s
          Bit Rate:5.5 Mb/s
          Bit Rate:11 Mb/s
          Extra:bcn_int=100
          Extra:atim=0
```

Choose a network you want to connect to and set the ESSID of your W-LAN interface:

```
# iwconfig wlan0 essid LSN
```

Your WLAN interface should now be associated with the access point:

```
# iwconfig wlan0
wlan0 IEEE 802.11b  ESSID:"LSN"
      Mode:Managed  Frequency:2.442 GHz  Access Point:
00:E0:63:50:9A:31
      Bit Rate=1 Mb/s   Tx-Power:25 dBm
      RTS thr:off   Fragment thr:off
      Encryption key:off
      Power Management:off
      Link Quality:99/100 Signal level:-88 dBm Noise level:-256 dBm
      Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
      Tx excessive retries:137  Invalid misc:38171  Missed beacon:0
```

Now you are ready to start using your new interface. Most likely your first step will be to request IP setting from an DHCP server (`dhclient -1 wlan0`). For further information on wireless networking under Linux, please refer to the numerous HOWTOs on the internet. A good place to start is `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Wireless.html`.

## 6 Modem

FC4-x86_64 has the ALSA kernel module `snd_via82xx_modem` pre-compiled. That module is capable of handling the internal soft modem. Furthermore, this module is properly loaded at startup. However, the modem is still a winmodem so we need a corresponding user space daemon that utilize this ALSA support for our modem.

That part is provided by Smart Link (http://www.smlink.com). It is quite tricky to compile with 64-bit support, so I started to browse internet for a suitable RPM package for FC4-x86_64. Finally I found one at: http://www.xs4all.nl/~pjl/slmodemd.

### 6.1 Install & configure modem

Here are the basic steps to get it up and running:

1. Verfiy that the proper ALSA kernel module is loaded and recognizes the modem:

```
# aplay -l
...
card 1: modem [VIA 82XX modem], device 0: VIA 82XX modem [VIA 82XX
modem]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

2. Download the file `slmodemd-<version>_alsa-4.FC4.LC.x86_64.rpm` and install it:

   ```
   # rpm -Uvh slmodemd-<version>_alsa-4.FC4.LC.x86_64.rpm
   ```

   I use version 2.9.9e_pre1, but if you manage to find a later version use that one.

3. Verify that the SmartLink driver is able to find and configure an interface for the modem:

   ```
   # slmodemd --country=SWEDEN --alsa --nortpriority
   symbolic link `/dev/ttySL0' -> `/dev/pts/5' created.
   modem `modem:1' created. TTY is `/dev/pts/5'
   Use `/dev/ttySL0' as modem device, Ctrl+C for termination.
   ```

   You may see a complete list of recognized countries by:

   ```
   # slmodemd --countrylist
   ```

4. For convenience I want the modem driver configured as a service that is started by the Sys V init system. However, the `/etc/init.d/slmodemd` file installed by the package needs to be modified a bit in order to accomplish this and also to avoid problems on disconnects. You will find my modified version in Appendix B.

5. Now just add this new service to the Sys V system:

   ```
   # chkconfig --add slmodemd
   ```

6. Now edit your configuration options in `/etc/sysconfig/slmodemd`:

   ```
   # A list of all supported country names can be retrieved
   # by calling "slmodemd --countrylist" from the shell prompt.
   COUNTRY="SWEDEN"

   # No additional device needed for ALSA mode
   #DEVICE="slamr0"

   # If set to yes the Advanced Linux Sound Architecture
   # subsystem is used to make your modem working.
   USE_ALSA="yes"
   ```

7. Verify that the new service starts correctly:

   ```
   # service slmodemd start
   Starting SmartLink Modem driver:                          [  OK  ]
   # service slmodemd status
   slmodemd (pid 8356) is running...
   ```

   and look in `/var/log/messages`:

```
    ...slmodemd: symbolic link `/dev/ttySL0' -> `/dev/pts/1' created.
    ...slmodemd: modem `modem:1' created. TTY is `/dev/pts/1'
```

Just as the system log says you may now find the modem at `/dev/ttySL0`.

## 6.2 Remaining issue

However, there is a remaining problem.

I have got the modem working and connecting properly, but when disconnecting it bails out. The best way to observe this is to start `slmodemd` manually with the debug flag set:

```
# slmodemd -d=1 --alsa --country="SWEDEN" --nortpriority
```

Dialing and connecting works fine and the debug output looks ok. Now disconnect your session and watch the debug output go south. It starts flooding your console:

```
...
<212.190899> main: alsa xrun: try to recover...
<212.191065> main: alsa xrun: recovered.
<212.191069> main: dev read = 0
<212.191185> main: alsa xrun: try to recover...
<212.191351> main: alsa xrun: recovered.
<212.191355> main: dev read = 0
...
```

Stop the `slmodemd` process with Ctrl-C or `service slmodemd stop` from another console window.

This little issue together with the fact that `slmodemd` runs with real-time priority by default is devastating.

**Note**: You should never run `slmodemd` in the background without the `--nortpriority` option set. Failing to do so will most likely hang your entire system when disconnecting.

My temporary fix for this problem is to restart the `slmodemd` service right after disconnecting a session. I have configured my dial-up client, Kppp, to execute `/etc/init.d/slmodemd restart` upon disconnect. Unarguable this is a really dirty fix, but it seems to work quite well. It will do for the moment.

# 7 Touch-pad

The Synaptics touch-pad was properly configured during the installation, and works well. I use it in conjunction with a USB mouse and both works well in parallel. I have seen some reports on problems with the touch-pad 4-way multi-button, but it works without any hassle for me. Likewise does the mouse wheel.

Even the function key to toggle the touch-pad (`Fn-F7`) works without any special configuration. For reference you may have a look at the Core Pointer's InputDevice section in my `/etc/X11/xorg.conf` file in Appendix A.

## 8 Special buttons

The special buttons are some what confusing. Some of the special buttons do not need any special configuration to work. Others need a key code mapping, while some even lack a scan code. For the moment I have not fully dug into this subject, but the chain of scan codes and key codes translations starts in the core Linux kernel and ends in your X configuration. Some examples of confusion:

The Mail button gets a pre-configured key code of `155` by Linux, but with X loaded the key code is `236`. The buttons `WWW`, `Fn-F8`, `Fn-up` and `Fn-down` shows similar behavior.

The buttons `P1`, `P2`, `Fn-F1`, `Fn-F2`, `Fn-F3` do not have pre-configured key codes, while their respective scan codes are `e074`, `e073`, `e025`, `e026` and `e027`. However, in single user mode they all lack scan-codes. I do not know why.

However, do not despair. It is possible to get most of the special buttons working. Here is a short summary of my current status on this issue.

| ***Button*** | ***Work*** | **Config** | | | ***Comments*** |
|---|---|---|---|---|---|
| | | setkeycodes | Xmodmap | KDE | |
| `Mail` | yes | | x | x | E-mail button, ex: launch Thunderbird |
| `WWW` | yes | | x | x | WWW button, ex: launch Firefox |
| `P1` | yes | x | x | x | User button, ex: launch Xmms |
| `P2` | yes | x | x | x | User button, ex: launch VMware |
| `Fn-F1` | yes | x | x | x | User button, ex: |
| `Fn-F2` | yes | x | x | x | User button, ex: |
| `Fn-F3` | yes | x | x | x | User button, ex: |
| `Fn-F4` | ? | | | | No scan- nor key-codes. Function unknown. |
| `Fn-F5` | ? | | | | No scan- nor key-codes. Function unknown. |
| `Fn-F6` | yes | | | | Dim display, no configuration needed |
| `Fn-F7` | yes | | | | Toggle touch-pad, no configuration needed |
| `Fn-F8` | yes | | x | | Toggle mute |
| `Fn-up` | yes | | x | | Volume raise |
| `Fn-down` | yes | | x | | Volume lower |

| | | Config | | | |
|---|---|---|---|---|---|
| `Fn-left` | yes | | | | Brightness lighter, no configuration needed |
| `Fn-right` | yes | | | | Brightness darker, no configuration needed |
| `Bluetooth` | yes | | | | No configuration needed |
| `WLAN` | yes | | | | No configuration needed |

## 8.1 Configuration procedure

It is a tedious procedure to find out the proper scan-codes, Linux key-codes and X key-codes. Yes, on top of the scan-codes there are both Linux and X key-codes to keep track of. I probably have it all confused, but here is how I did it:

1. I started to find out what key-codes that X already knew of. Here `xev` is a valuable friend. The buttons that had key codes configured by default for me was:

| **Button** | **X key-code** |
|---|---|
| Mail | 236 |
| WWW | 178 |
| Fn-F8 | 160 |
| Fn-up | 176 |
| Fn-down | 174 |

These are the buttons that are the simplest to get working. Save these X key-codes for future use. First we need to get the other keys to show up under X as well. In order for them to do so they need properly configured Linux key-codes that they lack for the moment.

2. In order to configure the Linux key codes we need the scan-codes, and in order to get the scan-codes we need to shutdown X. Close your X applications and go to run level 3 (or restart into runlevel 3):

```
# init 3
```

3. Now ask what key-codes the buttons have by issuing:

```
# showkey -k
```

Since you do not have any key-codes configured yet you will get an error message telling you what scan-code that needs to be configured (`showkey -s` is supposed to show the scan-code, but failed miserably). I got the following scan-codes:

| **Button** | **scan-code** |
|---|---|
| P1 | e074 |
| P2 | e073 |
| Fn-F1 | e025 |
| Fn-F2 | e026 |
| Fn-F3 | e027 |

4. Start up X again by going back to run level 5:

```
# init 5
```

5. Figure out what Linux key-codes that are available by looking at the Linux scan-code – key-code mapping.

```
# getkeycodes
Plain scancodes xx (hex) versus keycodes (dec)
for 1-83 (0x01-0x53) scancode equals keycode

 0x50:   80  81  82  83  99   0  86  87
 0x58:   88 117   0   0  95 183 184 185
 0x60:    0   0   0   0   0   0   0   0
 0x68:    0   0   0   0   0   0   0   0
 0x70:   93   0   0  89   0   0  85  91
 0x78:   90  92   0  94   0 124 121   0

Escaped scancodes e0 xx (hex)

e0 00:    0 195 196 197 198   0   0   0
e0 08:    0   0   0   0   0   0   0   0
e0 10:  165   0   0   0   0   0   0   0
e0 18:    0 163   0   0  96  97   0   0
e0 20:  113 140 164   0 166   0   0   0
e0 28:    0   0 255   0   0   0 114   0
e0 30:  115   0 150   0   0  98 255  99
e0 38:  100   0   0   0   0   0   0   0
e0 40:    0   0   0   0   0 119 119 102
e0 48:  103 104   0 105 112 106 118 107
e0 50:  108 109 110 111   0   0   0   0
e0 58:    0   0   0 125 126 127 116 142
e0 60:    0   0   0 143   0 217 156 173
e0 68:  128 159 158 157 155 226   0 112
e0 70:    0   0   0   0   0   0   0   0
e0 78:    0   0   0   0   0   0   0   0
```

6. First we need to set a Linux key-code for the buttons that lack one. Add the following lines to `/etc/rc.d/rc.local` to set the key-codes after all services are started.

```
# Set Linux key-codes for special buttons:
```

```
#
# Buttons:          P1            P2
setkeycodes      e074 151      e073 152
#
# Buttons:         Fn-F1         Fn-F2         Fn-F3
setkeycodes      e025 131      e026 132      e027 133
```

7. In order to avoid a restart to load these setting, issues the very same commands.

```
# setkeycodes e074 151 e073 152
# setkeycodes e025 131 e026 132 e027 133
```

8. Repeat the step 1 and use `xev` to figure out what X key-codes these buttons got. I got the following:

| **Button** | **X key-code** |
|---|---|
| P1 | 201 |
| P2 | 146 |
| Fn-F1 | 135 |
| Fn-F2 | 140 |
| Fn-F3 | 248 |

9. Ok, now we have X key-codes for all special buttons and need to map them to proper key-symbols. In order to do that put the following in the file `/etc/X11/Xmodmap`:

```
! Acer Ferrari 3400Lmi special buttons
!
! Button    X key-code
! ------    ----------
! Mail      236
! WWW       178
! P1        201
! P2        146
! Fn-F1     135
! Fn-F2     140
! Fn-F3     248
! Fn-F8     160
! Fn-up     176
! Fn-down   174
!
keycode 236 = XF86Mail
keycode 178 = XF86WWW
keycode 201 = XF86Launch1
keycode 146 = XF86Launch2
keycode 135 = XF86Launch3
keycode 140 = XF86Launch4
keycode 248 = XF86Launch5
keycode 160 = XF86AudioMute
keycode 176 = XF86AudioRaiseVolume
keycode 174 = XF86AudioLowerVolume
```

10. The setting above are loaded the next time X is started, but to load them without a restart of X do:

```
# xmodmap -verbose /etc/X11/Xmodmap
...
```

11. Finally it is time to configure the button actions. The audio buttons (mute & volume) do not need any further configuration and should work by now. Actions for the other buttons are easily configured in the KDE Control Center -> Regional & Accessibility -> Keyboard Shortcuts under the tab Command Shortcuts.

# 9 PC-card

I have only used the PC-card slot for a Compact Flash memory adapter and it just works. True plug-and-play.

# 10 Bluetooth

No special actions were needed for me to get Bluetooth up and running. It was truly amazing how easy it was. However, I might have been lucky so here are some things to check.

## 10.1 Verify the Bluetooth installation

Make sure that you have the bluez-utils package installed:

```
# rpm -q bluez-utils
bluez-utils-2.15-7
```

Also make sure that it is configured it to start at boot time:

```
# chkconfig --list bluetooth
bluetooth      0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

If not, you need to add it to the runlevel system:

```
# chkconfig --add bluetooth
```

Now watch your system log while you push the bluetooth button on the front of your laptop to activate your bluetooth circuities:

```
# tail -f /var/log/messages
...
... kernel: usb 4-2: new full speed USB device using uhci_hcd and
address 3
... kernel: Bluetooth: HCI USB driver ver 2.8
... hcid[1934]: HCI dev 0 registered
... kernel: usbcore: registered new driver hci_usb
... redneck hcid[1934]: HCI dev 0 up
... redneck hcid[1934]: Starting security manager 0
...
```

The blue led indicator should start blinking as well. Now verify that your bluetooth device is up and running:

```
# hciconfig -a
hci0:   Type: USB
        BD Address: 00:0E:9B:87:3B:90 ACL MTU: 192:8 SCO MTU: 64:8
        UP RUNNING PSCAN ISCAN
        RX bytes:107 acl:0 sco:0 events:14 errors:0
        TX bytes:300 acl:0 sco:0 commands:13 errors:0
        Features: 0xff 0xff 0x0f 0x00 0x00 0x00 0x00 0x00
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: SLAVE ACCEPT
        Name: 'redneck.superwise.net-0'
        Class: 0x120104
        Service Classes: Networking, Object Transfer
        Device Class: Computer, Desktop workstation
        HCI Ver: 1.1 (0x1) HCI Rev: 0x20d LMP Ver: 1.1 (0x1) LMP Subver:
0x20d
        Manufacturer: Cambridge Silicon Radio (10)
```

If you have come this far without any problems you are all set and ready to go.

## 10.2 Example – Use your phone as a modem

To help get you started I have summarized the basic steps to hook up your bluetooth capable phone as a modem. I am using a Sony Ericsson P900 myself, but the steps are general so it should work for most bluetooth phones.

The steps below are exactly the same whether you intend to connect to a remote modem or connect over GPRS. When connecting over GPRS the phone also needs to be connected and recognized as a modem. It is the actual dialing later on in the process that differs.

First you need to configure your phone so it is ready for use by your laptop. These steps may vary in detail depending on vendor and model, so the user manual for your phone may be handy.

Start by activating bluetooth on your phone and scan for other devices. You should now be able to see your computer.

Add your computer as a known bluetooth device to your phone. The phone will ask you for the pin code to connect to your computer. By default it is `BlueZ`, but you may modify it in `/etc/bluetooth/pin`. Use the same pin code when the computer prompts you whether to allow the incoming connection.

For convenience you should now configure your phone to allow this device (your computer) to connect without prompting for a pin code.

Now we are ready to configure the laptop. Start by scanning for bluetooth devices in your surrounding:

```
# hcitool scan
Scanning ...
        00:0A:D9:E9:D8:4F       S-Gs P900
```

The first field is the bluetooth address of your phone. The second field its given name. So now you should be able to ping it:

```
# l2ping 00:0A:D9:E9:D8:4F
Ping: 00:0A:D9:E9:D8:4F from 00:0E:9B:87:3B:90 (data size 20) ...
0 bytes from 00:0A:D9:E9:D8:4F id 0 time 60.87ms
0 bytes from 00:0A:D9:E9:D8:4F id 1 time 27.77ms
0 bytes from 00:0A:D9:E9:D8:4F id 2 time 36.54ms
3 sent, 3 received, 0% loss
```

Note that you should use your own phones address instead. (My phone will probably be out of range :-)

Now it is time to find out what services your phone provides:

```
# sdptool browse 00:0A:D9:E9:D8:4F
Browsing 00:0A:D9:E9:D8:4F ...
Service Name: Voice gateway
Service Description: Voice gateway
Service Provider: Sony Ericsson
Service RecHandle: 0x10000
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 8
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Headset" (0x1108)
    Version: 0x0100
```

```
Service Name: Bluetooth Serial Port
Service Description: Bluetooth Serial Port
Service Provider: Symbian Ltd.
Service RecHandle: 0x10001
Service Class ID List:
  "Serial Port" (0x1101)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100

Service Name: Dial-up Networking
Service Description: Dial-up Networking
Service Provider: Sony Ericsson
Service RecHandle: 0x10002
Service Class ID List:
  "Dialup Networking" (0x1103)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 2
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Dialup Networking" (0x1103)
    Version: 0x0100

Service Name: OBEX Object Push
Service RecHandle: 0x10003
Service Class ID List:
  "OBEX Object Push" (0x1105)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
  "OBEX" (0x0008)
Profile Descriptor List:
  "OBEX Object Push" (0x1105)
    Version: 0x0100

Service Name: OBEX File Transfer
Service RecHandle: 0x10004
Service Class ID List:
  "OBEX File Transfer" (0x1106)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 4
  "OBEX" (0x0008)
```

To use the phone as a modem the service of interest is the Dial-up Networking, so note its channel number.

The next step will be to create a virtual serial device and connect it to your phone:

```
# rfcomm connect 1 00:0A:D9:E9:D8:4F 2
Connected /dev/rfcomm1 to 00:0A:D9:E9:D8:4F on channel 2
Press CTRL-C for hangup
```

A short explanation of the command above:

```
rfcomm connect 1 00:0A:D9:E9:D8:4F 2
               |         |         |
 /dev/rfcomm1 _|         |         |_ Channel for the Dial-up
                    Your phones       Networking service
                 bluetooth address
```

That is about it. Now your phone is analogue to an external modem connected to your virtual serial device, `/dev/rfcomm1`. Configure ppp to make use of it and you are done.

## 10.3 Static configuration

Ok, now what? Do I need to repeat all the steps above each time I want to use my phone as a modem? No, for convenience you may configure your system for all this to take place automatically. However, the method you should use depends a bit on your phone.

The preferred method is to edit the `rfcomm.conf` file so a `/dev/rfcommX` port automatically binds to the DUN service on your phone when the bluetooth service starts. This means that the virtual serial device will be created and properly configured, but not connected. The actual connection will happen automagically when the virtual serial device is accessed.

Edit the file `/etc/bluetooth/rfcomm.conf` to contain a section similar to this:

```
rfcomm1 {
        # Automatically bind the device at startup
        bind yes;

        # Bluetooth address of the phone
        device 00:0A:D9:E9:D8:4F;

        # RFCOMM channel for the Dial Up Networking service
        channel 2;

        # Description of the connection
        comment "Modem on my phone";
}
```

### 10.4 Dynamic configuration

If the method above works you are all set and done. I started out that way and everything worked great for a while. Then all of a sudden I was unable to connect. It turned out that the DUN channel on my phone had changed!?! Instead of 2 as in the example above it showed up as channel 4, and later on as channel 3...

The statical configuration done by editing the `rfcomm.conf` file cannot handle this confusion. Instead I needed to dynamically decide which channel my phone used for the DUN service today and bind to it.

This is done in a simple shell script, that is called just before I intend to connect. Personally, I use Kppp and find it great for both modem-to-modem dial-ups and GPRS connections. So, I have configured KDE to call my script just before Kppp is opened.

First I was a bit suspicious about this method to work all the time, but I have not had any trouble this far. You may find a printout of the script in the Appendix C.

Now all you need to do when you want to use your phone as a modem are the most basic steps:

- Turn on bluetooth on your phone
- Turn on the bluetooth chip on your laptop
- Dial!

### 10.5 Sending files – OBEX Push

To send files to your OBEX (Object Exchange) capable phone you need the packages `openobex` and `openobex-apps`.

```
# opex_push 3 00:0A:D9:E9:D8:4F test.jpg
             |         |
             |         |_ Bluetooth address to send to
             |
             |_Channel for the OBEX Object Push service
```

# 11 USB

Works as expected. I have noticed no problems what-so-ever with the USB ports.

# 12 IEEE 1394 Firewire

My experience with Linux (Redhat & Fedora) and Firewire is mixed. Several years ago I was running Redhat 7.3 on an ASUS laptop and there were no hassle what so ever with firewire. This was in the days of USB 1.0 and most Windows users lacked support for Firewire. So many friends gazed with envy when I ran a fat pipe to my external 2.5” hard drive from the laptop.

However, that was Redhat 7.3 and since then I have been running Redhat 9, Fedora 1, 2 and 3 on the very same laptop with the very same external drives and have not enjoyed the same Firewire support ever since. On the other hand, I have not had the proper time to dig into this full heartedly. When I noticed the same problems with FC 4 on *My Precious* I was not surprised, just disappointed.

However, now I had a perfectly valid reason to spend some time on this matter and there is, as always, a solution. First a short description of the potential problems.

## 12.1 Potential problems

There are no problems regarding loading modules or mounting an external IEEE 1394 drive, and if I am patient I managed to browse the content as well. The problems starts when I try to transfer larger amounts of data. The process stalls and chokes up the system log with messages like:

```
kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Write (10): 2a 00 02 e1 bc 58 00 00 10 00
kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Write (10): 2a 00 02 e1 bc 58 00 00 10 00
kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Test Unit Ready: 00 00 00 00 00 00
kernel: ieee1394: sbp2: reset requested
kernel: ieee1394: sbp2: Generating sbp2 fetch agent reset
redneck kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Write (10): 2a 00 01 06 d0 df 00 00 03 00
kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Write (10): 2a 00 01 06 d0 df 00 00 03 00
kernel: ieee1394: sbp2: aborting sbp2 command
kernel: scsi1 : destination target 0, lun 0
kernel:         command: Write (10): 2a 00 02 e1 bd b0 00 00 20 00
```

Seems to me like a hole bunch of timeouts with corresponding bus resets. These suspicions got even stronger after timing a read data transfer:

```
# time cp -rp /media/ieee1394disk/430MB_folder ~
real    20m29.516s
user    0m0.052s
sys     0m6.476s
```

Copying 430 MB takes 20 minutes 29 seconds (comparable to USB 1.0 performance). However, the “actual” time is less than 7 seconds. 20 minutes and 22 seconds are spent waiting. Waiting for what? I do not know, but obviously some bits and pieces fail during the transfer. Furthermore, I do not feel comfortable with the data integrity when I see these kind of results.

After some digging in the kernel documentation and a quick look in the `sbp2.c` source file it turned out that this problem probably is related to a “buggy IEEE 1394 chip”. (Uhhh, I was deeply hurt. Someone has the guts to claim that *My Precious* is buggy? :-) The proposed solution was to load the `sbp2` module with the argument `serialize_io=1`. It turned out really well, so here are some tips regarding the IEEE 1394 configuration.

## 12.2 Configuring IEEE 1394 Firewire

If you experience the same problems as mentioned above, and you are running kernel version 2.6.13 or earlier, put the following line in your `/etc/modprobe.config`:

```
options sbp2 serialize_io=1 max_speed=2
```

The `serialize_io=1` option tells the scsi drivers to only send one scsi command at a time. Unfortunately, this setting has a small impact on performance, but it is the fix that makes things work.

In kernel version 2.6.14 the default value for `serialize_io` was changed from `0` to `1`. Thus, if you are running version 2.6.14 or later you should not need do do anything, unless you want to optimize performance (see comments below) or fiddle with the other option above.

The `max_speed` option might be useful in rare occasions if you want to limit the maximum transfer rate to support “even more buggy” external hardware. Valid values for the max_speed option are:

| | |
|---|---|
| 0 | 100 mb |
| 1 | 200 mb |
| 2 | 400 mb (default) |
| 3 | 800 mb |

When timing the very same read transfer as above I now get the following result:

```
# time cp -rp /media/ieee1394disk/430MB_folder ~
real    0m24.871s
user    0m0.076s
sys     0m6.400s
```

That is what I call improvement! Going from over 20 minutes down to roughly 25 seconds.

## 12.3 Comments on IEEE 1394 options

After some further exercises with other external hard drives it turned out that the problems described in the previous section seem to be related to the IEEE 1394 chip in the external drives (not in *My Precious* :-). With some hardware it is quite possible to use the faster `serialize_io=0` option. The performance benefit is in the range 20-25%, so consider your options. If you only use IEEE 1394 for your own hardware and it works well with the faster setting, go for it. Otherwise, compatibility with other hardware might be more valuable. Personally, I think it was a wise decision to change the default setting in the `sbp2` module. After all those “buggy IEEE 1394 chips” seem to be quite common, and prior to start optimizing performance you just want things to work.

# 13 Infrared

IrDA support is provided by means of the package `irda-utils`, so first make sure that this package is installed on your system.

My first attempt started with changing the `DEVICE` in `/etc/sysconfig/irda` to `/dev/ttyS1` and fire up the IrDA service (`/etc/init.d/irda start`). Voilà! Watching the log messages verified that all modules were loaded and I had got a new device, `irda0`, to play with. The device showed up with `ifconfig` as well. It was just too easy! And yes, although all looked perfect it did not work. Trying the `irdadump` reviled just a big silence.

## 13.1 Configuring IrDA

To make a long story short, the IR-chip in the Ferrari supports FIR (as well as SIR) and FIR is the default, while IrDA by default uses SIR. FIR is what you want to go for since it is faster than SIR. Below I'll walk you through the steps that got it working for me.

1. Start with grabbing a pen and a piece of paper and restart your *Precious*. Yes, this is one of those few occasions when you need to restart you Linux system. Press `F2` during boot-up to enter the BIOS and note the settings for your IR-port. You do not need to change anything, but you need to know your exact setting. I will use my own setting through out this example:

   ```
   Base I/O address:    [2F8]
   Interrupt:           [IRQ 3]
   DMA channel:         [DMA 1]
   ```

   Once you have noticed your corresponding setting just exit the BIOS without saving and start your system.

2. Make sure that no other services use IRQ 3. Most likely your setting is also IRQ 3, so start looking in the `/etc/pcmcia/config.opts` file. Here you need to uncomment the line

   ```
   exclude irq 3
   ```

   to prevent the pcmcia service from intervening.

3. Now we want that a module capable of handling FIR on the Ferrari chip is

loaded when the IdDA service is started. The module of choice is `nsc-ircc`, so add the following two lines in `/etc/modprobe.conf`:

```
alias irda0 nsc-ircc
options nsc-ircc dongle_id=0x09 io=0x2f8 irq=3 dma=1
```

Pay attention to use the settings from your own BIOS for the last three parameters.

4. Now we should tell the IrDa service to attach directly to the device for our FIR capable module, so make sure to change the `DEVICE` setting in `/etc/sysconfig/irda` to:

```
DEVICE=irda0
```

5. Finally we do not want the generic Linux serial driver to interfere. One way of doing that is to add the following line in `/etc/init.d/irda`:

```
setserial /dev/ttyS1 uart none
```

The line should be place just before daemon

```
/usr/sbin/irattach ${DEVICE} ${ARGS}
```

That is about it, You are done with the configuration.

## 13.2 Testing IrDA

Now start the IrDA service and watch the system log. Hopefully, you should see something similar the the following:

```
# service irda start
Starting IrDA:                                                  [  OK  ]
# dmesg | tail
...
NET: Registered protocol family 23
IrCOMM protocol (Dag Brattli)
CSLIP: code copyright 1989 Regents of the University of California
PPP generic driver version 2.4.2
nsc-ircc, Found chip at base=0x02e
nsc-ircc, driver loaded (Dag Brattli)
IrDA: Registered device irda0
nsc-ircc, Using dongle: IBM31T1100 or Temic TFDS6000/TFDS6500
```

This verifies that you got the proper modules in place. The last step is to verify that we are able both of transmitting and receiving traffic. So activate IR on the remote device, e.g. your phone, and place the two IR-ports eye-to-eye. Then do a dump of the traffic:

```
# irdadump -i irda0
...xid:cmd 4d975258 > ffffffff S=6 s=2 (14)
...xid:cmd 4d975258 > ffffffff S=6 s=3 (14)
...xid:cmd 4d975258 > ffffffff S=6 s=4 (14)
```

```
...xid:cmd 4d975258 > ffffffff S=6 s=5 (14)
...xid:cmd 4d975258 > ffffffff S=6 s=* redneck hint=4400 [ Computer LAN
Access ] (23)
...xid:cmd 4d975258 > ffffffff S=6 s=0 (14)
...xid:rsp 4d975258 < 51a3abf6 S=6 s=0 P900 hint=9325 [ PnP PDA/Palmtop
Modem Telephony IrCOMM IrOBEX ] (21)
```

You're all set! The first I did after this was to use `irobex_palm3 <SIS-file>` to upload and install GnuBox and some other programs on my phone. To do this you need to have the `openobex` and `openobex-apps` packages installed and your phone must supports the OBEX protocol... Pretty neat!

# 14 5-in-1 Card reader

The 5-in-1 card reader utilizes the USB interface and is operational right after installation. Likewise to the USB ports there is no hassle at all.

# 15 References

Below are the links I have found most useful when setting up *my Precious*. Thank you guys:

Debian GNU/Linux on the Acer Ferrari 3400LMi
    René Tschirley
    http://www.tschirley.com/linux/acer-ferrari-3400.html

Fedora Core 1 on the Acer Ferrari 3000LMi
    Evan
    http://ferrari.kicks-ass.org/

SuSE 9.1 Pro on the Acer Ferrari 3000LMi
    Dirk Praet
    http://www.designisdead.com/ferrari/

P800 and Linux
    Celso Martinho
    http://celso.arrifana.org/wiki/p800_and_linux

Wireless LAN resources for Linux
    Jean Tourrilhes
    http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Wireless.html

# Appendix A – `/etc/X11/xorg.conf`

```
# **********************************************************************
# Refer to the xorg.conf man page for details about the format of
# this file.
# **********************************************************************
# **********************************************************************
# DRI Section
# **********************************************************************
Section "DRI"
# Access to OpenGL ICD is allowed for all users:
   Mode 0666
EndSection

# **********************************************************************
# Module section -- This  section is used to specify
# which dynamically loadable modules to load.
# **********************************************************************
Section "Module"
   Load "dbe"    # Double buffer extension
   SubSection "extmod"
        Option   "omit xfree86-dga"   # don't initialise the DGA
extension
   EndSubSection
   Load "type1"
   Load "freetype"
   Load "glx"   # libglx.a
   Load "dri"   # libdri.a
   Load "fbdevhw"
   Load "record"
   Load "synaptics"
EndSection

# **********************************************************************
# Files section -- This allows default font and rgb paths to be set
# **********************************************************************
Section "Files"
   RgbPath       "/usr/X11R6/lib/X11/rgb"
   FontPath      "unix/:7100"
#  ModulePath    "/usr/X11R6/lib/modules"
EndSection

# **********************************************************************
# Server flags section.
# **********************************************************************
# **********************************************************************
# Input devices
# **********************************************************************
# **********************************************************************
# Core keyboard's InputDevice section
# **********************************************************************
Section "InputDevice"
   Identifier    "Keyboard1"
   Driver        "kbd"
   Option        "XkbModel"          "pc105"
   Option        "XkbLayout"         "se"
EndSection
```

```
# **********************************************************************
# Core Pointer's InputDevice section
# **********************************************************************
Section "InputDevice"
   Identifier    "Mouse1"
   Driver        "mouse"
   Option        "Protocol"          "IMPS/2"
   Option        "Device"            "/dev/input/mice"
   Option        "ZAxisMapping"      "4 5"
EndSection

Section "InputDevice"
   Identifier    "Synaptics"
   Driver        "synaptics"
   Option        "Protocol"          "auto-dev"
   Option        "Device"            "/dev/input/mice"
   Option        "Emulate3Buttons"   "yes"
EndSection

# **********************************************************************
# Monitor section.
# **********************************************************************
Section "Monitor"
   Identifier    "Monitor0"
   VendorName    "Monitor Vendor"
   ModelName     "LCD Panel 1400x1050"
   HorizSync     31.5 - 90.0
   VertRefresh   59.0 - 75.0
   Option        "DPMS"
EndSection


# **********************************************************************
# Graphics device section
# **********************************************************************

# === ATI device section ===
Section "Device"
   Identifier    "ATI Graphics Adapter"
   Driver        "fglrx"
# ### generic DRI settings ###
# === disable PnP Monitor  ===
#  Option        "NoDDC"
# === disable/enable XAA/DRI ===
   Option        "no_accel"          "no"
   Option        "no_dri"            "no"
# === misc DRI settings ===
   Option        "mtrr"              "off"   # disable DRI mtrr mapper,
                                     # driver has its own code for mtrr
# ### FireGL DDX driver module specific settings ###
# === Screen Management ===
   Option        "DesktopSetup"      "0x00000100"
   Option        "MonitorLayout"      "AUTO, AUTO"
   Option        "IgnoreEDID"         "off"
   Option        "HSync2"             "unspecified"
   Option        "VRefresh2"          "unspecified"
   Option        "ScreenOverlap"      "0"
# === TV-out Management ===
   Option        "NoTV"               "no"
   Option        "TVStandard"         "PAL-B"
#  Option        "TVStandard"         "VIDEO" # {VIDEO|SCART|YUV}
#  Option        "TVFormat"           "PAL-B"
   Option        "TVHSizeAdj"         "0"
   Option        "TVVSizeAdj"         "0"
   Option        "TVHPosAdj"          "0"
   Option        "TVVPosAdj"          "0"
   Option        "TVHStartAdj"        "0"
   Option        "TVColorAdj"         "0"
   Option        "GammaCorrectionI"   "0x06419064"
   Option        "GammaCorrectionII"  "0x06419064"
# === OpenGL specific profiles/settings ===
   Option        "Capabilities"       "0x00000000"
# === Video Overlay for the Xv extension ===
   Option        "VideoOverlay"       "on"
# === OpenGL Overlay ===
# Note: When OpenGL Overlay is enabled, Video Overlay
#       will be disabled automatically
   Option        "OpenGLOverlay"      "off"
# === Center Mode (Laptops only) ===
   Option        "CenterMode"         "off"
# === Pseudo Color Visuals (8-bit visuals) ===
   Option        "PseudoColorVisuals" "off"
# === QBS Management ===
   Option        "Stereo"             "off"
   Option        "StereoSyncEnable"   "1"
# === FSAA Management ===
   Option        "FSAAEnable"         "no"
   Option        "FSAAScale"          "1"
   Option        "FSAADisableGamma"   "no"
   Option        "FSAACustomizeMSPos" "no"
   Option        "FSAAMSPosX0"        "0.000000"
   Option        "FSAAMSPosY0"        "0.000000"
   Option        "FSAAMSPosX1"        "0.000000"
   Option        "FSAAMSPosY1"        "0.000000"
   Option        "FSAAMSPosX2"        "0.000000"
   Option        "FSAAMSPosY2"        "0.000000"
   Option        "FSAAMSPosX3"        "0.000000"
   Option        "FSAAMSPosY3"        "0.000000"
   Option        "FSAAMSPosX4"        "0.000000"
   Option        "FSAAMSPosY4"        "0.000000"
   Option        "FSAAMSPosX5"        "0.000000"
   Option        "FSAAMSPosY5"        "0.000000"
# === Misc Options ===
   Option        "UseFastTLS"         "0"
```

```
    Option        "BlockSignalsOnLock" "on"
    Option        "UseInternalAGPGART" "yes"
    Option        "ForceGenericCPU"    "no"
    BusID         "PCI:1:0:0"          # vendor=1002, device=4e50
    Screen        0
EndSection

# **********************************************************************
# Screen sections
# **********************************************************************
# Any number of screen sections may be present. Each describes
# the configuration of a single screen. A single specific screen section
# may be specified from the X server command line with the "-screen"
# option.

Section "Screen"
    Identifier    "Screen0"
    Device        "ATI Graphics Adapter"
    Monitor       "Monitor0"
    DefaultDepth  24

    SubSection "Display"
         Depth    16
         Modes    "800x600" "640x480"
         Viewport 0 0
    EndSubSection

    SubSection "Display"
         Depth    24
         Modes    "1400x1050" "1280x1024" "1152x864" "1024x768" "800x600"
         Viewport 0 0 # initial origin if mode is smaller than desktop
    EndSubSection
EndSection

# **********************************************************************
# ServerLayout sections.
# **********************************************************************
# Any number of ServerLayout sections may be present. Each describes
# the way multiple screens are organised. A specific ServerLayout
# section may be specified from the X server command line with the
# "-layout" option. In the absence of this, the first section is used.
# When now ServerLayout section is present, the first Screen section
# is used alone.

Section "ServerLayout"
    Identifier    "Server Layout"
    Screen        0 "Screen0" 0 0
    InputDevice   "Mouse1"             "CorePointer"
    InputDevice   "Synaptics"          "AlwaysCore"
    InputDevice   "Keyboard1"          "CoreKeyboard"
EndSection
```

```
### EOF ###
```

## Appendix B - /etc/init.d/slmodemd

```
#!/bin/sh
#
# chkconfig:   2345 66 33
# description: SmartLink Modem Driver \
#              User space daemon of winmodem driver
# config:      /etc/sysconfig/slmodemd

PROG=slmodemd
PROG_PATH=/usr/sbin/${PROG}
RETVAL=0

# Default configuration
COUNTRY="SWEDEN"
USE_ALSA="yes"
#DEVICE=slamr0

# Source function library.
if [ -f /etc/init.d/functions ] ; then
    . /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
    . /etc/rc.d/init.d/functions
else
    exit 0
fi

# Source configuration
CONFIG=/etc/sysconfig/${PROG}
if [ -f $CONFIG ]; then
    . $CONFIG
fi

# Do not try to start on a kernel which does not support it
if [ $USE_ALSA != "yes" ]; then
    grep -q 'slamr\.o' /lib/modules/`uname -r`/modules.dep || exit 0
fi

start() {
    echo -n "Starting SmartLink Modem driver: "
    # IMPORTANT! use the --nortpriority option,
    # otherwise your system will hang on disconnect.
    OPTS=" --daemon --country=$COUNTRY --nortpriority"
    if [ $USE_ALSA = "yes" ]; then
        OPTS="$OPTS --alsa"
    else
        OPTS="$OPTS $DEVICE"
        modprobe slamr
        modprobe slusb
```

```
     sleep 3
    fi
    daemon $(${PROG_PATH} ${OPTS} 2>&1 | logger -t ${PROG})
    PID=`pidof ${PROG}`
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && touch /var/lock/subsys/${PROG}
    return $RETVAL
}

stop() {
    echo -n "Shutting down SmartLink Modem driver: "
    killproc ${PROG}
    RETVAL=$?
    echo
    if [ $USE_ALSA != "yes" ]; then
        modprobe -r slamr slusb
    fi
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/${PROG}
    return $RETVAL
}

# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status ${PROG}
        ;;
    restart|reload)
        stop
        start
        ;;
    condrestart)
        if [ -f /var/lock/subsys/${PROG} ]; then
            stop
            start
        fi
        ;;
    install)
        chkconfig --add ${PROG}
        ;;
    *)
        echo "Usage: ${PROG} {start|stop|status|restart|condrestart}"
        exit 1
esac

exit $?
```

# Appendix C – dun-bind.sh

```
#!/bin/sh
#
# A verbose sample script for finding the Dial Up Networking
# channel on a Bluetooth device and binding a /dev/rfcomm port to it.
#
# By Sven-Göran Bergh, 2005-11-03

# Use this /dev/rfcomm port
RFPORT=1

# Name of Bluetooth device to bind to:
BTNAME="S-Gs P900"

success() {
  echo -e $"\\033[60G[  \\033[1;32mOK\\033[0;39m  ]"
  return 0
}

failure() {
  echo -e $"\\033[60G[\\033[1;31mFAILED\\033[0;39m]"
  exit 1
}

# Check if the rfcomm port is free
echo -n $"Checking that /dev/rfcomm${RFPORT} is free..."
rfcomm show ${RFPORT} &> /dev/null && \
    failure || success

# Check that local Bluetooth device is active
echo -n $"Checking for local Bluetooth device..."
hciconfig | grep 'UP RUNNING' &> /dev/null && \
    success || failure

# Check for the remote Bluetooth device
echo -n $"Searching for remote Bluetooth device ${BTNAME}..."
BTADDR=`hcitool scan | grep "${BTNAME}" | awk -- '{print $1}'`
[ "${BTADDR}" ] && success || failure

# Find the Dial Up Networking channel
echo -n $"Searching for Dial Up Networking service..."
DUN=`sdptool search --bdaddr ${BTADDR} DUN \
    | awk -- '/Channel/ {print $2}'`
[ "${DUN}" ] && success || failure

# Bind the rfcomm port to the DUN channel
echo -n $"Binding /dev/rfcomm${RFPORT} to DUN channel ${DUN}..."
rfcomm bind ${RFPORT} ${BTADDR} ${DUN} && \
    success || failure
```