Framebuffer HOWTO

Table of Contents

Table of Contents

Framebuffer HOWTO

Alex Buell, `alex.buell@tahallah.clara.co.uk`

v1.2, 27 Feb 2000

This document describes how to use the framebuffer devices in Linux with a variety of platforms. This also includes how to set up multi−headed displays.

---

8. Using framebuffer devices on Macintosh m68k platforms

9. Using framebuffer devices on PowerPC platforms

10. Using framebuffer devices on Alpha platforms

11. Using framebuffer devices on SPARC platforms

12. Using framebuffer devices on MIPS platforms

13. Using framebuffer devices on ARM platforms

14. Using multi−headed framebuffers

15. Using/Changing fonts

16. Changing console modes

17. Setting up the X11 FBdev driver

18. How do I convert XFree86 mode−lines into framebuffer device timings?

19. Changing the Linux logo

20. Looking for further information?

---

1. History

Revision history

19990607 – Release of 1.0

19990722 – Release of 1.1

20000222 – Release of 1.2

---

2. Contributors

Thanks go to these people listed below who helped improve the Framebuffer HOWTO.

- Jeff Noxon `jeff@planetfall.com`
- Francis Devereux `f.devereux@cs.ucl.ac.uk`
- Andreas Ehliar `ehliar@futurniture.se`
- Martin McCarthy `marty@ehabitat.demon.co.uk`
- Simon Kenyon `simon@koala.ie`
- David Ford `david@kalifornia.com`
- Chris Black `cblack@cmpteam4.unil.ch`
- N Becker `nbecker@fred.net`
- Bob Tracy `rct@gherkin.sa.wlk.com`
- Marius Hjelle `marius.hjelle@roman.uib.no`
- James Cassidy `jcassidy@misc.dyn.ml.org`
- Andreas U. Trottmann `andreas.trottmann@werft22.com`
- Lech Szychowski `lech7@lech.pse.pl`
- Aaron Tiensivu `tiensivu@pilot.msu.edu`
- Jan−Frode Myklebust for his info on permedia cards `janfrode@ii.uib.no`
- Many others too numerous to add, but thanks!

Thanks go to Rick Niles `frederick.a.niles@gsfc.nasa.gov` who has very kindly handed over his Multi−Head Mini−HOWTO for inclusion in this HOWTO.

Thanks to these people listed below who built libc5/glibc2 versions of the XF86_FBdev X11 framebuffer driver for X11 on Intel platforms:

- Brion Vibber `brion@pobox.com`

- Gerd Knorr `kraxel@cs.tu-berlin.de`

and of course the authors of the framebuffer devices:

- Martin Schaller − original author of the framebuffer concept
- Roman Hodek `Roman.Hodek@informatik.uni-erlangen.de`
- Andreas Schwab `schwab@issan.informatik.uni-dortmund.de`
- Guenther Kelleter
- Geert Uytterhoeven `Geert.Uytterhoeven@cs.kuleuven.ac.be`
- Roman Zippel `roman@sodom.obdg.de`
- Pavel Machek `pavel@atrey.karlin.mff.cuni.cz`
- Gerd Knorr `kraxel@cs.tu-berlin.de`
- Miguel de Icaza `miguel@nuclecu.unam.mx`
- David Carter `carter@compsci.bristol.ac.uk`
- William Rucklidge `wjr@cs.cornell.edu`
- Jes Sorensen `jds@kom.auc.dk`
- Sigurdur Asgeirsson
- Jeffrey Kuskin `jsk@mojave.stanford.edu`
- Michal Rehacek `michal.rehacek@st.mff.cuni.edu`
- Peter Zaitcev `zaitcev@lab.ipmce.su`
- David S. Miller `davem@dm.cobaltmicro.com`
- Dave Redman `djhr@tadpole.co.uk`
- Jay Estabrook
- Martin Mares `mj@ucw.cz`
- Dan Jacobowitz `dan@debian.org`
- Emmanuel Marty `core@ggi-project.org`
- Eddie C. Dost `ecd@skynet.be`
- Jakub Jelinek `jj@ultra.linux.cz`
- Phil Blundell `philb@gnu.org`
- Anyone else, stand up and be counted. :o)

---

3. What is a framebuffer device?

A framebuffer device is an abstraction for the graphic hardware. It represents the frame buffer of some video hardware, and allows application software to access the graphic hardware through a well−defined interface, so that the software doesn't need to know anything about the low−level interface stuff [Taken from Geert Uytterhoeven's framebuffer.txt in the linux kernel sources]

---

4. What advantages does framebuffer devices have?

Penguin logo. :o) Seriously, the major advantage of the framebuffer drives is that it presents a generic interface across all platforms. It was the case until late in the 2.1.x kernel development process that the Intel platform had console drivers completely different from the other console drivers for other platforms. With the introduction of 2.1.109 all this has changed for the better, and introduced more uniform handling of the console under the Intel platforms and also introduced true bitmapped graphical consoles bearing the Penguin logo on Intel for the first time, and allowed code to be shared across different platforms. Note that 2.0.x kernels do not support framebuffer devices, but it is possible someday someone will backport the code from the 2.1.x kernels to 2.0.x kernels. There is an exception to that rule in that the v0.9.x kernel port for m68k platforms does have the framebuffer device support included.

With the release of the 2.2.x kernel, framebuffer device support is very solid and stable. You should use the framebuffer device if your graphic card supports it, if you are using 2.2.x kernels. Older 2.0.x kernels does not support framebuffer devices, at least on the Intel platform.

- 0.9.x (m68k) – introduced m68k framebuffer devices. Note that m68k 0.9.x is functionally equivalent to Intel 1.0.9 (plus 1.2.x enhancements)
- 2.1.107 – introduced Intel framebuffer/new console devices and added generic support, without scrollback buffer support.
- 2.1.113 – scrollback buffer support added to vgacon.
- 2.1.116 – scrollback buffer support added to vesafb.
- 2.2.x – includes matroxfb(Matrox) and atyfb(ATI).

There are some cool features of the framebuffer devices, in that you can give generic options to the kernel at bootup–time, including options specific to a particular framebuffer device. These are:

- `video=xxx:off` – disable probing for a particular framebuffer device
- `video=map:octal-number` – maps the virtual consoles (VCs) to framebuffer (FB) devices
    - `video=map:01` will map VC0 to FB0, VC1 to FB1, VC2 to FB0, VC3 to FB1..
    - `video=map:0132` will map VC0 to FB0, VC1 to FB1, VC2 to FB3, VC4 to FB2, VC5 to FB0..

Normally framebuffer devices are probed for in the order specified in the kernel, but by specifying the `video=xxx` option, you can add the specific framebuffer device you want probed before the others specified in the kernel.

---

5. Using framebuffer devices on Intel platforms

5.1 What is vesafb?

Vesafb is a framebuffer driver for Intel architecture that works with VESA 2.0 compliant graphic cards. It is closely related to the framebuffer device drivers in the kernel.

vesafb is a display driver that enables the use of graphical modes on your Intel platform for bitmapped text consoles. It can also display a logo, which is probably the main reason why you'd want to use vesafb :o)

Unfortunately, you can not use vesafb successfully with VESA 1.2 cards. This is because these 1.2 cards do not use *linear* frame buffering. Linear frame buffering simply means that the system's CPU is able to access every bit of the display. Historically, older graphic adapters could allow the CPU to access only 64K at a time, hence the limitations of the dreadful CGA/EGA graphic modes! It may be that someone will write a vesafb12 device driver for these cards, but this will use up precious kernel memory and involve a nasty hack.

There is however a potential workaround to add VESA 2.0 extensions for your legacy VESA 1.2 card. You may be able to download a TSR type program that will run from DOS, and used in cojunction with loadlin, can help configure the card for the appropriate graphic console modes. Note that this will not always work, as an example some Cirrus Logic cards such as the VLB 54xx series are mapped to a range of memory addresses (for example, within the 15MB–16MB range) for frame buffering which preludes these from being used successfully with systems that have more than 32MB of memory. There is a way to make this work, i.e. if you have a BIOS option to leave a memory hole at 15MB−16MB range, it might work, Linux doesn't support the use of memory holes. However there are patches for this option though [Who has these and where do one gets them from?]. If you wish to experiment with this option, there are plenty of TSR style programs available, a prime example is UNIVBE, which can be found on the Internet.

Alternatively, you may be able to download kernel patches to allow your VESA 1.2 card to work with the VESA framebuffer driver. For example, there are patches for use with older S3 boards (such as S3 Trio, S3 Virge) that supports VESA 1.2. For these cards, you can pick up patches from

`ftp://ccssu.crimea.ua/pub/linux/kernel/v2.2/unofficial/s3new.diff.gz`

5.2 How do I activate the vesafb drivers?

Assuming you are using menuconfig, you will need to do the following steps:

If your processor (on Intel platforms) supports MTRRs, enable this. It speeds up memory copies between the processor and the graphic card, but not strictly necessary. You can of course, do this after you have the console device working.

IMPORTANT: For 2.1.x kernels, go into the Code Maturity Level menu, and enable the prompt for development andor incomplete drivers. This is no longer necessary for the 2.2.x kernels.

Go into the Console Drivers menu, and enable the following:

- VGA Text Console
- Video Selection Support
- Support for frame buffer devices (experimental)
- VESA VGA Graphic console
- Advanced Low Level Drivers
- Select Mono, 2bpp, 4bpp, 8bpp, 16bpp, 24bpp and 32bpp packed pixel drivers

VGA Chipset Support (text only) – vgafb – used to be part of the list above, but it has been removed as it is now deprecated and no longer supported. It will be removed shortly. Use VGA Text Console (fbcon) instead. VGA Character/Attributes is only used with VGA Chipset Support, and doesn't need to be selected.

Ensure that the Mac variable bpp packed pixel support is not enabled. Linux kernel release 2.1.111 (and 112) seemed to enable this automatically if Advanced Low Level Drivers was selected for the first time. This no longer happens with 2.1.113.

There is also the option to compile in fonts into memory, but this isn't really necessary, and you can always use kbd−0.99's (see section on fonts) setfont utility to change fonts by loading fonts into the console device.

Make sure these aren't going to be modules. [Not sure if it's possible to build them as modules yet – please correct me on this]

You'll need to create the framebuffer device in /dev. You need one per framebuffer device, so all you need to do is to type in mknod /dev/fb0 c 29 0 for the first one. Subsequent ones would be in multiples of 32, so for example to create /dev/fb1, you would need to type in mknod /dev/fb1 c 29 32, and so on up to the eighth framebuffer device (mknod /dev/fb7 c 29 224)

Then rebuild the kernel, modify /etc/lilo.conf to include the VGA=ASK parameter, and run lilo, this is required in order for you to be able to select the modes you wish to use.

Here's a sample LILO configuration (taken from my machine)

```
# LILO configuration file
boot = /dev/hda3
delay = 30
prompt
vga = ASK # Let user enter the desired modes
image = /vmlinuz
  root = /dev/hda3
  label = Linux
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

Reboot the kernel, and as a simple test, try entering 0301 at the VGA prompt (this will give you 640x480 @ 256), and you should be able to see a cute little Penguin logo.

Note, that at the VGA prompt, you're required to type in the number in the format of "0" plus the 3 digit figure, and miss out the 'x'. This isn't necessary if you're using LILO.

Once you can see that's working well, you can explore the various VESA modes (see below) and decide on the one that you like the best, and hardwire that into the "VGA=x" parameter in lilo.conf. When you have chosen the one you like the best, look up the equivalent hexadecimal number from the table below and use that (i.e. for 1280x1024 @ 256, you just use "VGA=0x307"), and re−run lilo. That's all there it is to it. For further references, read the LoadLin/LILO HOWTOs.

NOTE! vesafb does not enable scrollback buffering as a default. You will need to pass to the kernel the option to enable it. Use video=vesa:ypan or video=vesa:ywrap to activate it. Both does the same thing, but in different ways. ywrap is a lot faster than ypan but may not work on slightly broken VESA 2.0 graphic cards. ypan is slower than ywrap but a lot more compatible. This option is only present in kernel 2.1.116 and above. Earlier kernels did not have the ability to allow scrollback buffering in vesafb.

5.3 What VESA modes are available to me?

This really depends on the type of VESA 2.0 compliant graphic card that you have in your system, and the amount of video memory available. This is just a matter of testing which modes work best for your graphic card.

The following table shows the mode numbers you can input at the VGA prompt or for use with the LILO program. (actually these numbers are plus 0x200 to make it easier to refer to the table)

```
Colours   640x400 640x480 800x600 1024x768 1152x864 1280x1024 1600x1200
--------+--------------------------------------------------------------
 4 bits |    ?       ?     0x302      ?        ?        ?         ?
 8 bits |  0x300   0x301   0x303    0x305    0x161    0x307     0x31C
15 bits |    ?     0x310   0x313    0x316    0x162    0x319     0x31D
16 bits |    ?     0x311   0x314    0x317    0x163    0x31A     0x31E
24 bits |    ?     0x312   0x315    0x318      ?      0x31B     0x31F
32 bits |    ?       ?       ?        ?      0x164      ?
```

Key: 8 bits = 256 colours, 15 bits = 32,768 colours, 16 bits = 65,536 colours, 24 bits = 16.8 million colours, 32 bits − same as 24 bits, but the extra 8 bits can be used for other things, and fits perfectly with a 32 bit

PCI/VLB/EISA bus.

Additional modes are at the discretion of the manufacturer, as the VESA 2.0 document only defines modes up to 0x31F. You may need to do some fiddling around to find these extra modes.

5.4 Got a Matrox card?

If you've got a Matrox graphic card, you don't actually need vesafb, you need the matroxfb driver instead. This greatly enhances the capabilities of your card. Matroxfb will work with Matrox Mystique Millennium I & II, G100 and G200. It also supports multiheaded systems (that is, if you have two Matrox cards in your machine, you can use two displays on the same machine!). To configure for Matrox, you will need to do the following:

You might want to upgrade the Matrox BIOS though, you can download the BIOS upgrade from

```
http://www.matrox.com/mgaweb/drivers/ftp_bios.htm
```

Beware that you will need DOS to do this.

Go into the Code Maturity Level menu, and enable the prompt for development and/or incomplete drivers [note this may change for future kernels – when this happens, this HOWTO will be revised]

Go into the Console Drivers menu, and enable the following:

- VGA Text Console
- Video Selection Support
- Support for frame buffer devices (experimental)
- Matrox Acceleration
- Select the following depending on the card that you have
    - Millennium I/II support
    - Mystique support
    - G100/G200 support
- Enable Multihead Support if you want to use more than one Matrox card
- Advanced Low Level Drivers
- Select Mono, 2bpp, 4bpp, 8bpp, 16bpp, 24bpp and 32bpp packed pixel drivers

Rebuild your kernel. Then you will need to modify your lilo.conf file to enable the Matroxfb device. The quickest and simplest way is re–use mine.

```
# LILO configuration file
boot = /dev/hda3
delay = 30
prompt
vga = 792    # You need to do this so it boots up in a sane state
# Linux bootable partition config begins
image = /vmlinuz
  append = "video=matrox:vesa:440" # then switch to Matroxfb
  root = /dev/hda3
  label = Linux
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

Lastly, you'll need to create the framebuffer device in /dev. You need one per framebuffer device, so all you need to do is to type in mknod /dev/fb0 c 29 0 for the first one. Subsequent ones would be in multiples of 32, so for example to create /dev/fb1, you would need to type in mknod /dev/fb1 c 29 32, and so on up to the eight framebuffer device (mknod /dev/fb7 c 29 224)

And that should be it! [NOTE: Is anyone using this multiheaded support, please get in touch with me ASAP – I need to talk to you about it so I can document it!

5.5 Got a Permedia card?

Permedia cards cannot be used with the vesafb driver, but fortunately, there is the Permedia framebuffer driver available to use. Assuming you are using menuconfig, do the following:

Go into the Code Maturity Level menu, and enable the prompt for development and/or incomplete drivers [note this may change for future kernels – when this happens, this HOWTO will be revised]

Go into the Console Drivers menu and select the following:

- VGA Text Console
- Video Selection Support
- Support for frame buffer devices (experimental)
- Permedia2 support (experimental)
- Generic Permedia2 PCI board support
- Advanced Low Level Drivers
- Select Mono, 2bpp, 4bpp, 8bpp, 16bpp, 24bpp and 32bpp packed pixel drivers
- Optionally, select the following, if you wish to use the compiled in fonts
    - Select compiled–in fonts
    - Select Sparc console 12x22 font

Rebuild your kernel. Then you will need to modify your lilo.conf file to enable the pm2fb device. The quickest and simplest way is re–use the following

```
# LILO configuration file
boot = /dev/hda3
delay = 30
prompt
vga = 792    # You need to do this so it boots up in a sane state
# Linux bootable partition config begins
image = /vmlinuz
  append = "video=pm2fb:mode:1024x768-75,font:SUN12x22,ypan" # then switch to pm2fb
  root = /dev/hda3
  label = Linux
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

The line "pm2fb:mode:1024x768–75,font:SUN12x22,ypan" indicates you are selecting a 1024x768 mode at 75Hz, with the SUN12x22 font selected (if you did select it), including ypan for scrollback support. You may select other modes if you desire.

Lastly, you'll need to create the framebuffer device in /dev. You need one per framebuffer device, so all you need to do is to type in mknod /dev/fb0 c 29 0 for the first one. Subsequent ones would be in multiples of 32, so for example to create /dev/fb1, you would need to type in mknod /dev/fb1 c 29 32, and so on up to the eight framebuffer device (mknod /dev/fb7 c 29 224)

For more information on the other features of the Permedia framebuffer driver, point your browser at:

```
http://www.cs.unibo.it/~nardinoc/pm2fb/index.html
```

video=pm2fb:[option[,option[,option...]]]

where option is one of the following

- off to disable the driver.
- mode:resolution to set the console resolution. The modes have been taken from the fb.modes.ATI file in Geert's fbset package. The depth for all the modes is 8bpp. This is the list of the available modes:
    - 640x480−(60,72,75,90,100)
    - 800x600−(56,60,70,72,75,90,100)
    - 1024x768−(60,70,72,75,90,100,illo) illo=80KHz 100Hz
    - 1152x864−(60,70,75,80)
    - 1280x1024−(60,70,74,75)
    - 1600x1200−(60,66,76)
- The default resolution is 640x480−60.
- font:font name to set the console font. Example: font:SUN12x22
- ypan sets the current virtual height as big as video memory size permits.
- oldmem this option is for CybervisionPPC users only. Specify this if your board has Fujitsu SGRAMs mounted on (all CVisionPPCs before 30−Dec−1998).
- virtual (temporary) specify this if the kernel remaps the PCI regions on your platform.

5.6 Got a ATI card?

[Note: This information is at best, only second−hand or third−hand, since I don't have an ATI card to test it with. Feel free to correct me if I am wrong or flame me!] 8)

ATI cards can be used with the vesafb driver, but you may or may not have problems, depending on how horribly broken the card is. Fortunately, there is the atyfb framebuffer driver available to use. Assuming you are using menuconfig, do the following:

Go into the Code Maturity Level menu, and enable the prompt for development and/or incomplete drivers [note this may change for future kernels − when this happens, this HOWTO will be revised]

Go into the Console Drivers menu and select the following:

- VGA Text Console
- Video Selection Support
- Support for frame buffer devices (experimental)
- ATI Mach64 display support
- Advanced Low Level Drivers
- Select Mono, 2bpp, 4bpp, 8bpp, 16bpp, 24bpp and 32bpp packed pixel drivers
- Optionally, select the following, if you wish to use the compiled in fonts
    - Select compiled−in fonts
    - Select Sparc console 12x22 font

Rebuild your kernel. Then you will need to modify your lilo.conf file to enable the atyfb device. The quickest and simplest way is re−use the following

```
# LILO configuration file
boot = /dev/hda3
```

```
delay = 30
prompt
vga = 792    # You need to do this so it boots up in a sane state
# Linux bootable partition config begins
image = /vmlinuz
  append = "video=atyfb:mode:1024x768,font:SUN12x22"
  root = /dev/hda3
  label = Linux
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

The line "atyfb:mode:1024x768,font:SUN12x22" indicates you are selecting a 1024x768 mode.

Lastly, you'll need to create the framebuffer device in /dev. You need one per framebuffer device, so all you need to do is to type in mknod /dev/fb0 c 29 0 for the first one. Subsequent ones would be in multiples of 32, so for example to create /dev/fb1, you would need to type in mknod /dev/fb1 c 29 32, and so on up to the eight framebuffer device (mknod /dev/fb7 c 29 224)

video=atyfb:[option[,option[,option...]]]

where option is one of the following

- font:STRING selects the built−in font (compiled into the kernel)
- noblink Turns off blinking
- noaccel Disables acceleration
- vram:ULONG Tells the atyfb driver how much memory you have
- pll:ULONG Unknown
- mclk:ULONG Unknown
- vmode:ULONG Unknown
- cmode:ULONG − sets depth − 0, 8, 15, 16, 24 and 32

5.7 Which graphic cards are VESA 2.0 compliant?

This lists all the graphic cards that are known to work with the vesafb device:

- ATI PCI VideoExpression 2MB (max. 1280x1024 @ 8bit)
- ATI PCI All−in−Wonder
- Matrox Millennium PCI − BIOS v3.0
- Matrox Millennium II PCI − BIOS v1.5
- Matrox Millennium II AGP − BIOS v1.4
- Matrox Millennium G200 AGP − BIOS v1.3
- Matrox Mystique & Mystique 220 PCI − BIOS v1.8
- Matrox Mystique G200 AGP − BIOS v1.3
- Matrox Productiva G100 AGP − BIOS v1.4
- All Riva 128 based cards
- Diamond Viper V330 PCI 4MB
- Genoa Phantom 3D/S3 ViRGE/DX
- Hercules Stingray 128/3D with TV output
- Hercules Stingray 128/3D without TV output − needs BIOS upgrade (free from support@hercules.com)
- SiS 6326 PCI/AGP 4MB
- STB Lightspeed 128 (Nvida Riva 128 based) PCI
- STB Velocity 128 (Nvida Riva 128 based) PCI

- Jaton Video−58P ET6000 PCI 2MB−4MB (max. 1600x1200 @ 8bit)
- Voodoo2 2000

This list is composed of on−board chipsets on systems' motherboards:

- Trident Cyber9397
- SiS 5598

This list below blacklists graphic cards that doesn't work with the vesafb device:

- TBA

5.8 Can I make vesafb as a module?

As far as is known, vesafb can't be modularised, although at some point in time, the developer of vesafb may decide to modify the sources for modularising. Note that even if modularising is possible, at boot time you will not be able to see any output on the display until vesafb is *modprobed*. It's probably a lot wiser to leave it in the kernel, for these cases when there are booting problems.

5.9 How do I modify the cursor?

[Taken from VGA−softcursor.txt − thanks Martin Mares!]

Linux now has some ability to manipulate cursor appearance. Normally, you can set the size of hardware cursor (and also work around some ugly bugs in those miserable Trident cards−−see #define TRIDENT_GLITCH in drivers/char/ vga.c). In case you enable "Software generated cursor" in the system configuration, you can play a few new tricks: you can make your cursor look like a non−blinking red block, make it inverse background of the character it's over or to highlight that character and still choose whether the original hardware cursor should remain visible or not. There may be other things I have never thought of.

The cursor appearance is controlled by a

```
<ESC>[?1;2;3c
```

escape sequence where 1, 2 and 3 are parameters described below. If you omit any of them, they will default to zeroes.

Parameter 1 specifies cursor size (0=default, 1=invisible, 2=underline, ..., 8=full block) + 16 if you want the software cursor to be applied + 32 if you want to always change the background colour + 64 if you dislike having the background the same as the foreground. Highlights are ignored for the last two flags.

The second parameter selects character attribute bits you want to change (by simply XORing them with the value of this parameter). On standard VGA, the high four bits specify background and the low four the foreground. In both groups, low three bits set colour (as in normal colour codes used by the console) and the most significant one turns on highlight (or sometimes blinking−−it depends on the configuration of your VGA).

The third parameter consists of character attribute bits you want to set. Bit setting takes place before bit toggling, so you can simply clear a bit by including it in both the set mask and the toggle mask.

To get normal blinking underline, use: echo −e '\033[?2c' To get blinking block, use: echo −e '\033[?6c' To get red non−blinking block, use: echo −e '\033[?17;0;64c'

6. Using framebuffer devices on Atari m68k platforms

This section describes framebuffer options on Atari m68k platforms.

6.1 What modes are available on Atari m68k platforms?

```
Colours   320x200 320x480 640x200 640x400 640x480 896x608 1280x960
--------+---------------------------------------------------------
 1 bit  |                         sthigh   vga2    falh2   tthigh
 2 bits |                 stmid            vga4
 4 bits | stlow                         ttmid/vga16 falh16
 8 bits |         ttlow                   vga256
```

`ttlow, ttmid and tthigh` are only used by the TT, whilst `vga2, vga4, vga15, vga256, falh3 and falh16` are only used by the Falcon.

When used with the kernel option `video=xxx`, and no suboption is given, the kernel will probe for the modes in the following order until it finds a mode that is possible with the given hardware:

- `ttmid`
- `tthigh`
- `vga16`
- `sthigh`
- `stmid`

You may specify the particular mode you wish to use, if you don't wish to auto−probe for the modes you desire. For example, `video=vga16` gives you a 4 bit 640x480 display.

6.2 Additional suboptions on Atari m68k platforms

There are a number of suboptions available with the `video=xxx` parameter:

- `inverse` − inverts the display so that the background/foreground colours are reversed. Normally the background is black, but with this suboption, it gets sets to white.
- `font` − sets the font to use in text modes. Currently you can only select `VGA8x8`, `VGA8x16`, `PEARL8x8`. The default is to use the `VGA8x8` only if the vertical size of the display is less than 400 pixels, otherwise it defaults to `VGA8x16`.
- `internal` − a very interesting option. See the next section for information.
- `external` − as above.
- `monitorcap` − describes the capabilities for multisyncs. DON'T use with a fixed sync monitor!

6.3 Using the internal suboption on Atari m68k platforms

Syntax: `internal:(xres);(yres)[;(xres_max);(yres_max);(offset)]`

This option specifies the capabilities of some extended internal video hardware, i.e OverScan modes.

(xres) and (yres) gives the extended dimensions of the screen.

If your OverScan mode needs a black border, you'll need to write the last three arguments of the internal: suboption. (xres_max) is the maximum line length that the hardware allows, (yres_max) is the maximum number of lines, and (offset) is the offset of the visible part of the screen memory to its physical start, in bytes.

Often extended internal video hardware has to be activated, for this you will need the "switches=*" options. [Note: Author would like extra information on this, please. The m68k documentation in the kernel isn't clear enough on this point, and he doesn't have an Atari! Examples would be helpful too]

6.4 Using the external suboption on Atari m68k platforms

Syntax:
`external:(xres);(yres);(depth);(org);(scrmem)[;(scrlen)[;(vgabase)[;(colw)[;(coltype)[;(xres_virtual)]]]]]`

This is quite complicated, so this document will attempt to explain as clearly as possible, but the Author would appreciate if someone would give this a look over and see that he hasn't fscked something up! :o)

This suboption specifies that you have an external video hardware (most likely a graphic board), and how to use it with Linux. The kernel is basically limited to what it knows of the internal video hardware, so you have to supply the parameters it needs in order to be able to use external video hardware. There are two limitations; you must switch to that mode before booting, and once booted, you can't change modes.

The first three parameters are obvious; gives the dimensions of the screen as pixel height, width and depth. The depth supplied should be the number of colours is 2^n that of the number of planes required. For example, if you desire to use a 256 colour display, then you need to give 8 as the depth. This depends on the external graphic hardware, though so you will be limited by what the hardware can do.

Following from this, you also need to tell the kernel how the video memory is organised – supply a letter as the (org) parameter

- n – use normal planes, i.e one whole plane after another
- i – use interleaved planes, i.e. 16 bits of the first plane, then the 16 bits of the next plane and so on. Only built–in Atari video modes uses this – and there are no graphic card that supports this mode.
- p – use packed pixels, i.e consecutive bits stands for all planes for a pixel. This is the most common mode for 256 colour displays on graphic cards.
- t – use true colour, i.e this is actually packed pixels, but does not require a colour lookup table like what other packed pixel modes uses. These modes are normally 24 bit displays – gives you 16.8 million colours.

However, for monochrome modes, the (org) parameter has a different meaning

- n – use normal colours, i.e 0=white, 1=black
- i – use inverted colours, i.e. 0=black, 1=white

The next important item about the video hardware is the base address of the video memory. That is given by the (scrmem) parameter as a hexadecimal number with an 0x prefix. You will need to find this out from the documentation that comes with your external video hardware.

The next paramter `(scrlen)` tells the kernel about the size of the video memory. If it's missing, this is calculated from the `(xres)`, `(yres)` and `(depth)` parameters. It's not useful to write a value here these days anyway. To leave this empty, give two consecutive semicolons if you need to give the `(vgabase)` parameter, otherwise, just leave it.

The `(vgabase)` parameter is optional. If it isn't given, the kernel can't read/write any colour registers of the video hardware, and thus you have to set up the appropriate colours before you boot Linux. But if your card is VGA compatible, you can give it the address where it can locate the VGA register set so it can change the colour lookup tables. This information can be found in your external video hardware documentation. To make this *clear*, `(vgabase)` is the *base* address, i.e a 4k aligned address. For reading/writing the colour registers, the kernel uses the address range between `(vgabase) + 0x3c7` and `(vgabase) + 0x3c9`. This parameter is given in hexadecimal and must have a `0x` prefix, just like `(scrmem)`.

`(colw)` is only meaningful, if the `(vgabase)` parameter is specified. It tells the kernel how wide each of the colour register is, i.e the number of bits per single colour (red/green/blue). Default is usually 6 bits, but it is also common to specify 8 bits.

`(coltype)` is used with the `(vgabase)` parameter, it tells the kernel about the colour register model of your graphic board. Currently the types supported are `vga` and `mv300`. `vga` is the default.

`(xres_virtual)` is only required for the ProMST/ET4000 cards where the physical linelength differs from the visible length. With ProMST, you need to supply 2048, whilst for ET4000, it depends on the initialisation of the video board.

---

7. Using framebuffer devices on Amiga m68k platforms

This section describes the options for Amigas, which are quite similiar to that for the Atari m68k platforms.

7.1 What modes are available for Amiga m68k platforms?

This depends on the chipset used in the Amiga. There are three main ones; `OCS, ECS and AGA` which all uses the colour frame buffer device.

- NTSC modes
    - `ntsc` – 640x200
    - `ntsc-lace` – 640x400
- PAL modes
    - `pal` – 640x256
    - `pal-lace` – 640x512
- ECS modes – 2 bit colours on ECS, 8 bit colours on AGA chipsets only.
    - `multiscan` – 640x480
    - `multiscan-lace` – 640x960
    - `euro36` – 640x200
    - `euro36-lace` – 640x400
    - `euro72` – 640x400
    - `euro72-lace` – 640x800
    - `super72` – 800x300
    - `super72-lace` – 800x600

    - `dblntsc` – 640x200
    - `dblpal` – 640x256
    - `dblntsc-ff` – 640x400
    - `dblntsc-lace` – 640x800
    - `dblpal-ff` – 640x512
    - `dblpal-lace` – 640x1024
- VGA modes – 2 bit colours on ECS, 8 bit colours on AGA chipsets only.
    - `vga` – 640x480
    - `vga70` – 640x400

7.2 Additional suboptions on Amiga m68k platforms

These are similar to the Atari m68k suboptions. They are:

- `depth` – specifies the pixel bit depth.
- `inverse` – does the same thing as the Atari suboption.
- `font` – does the same thing as the Atari suboption, although the `PEARL8x8` font is used instead of `VGA8x8` font, if the display size is less than 400 pixel wide.
- `monitorcap` – specifies the capabilities of the multisync monitor. Do not use with fixed sync monitors.

7.3 Supported Amiga graphic expansion boards

- `Phase5 CyberVision 64` (S3 Trio64 chipset)
- `Phase5 CyverVision 64-3D` (S3 ViRGE chipset)
- `MacroSystems RetinaZ3` (NCR 77C32BLT chipset)
- `Helfrich Piccolo, SD64, GVP ECS Spectrum, Village Tronic Picasso III`I+ and IV/ (Cirrus Logic GD542x/543x)

---

8. Using framebuffer devices on Macintosh m68k platforms

Currently, the framebuffer device implemented only supports the mode selected in MacOS before booting into Linux, also supports 1, 2, 4 and 8 bit colours modes.

Framebuffer suboptions are selected using the following syntax

```
video=macfb:<font>:<inverse>
```

You can select fonts such as VGA8x8, VGA8x16 and 6x11 etc. The inverse option allows you to use reverse video.

---

9. Using framebuffer devices on PowerPC platforms

The author would love to receive information on the use of framebuffers on this platform.

---

10. Using framebuffer devices on Alpha platforms

10.1 What modes are available to me?

So far, there is only the TGA PCI card – which only does 80x30 with a resolution of 640x480 at either 8 bits or 24/32 bits.

10.2 Which graphic cards can work with the frambuffer device?

This lists all the graphic cards that are known to work:

- `DEC TGA PCI (DEC21030)` – 640x480 @ 8 bit or 24/32 bit versions

---

11. Using framebuffer devices on SPARC platforms

11.1 Which graphic cards can work with the framebuffer device?

This lists all the graphic cards available:

- MG1/MG2 – SBus or integrated on Sun3 – max. 1600x1280 @ mono (BWtwo)
- CGthree – Similar to MG1/MG2 but supports colour – max resolution ?
- GX – SBus – max. 1152x900 @ 8bit (CGsix)
- TurboGX – SBus – max. 1152x900 @ 8 bit (CGsix)
- SX – SS10/SS20 only – max. 1280x1024 @ 24 bit – (CGfourteen)
- ZX(TZX) – SBus – accelerated 24bit 3D card – max resolution ? (Leo)
- TCX – AFX – for Sparc 4 only – max. 1280x1024 @ 8bit
- TCX(S24) – AFX – for Sparc 5 only – max. 1152x900 @ 24bit
- Creator – SBus – max. 1280x1024 @ 24bit (FFB)
- Creator3D – SBus – max. 1920x1200 @ 24bit (FFB)
- ATI Mach64 – accelerated 8/24bit for Sparc64 PCI only

There is the option to use the PROM to output characters to the display or to a serial console.

Also, have a look at the Sparc Frame Buffer FAQ at

`http://c3-a.snvl1.sfba.home.com/Framebuffer.html`

11.2 Configuring the framebuffer devices

During make config, you need to choose whether to compile `promcon` and/or `fbcon`. You can select both, but if you do this, you will need to set the kernel flags to select the device. `fbcon` always takes precedence if not set. If `promcon` is not selected in, on boot up, it defaults to `dummycon`. If `promcon` is selected, it will use this device. Once the buses are booted, and `fbcon` is compiled in, the kernel probes for the above framebuffers and will use `fbcon`. If there is no framebuffer devices, it will default to `promcon`.

Here are the kernel options

```
video=sbus:options
        where options is a comma separated list:
                nomargins       sets margins to 0,0
                margins=12x24   sets margins to 12,24 (default is computed
from resolution)
                off             don't probe for any SBus/UPA framebuffers
                font=SUN12x22   use a specific font
```

So for example, booting with

```
 video=sbus:nomargins,font=SUN12x22
```

gives you a nice fast text console with a text resolution of 96x40, looks similar to a Solaris console but with colours and virtual terminals just like on the Intel platform.

If you want to use the `SUN12x22` font, you need to enable it during make config (disable the `fontwidth != 8` option). The accelerated framebuffers can support any font width between 1 to 16 pixels, whilst dumb frame buffers only supports 4, 8, 12 and 16 pixel font widths.

It is recommended that you grab a recent consoletools packages.

---

12. Using framebuffer devices on MIPS platforms

There is no need to change anything for this platform, this is all handled for you automatically. Indys in particular are hardwired to use a console size of 160x64. However, moves are afoot to rewrite the console code for these Indys, so keep an eye on this section.

---

13. Using framebuffer devices on ARM platforms

13.1 Netwinders

For the Netwinders (which uses the ARM SA110 RISC chip – a lovely British processor), there are two versions of the Cyber2000 framebuffer driver – one for 2.0.x kernels and one for 2.2.x kernels. It is quite straightforward to enable and use this driver on both kernels, however, the older version is hardcoded for depth and resolution (blech), but the good news is that the newer version in the 2.2.x kernels is much more flexible, but currently in a state of flux as it is still in development. To get this up and running, your best bet is to read the documentation that comes with the ARM port of the kernel sources.

The Netwinders uses a VGA compatible chipset, but unfortunately noone has ported vgafb to it yet. That might happen if someone has some time on their hands. [I would do it if someone would give me a NetWinder to play with]

13.2 Acorn Archimedes

Acorns have always had framebuffer support since the Linux 1.9.x days. However the Acornfb driver in 2.2.x is totally new since the generic framebuffer interface changed during the development of 2.1.x kernels (which, of course, became 2.2.x). As previously, it is a simple matter to activate the driver and set depths and resolutions.

13.3 Other ARM ports (SA 7110s et. al)

Surprisingly, there is even a framebuffer driver for the Psion 5 and the Geofox! I have been told that it displays the Penguin quite well. [Someone please donate me a Psion 5!]

14. Using multi−headed framebuffers

This part of the document was very kindly donated by Frederick A. Niles, who retains all rights to the information contained herewith this section of the HOWTO.

14.1 Introduction

The main goal of this document is to get you started with running a dual head configuration of Linux. While this process is pretty straight forward there are numerous things that one can do wrong along the way.

The example I concentrate on is getting an X−server running on a second monitor. I find this nice as you can usually find old large 19" to 21" fixed frequency monitors around that people are giving away because they can't use them. This way you can boot off a small multisync and then use X on a nice big monitor.

Please understand dual head support is currently developing so this information changes rapidly. Anything in this document could be out of date or just plain incorrect by the time you are reading this.

** WARNING ** This document was written before any XFree86 4.0 release. If you are reading this and XFree86 4.0 is already released many things may have changed. Try getting a newer version of this document if it's available.

14.2 Feedback

Feedback is most certainly welcome for this document. Without your submissions and input, this document wouldn't exist. So, please post your additions, comments and criticisms to: Frederick.A.Niles@gsfc.nasa.gov.

14.3 Contributors

The following people have contributed to this mini−HOWTO.

* Petr Vandrovec `vandrove@vc.cvut.cz`

* Andreas Ehliar `ehliar@lysator.liu.se` (x2x)

* Marco Bizzarri `m.bizzarri@icube.it` (multiple X servers)

14.4 Standard Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies that could be damaging to your system. Proceed with caution, and although this is highly unlikely, I don't take any responsibility for that.

14.5 Copyright Information

14.6 What hardware is supported?

Most video cards assume they will be the only one in the system and are permanently set with the addressing of the primary display adapter. There are a few exceptions.

* Matrox cards: This includes Matrox Millennium, Matrox Millennium II, Matrox Mystique, Matrox Mystique 220, Matrox Productiva G100, Matrox Mystique G200, Matrox Millennium G200 and Matrox Marvel G200 video cards

* MDA: This includes monochrome Hercules graphics adapter among others. This for text only second head support.

Note: it's only the second adapter that has to be one of the above.

14.7 Commercial support

This mini−HOWTO in primarily concerned with free software. However, there are commercial X servers with multi−head support. These include Metro Link's (www.metrolink.com) Metro−X and Xi Graphics' (www.xig.com) Accelerated−X.

14.8 Getting all the stuff.

You'll need the following patches and programs:

* "fbset" program try:

`http://www.cs.kuleuven.ac.be/~geert/bin/`

(note: this program comes with RedHat 6.0)

* "fbaddon" Matrox dual head patches for Linux kernel try:

`ftp://platan.vc.cvut.cz/pub/linux/matrox-latest/`

* "con2fb" program try:

`ftp://platan.vc.cvut.cz/pub/linux/matrox-latest/`

* The X11 frame buffer server XF86_FBDev. This is a standard part of XFree86 3.3.1.

14.9 Getting Started

The first thing you'll need to do is to patch a copy of the Linux source with the "fbaddon" patch. Then you need to configure the kernel and turn on frame buffer support. If you have Matrox cards turn on Matrox unified accelerated driver support as well as the particular type of card you have. Don't turn on VESA frame buffer support. It can cause a conflict. Do turn on multi−head support (obviously). Build the kernel and reboot.

Now you need to install the "fbset" program and carefully read all the documentation on how to adjust the settings. Using a "/etc/fb.modes" file is highly recommended once you've decided on your settings. The fbset program includes a Perl script to convert your XF86Config file to fb.modes settings. I've included my octave/Borne shell script to convert your XF86Config file in Appendix A & B.

You need to get comfortable with using the frame buffer device on one monitor, understanding any issues that can arise from your set up that have nothing to do with multi−head support. This can save a lot of head scratching later.

I'm going to concentrate my explanation on getting X running on the second monitor as doing most other configurations will just be a obvious subset of the procedure.

Move a console over...

Compile the "con2fb" program. If you run it without any arguments you'll get the following usage message:

"usage: con2fb fbdev console".

Thus, an example command would be "con2fb /dev/fb1 /dev/tty6" to move virtual console number six over to the second monitor. Use Ctrl−Alt−F6 to move over to that console and see that it does indeed show up on the second monitor.

Use "fbset" to adjust the setting on this second monitor

Only set the "fbset" settings on the monitor you run the "fbset" command on. Therefore, you must be careful to use the "−fb" flag on the second monitor. In particular, if you do nothing else you'll probably want to at least set the virtual vertical resolution to your actually vertical resolution.

e.g. "fbset –fb /dev/fb1 –vyres 600"

This will seriously slow down text mode, but X will be obnoxious without it.

Set up X for Frame Buffer support.

The framebuffer.txt file explains this better than I can, but here's the two important points.

Make sure you set the link for "X" to point to "XF86_FBDev".

Next you need to add a monitor section to your XF86Config file for the frame buffer device. Here's an example:

```
# The Frame Buffer server

Section "Screen"
    Driver      "fbdev"
    Device      "Millennium"
    Monitor     "NEC MultiSync 5FGp"
    Subsection "Display"
        Depth       8
        Modes       "default"
        ViewPort    0 0
    EndSubsection
    Subsection "Display"
        Depth       16
        Modes       "default"
        ViewPort    0 0
    EndSubsection
    Subsection "Display"
        Depth       24
        Modes       "default"
        ViewPort    0 0
    EndSubsection
    Subsection "Display"
        Depth       32
        Modes       "default"
        ViewPort    0 0
    EndSubsection
EndSection
```

Use the "default" modes as I don't think any other one will work with the Matrox frame buffer.

Try starting the X server on the second monitor.

Set the variable FRAMEBUFFER to the second frame buffer.

"export FRAMEBUFFER=/dev/fb1"

or

"setenv FRAMEBUFFER /dev/fb1"

You need to start the X server so that it both matches the selected color depth and it appears on the same monitor you start the X server from.

e.g. "startx −− :0 −bpp 16 vt06"

This example will start the "zeroth" X server on virtual console six with 16 bit color. Using ":1" when launching another X server for the other frame buffer will allow you to have two X servers running.

14.10 Summary

The steps involved in getting an X server running on a second monitor can be summarized as follows:

* Get the kernel patch, fbset, and con2fb.

* Patch the kernel, configure, rebuild, and reboot.

* Add XF86_FBDev section to XF86Config file and set X link.

Then each time you reboot:

* Move a console over. e.g. "con2fb /dev/fb1 /dev/tty6"

* Adjust the settings e.g. "fbset −fb /dev/fb1 1280x1024"

* Set the FRAMEBUFFER. e.g. "export FRAMEBUFFER=/dev/fb1"

* Start the X server. e.g. "startx −− −bpp 16 vt06"

You can automate this each time you reboot via a shell alias. It must be an alias and not a shell script since it needs to detect the current console number. This is my C−shell alias to start up X on a second fixed frequency monitor:

```
alias startxfb = "
setenv FRAMEBUFFER /dev/fb\!*;     # Set the env var to the cmd arg.
con2fb $FRAMEBUFFER /dev/$tty;     # Move the fb to the current tty.
fbset -fb $FRAMEBUFFER 1280x1024@62;  # Favorite from /etc/fb.modes
startx -- :\!* -bpp 16 vt0`echo $tty | cut -dy f 2`' # X on this tty.
"
```

In my .cshrc file these are all on the same line together without the comments, but it's easier to read here with line breaks and comments inserted. I just give the number of the frame buffer as an argument and it starts right up.

I'm not sure how to do this same alias in bash. I don't know how to determine the current tty or get the arguments to an alias in bash. If someone lets me know I'll insert it here. However, you can use the "tty" command to get the name of the current VT and just make two separate aliases for each X server.

14.11 Other Notes and Problems

* Both "fbset" and "startx" MUST be run from the same frame buffer as the one being affected. This places serious limits on how much of these commands can be automated via scripts.

* XFree86 4.0 will have "proper" multi−head support, but 3.3.1 does not. You can run two servers with 3.3.1 and use x2x to switch between them however...(see the next bullet)

* The inactive frame buffer will just hold the last image of when it was active, no updates with occur.

* The monitor that's not selected doesn't always preseve it's state when not active. (But it usually does.)

* Geert Uytterhoeven (the frame buffer maintainer) and Linus Torvalds don't agree with the current "frame buffer per VT" multi−head console support changes (i.e. fbaddon) so it may never be in the mainstream kernel tree. (This was heard third hand and may be wildly untrue.)

* If you "break the rules" and start the X server (run "startx") from a different monitor, the machine can eventually crash badly with the keyboard and mouse input all mixed together.

* The documentation framebuffer.txt in the kernel source explains that you can use the Modeline settings in your XF86Config file directly when running X. Using the Matrox frame buffer seems to force the X server to drop all of those. So you can only have the one ("default") setting at at time (the same one you had in text mode).

* The XF86_FBDev is unaccelerated. However, there are patches for accelerated Matrox support at

```
http://www.in-berlin.de/User/kraxel/xfree86/
```

Getting "init level five" (i.e. xdm/gdm) to work

I have not yet figured out a way to boot with init level 5 with a dual monitor configuration (and actually have the server on either the second montior or both). While it seems easy enough to add a line to the gdm/xdm Xservers file, the constrain that you must start the X server from the same frame buffer prevents the obvious solution from working. If anyone finds a way please e−mail me and I'll add it here.

Using the x2x program.

There's a nice little program called x2x that will switch X servers for you when you get to the edge of the screen. Last known home for this program was:

```
http://ftp.digital.com/pub/DEC/SRC/x2x/
```

It's also an optional Debian package. I haven't tried it yet but some users have reported success.

Other useful commands

These are existing linux commands that are worth remembering when dealing with a multi−head configuration (especially in writing scripts).

* "chvt" will allow you to switch between virtual terminals.

* "openvt" start a program on a new virtual terminal (VT).

* "tty" will report the name of the current terminal.

Appendix A. Octave cvtmode.m script

(note the bpp setting)

```
#!/usr/bin/octave -q
bpp = 16;
DCF = sscanf(argv(1,:), "%f");
HR  = sscanf(argv(2,:), "%f");
SH1 = sscanf(argv(3,:), "%f");
SH2 = sscanf(argv(4,:), "%f");
HFL = sscanf(argv(5,:), "%f");
VR  = sscanf(argv(6,:), "%f");
SV1 = sscanf(argv(7,:), "%f");
SV2 = sscanf(argv(8,:), "%f");
VFL = sscanf(argv(9,:), "%f");
pixclock = 1000000 / DCF;
left_margin = HFL - SH2;
right_margin = SH1 - HR;
hsync_len = SH2 - SH1;

# 3) vertical timings:
upper_margin = VFL - SV2;
lower_margin = SV1 - VR;
vsync_len = SV2 - SV1;

RR = DCF / (HFL * VFL) *1e6;
HSF = DCF / HFL * 1e3;

printf("mode \"%dx%d\"\n",HR,VR);
printf("   # D: %3.2f MHz, H: %3.2f kHz, V: %2.2f Hz\n", DCF, HSF, RR);
printf("   geometry %d %d %d %d %d\n", HR, VR, HR, VR, bpp);
printf("   timings %d %d %d %d %d %d %d\n", ...
                                 pixclock, left_margin, right_margin, ...
                                 upper_margin, lower_margin, ...
                                 hsync_len, vsync_len);
printf("endmode\n");
```

Appendix B. Borne Shell script "cvtfile"

(This calls the octave script "cvtmode")

```
#!/bin/sh

# Shell script to convert XF86Config file to fb.modes file.
# Uses octave script cvtmode.m

if [ -z $1 ]; then
  FILE=/etc/X11/XF86Config
else
  FILE=$1
fi

i=1
LEN=`grep Modeline $FILE | wc -l`
while expr $i \< $LEN > /dev/null ;
do
  CURLINE=`grep Modeline $FILE | cut -d'"' -f 3-20 | head -$i | tail -1 `
  ./cvtmode.m $CURLINE
  echo " "
```

```
  i=`expr $i + 1`
done
```

15. Using/Changing fonts

To get the capability to change fonts, you need kbd−0.99. You may obtain this from

`ftp://ftp.win.tue.nl/pub/linux/utils/kbd`

One advantage of downloading and installing kbd−0.99 is that you will be able to load international fonts (i.e Euro symbol) into your console device (It is tres chic to have three symbols on my keyboard, the dollar sign, the English pound sign and the Euro sign!).

16. Changing console modes

To get the capability to change modes (i.e 640x480, 800x800 etc), you need fbset (currently fbset−19990118.tar.gz) − you may ftp it from:

`http://www.cs.kuleuven.ac.be/~geert/bin/fbset-19990118.tar.gz`

This comes with a full set of instructions on how to operate this.

17. Setting up the X11 FBdev driver

If you are not using XFree86 3.3.3.1 or later, you are urged to upgrade to XFree86 3.3.3.1 − it includes a FBdev X driver for framebuffer devices. Otherwise, follow the steps below to either download or build your own FBdev driver for older XFree86 versions such as 3.3.2, 3.3.3 etc.

Go to

`http://www.xfree86.org`

, and download the latest XServers source archive, unpack, and configure the drivers, following these steps:

- Edit xc/config/cf/xf86site.def, uncomment the #define for XF68FBDevServer
- Comment out *all* references to FB_VISUAL_STATIC_DIRECTCOLOR, as these are bogus and aren't used any more. If you are using XFree86 3.3.3.1, there is no need to do this step − as they have removed this.
- Edit xc/programs/Xserver/hw/xfree86/os−support/linux/lnx_io.c, and change K_RAW to K_MEDIUMRAW.

and then build the driver. Don't worry about the m68k references, it supports Intel platforms. Then build the whole thing − it'll take a long time though as it's a large source tree.

Alternatively, if you don't have the time to spare, you can obtain the binaries from the sites below. Please note that these are 'unofficial' builds and you use them at your risk.

For libc5, use the one at:

```
http://user.cs.tu-berlin.de/~kraxel/linux/XF68_FBDev.gz
```

For glibc2, download from these URLs.

```
http://user.cs.tu-berlin.de/~kraxel/linux/XF68_FBDev.libc6.gz
http://pobox.com/~brion/linux/fbxserver.html
```

There have been reports that X11 is non functional on certain graphic cards with this vesafb feature enabled, if this is happening, try the new XF86_FBdev driver for X11.

This driver, along with vesafb can also help run X11 in higher graphic resolutions with certain graphic chipsets which are not supported by any of the current X11 drivers. Examples are MGA G−200 et. al.

To configure the XF86_FBdev driver with your X11 system, you'll need to edit your XF86Config for the following:

```
Section "Screen"
        Driver          "FBDev"
        Device          "Primary Card"
        Monitor         "Primary Monitor"
        SubSection      "Display"
                Modes           "default"
        EndSubSection
EndSection
```

You'll also need to set XkbDisable in the keyboard section as well, or invoke the XF86_FBDev server with the '−kb' option to set up your keyboard so it works properly. If you forget to set XkbDisable, you will have to put the following lines in your .Xmodmap to straighten out the keyboard mappings. Alternatively, you can edit your xkb to reflect the list below.

This is fixed in XFree86 3.3.3.1, and it is a good idea to upgrade to this version anyway because there are quite a few bug fixes, and also, it includes FBDev as one of the drvers, as I've mentioned previously.

```
! Keycode settings required
keycode 104 = KP_Enter
keycode 105 = Control_R
keycode 106 = KP_Divide
keycode 108 = Alt_R Meta_R
keycode 110 = Home
keycode 111 = Up
keycode 112 = Prior
keycode 113 = Left
keycode 114 = Right
keycode 115 = End
keycode 116 = Down
keycode 117 = Next
keycode 118 = Insert
keycode 119 = Delete
```

You may need to do some fiddling around with this (try copying the original definition from the original X11 driver that you were using and editing the name of the driver to FBDev), but basically this is what you need to do to use the vesafb X11 driver.

Hopefully the X11 problems with supported graphic cards will be fixed in future releases.

18. How do I convert XFree86 mode−lines into framebuffer device timings?

If you have XFree86 (X11) installed on your machine, and you can use it successfully, it is a simple matter to convert the mode−lines in your XF86Config to the required timings needed by the framebuffer devices.

The framebuffer device requires the following fields

- pixclock − pixel clock in pico seconds
- left_margin − time fron sync to picture
- right_margin − time from picture to sync
- upper_margin − time from sync to picture
- lower_margin − time from picture to sync
- hsync_len − length of horizontal sync
- vsync_len − length of vertical sync

An XFree86 mode line has the following fields

```
Modeline  "1280x1024" DCF HR SH1 SH2 HFL VR SV1 SV2 VFL
```

It is necessary to do some simple calculations to translate the XF86 mode−lines into a set of framebuffer device timings. As an example, we shall examine how to convert a mode−line taken from my XF86Config file.

```
Modeline  "1280x1024" 110.00 1280 1328 1512 1712 1024 1025 1028 1054
```

First, calculate the required pixclock rate. XFree86 uses megahertz whilst framebuffer devices uses picoseconds (Why, I don't know). Divide one million by DCF. For example, 1,000,000 / 110.0 = 9090.9091

Now we need to calculate the horizontal timings.

- left_margin = HFL − SH2
- right_margin = SH1 − HR
- hsync_len = SH2 − SH1

In our example, this would be:

- left_margin = 1712 − 1512 = 200
- right_margin = 1328 − 1280 = 48
- hsync_len = 1512 − 1328 = 184

And now we need to calculate the vertical timings.

- upper_margin = VFL − SV2
- lower_margin = SV1 − VR
- vsync_len = SV2 − SV1

For our example, this would be:

* upper_margin = 1054 − 1028 = 26
* lower_margin = 1025 − 1024 = 1
* vsync_len = 1028 − 1025 = 3

Now we can use this information to set up the framebuffer for the desired mode. For example, for the matroxfb framebuffer, it requires:

```
video=matrox:xres:<>,yres:<>,depth:<>,left:<>,right:<>,hslen:<>,upper:<>,lower:<>,vslen:<>
```

I put in my /etc/lilo.conf the following line:

```
append = "video=matrox:xres:1280,yres:1024,depth:32,left:200,right:48,hslen:184,upper:26,lower:0,vslen:3"
```

Note that in this case the pixclock isn't used. It's only necessary if you don't like the default pixclock rates. You can supply this as a parameter as well. Setting the pixclock is documented in other parts of this HOWTO.

19. Changing the Linux logo

It can be customised by changing the file linux_logo.h in include/linux directory. its a c header and pretty hard to change by hand however there is a plugin available for gimp

```
http://registry.gimp.org/detailview.phtml?plugin=Linux+Logo
```

that will create one for you. all you need is a picture 80x80 with less than 224 colours. you can either let the plug in create the 3 varieties (2,16,224) or create them yourself and use them with the plug−in. it will ask you where you want to store the file and if you are game you can put it in ($SRCDIR)/include/linux/linux_logo.h. once that is finished all you need to do is recompile the kernel as usual, reboot, and if framebuffer is working you will see your new logo upon bootup.

20. Looking for further information?

For those of you interested in working with the framebuffer drivers, point your browser at

```
http://www.linux-fbdev.org
```

for information on programming.

French speakers, there is a translation at

```
http://www.freenix.org/unix/linux/HOWTO/mini/Vesafb.html
```