```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
        "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" []>

<book id="LinuxDriversAPI">
 <bookinfo>
  <title>Linux Device Drivers</title>

  <legalnotice>
   <para>
     This documentation is free software; you can redistribute
     it and/or modify it under the terms of the GNU General Public
     License as published by the Free Software Foundation; either
     version 2 of the License, or (at your option) any later
     version.
   </para>

   <para>
     This program is distributed in the hope that it will be
     useful, but WITHOUT ANY WARRANTY; without even the implied
     warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
     See the GNU General Public License for more details.
   </para>

   <para>
     You should have received a copy of the GNU General Public
     License along with this program; if not, write to the Free
     Software Foundation, Inc., 59 Temple Place, Suite 330, Boston,
     MA 02111-1307 USA
   </para>

   <para>
     For more details see the file COPYING in the source
     distribution of Linux.
   </para>
  </legalnotice>
 </bookinfo>

<toc></toc>

  <chapter id="Basics">
     <title>Driver Basics</title>
     <sect1><title>Driver Entry and Exit points</title>
!Iinclude/linux/init.h
     </sect1>

     <sect1><title>Atomic and pointer manipulation</title>
!Iarch/x86/include/asm/atomic.h
!Iarch/x86/include/asm/unaligned.h
     </sect1>

     <sect1><title>Delaying, scheduling, and timer routines</title>
!Iinclude/linux/sched.h
!Ekernel/sched.c
!Ekernel/timer.c
     </sect1>
```

```
     <sect1><title>High-resolution timers</title>
!Iinclude/linux/ktime.h
!Iinclude/linux/hrtimer.h
!Ekernel/hrtimer.c
     </sect1>
     <sect1><title>Workqueues and Kevents</title>
!Ekernel/workqueue.c
     </sect1>
     <sect1><title>Internal Functions</title>
!Ikernel/exit.c
!Ikernel/signal.c
!Iinclude/linux/kthread.h
!Ekernel/kthread.c
     </sect1>

     <sect1><title>Kernel objects manipulation</title>
<!--
X!Iinclude/linux/kobject.h
-->
!Elib/kobject.c
     </sect1>

     <sect1><title>Kernel utility functions</title>
!Iinclude/linux/kernel.h
!Ekernel/printk.c
!Ekernel/panic.c
!Ekernel/sys.c
!Ekernel/rcupdate.c
     </sect1>

     <sect1><title>Device Resource Management</title>
!Edrivers/base/devres.c
     </sect1>

  </chapter>

  <chapter id="devdrivers">
     <title>Device drivers infrastructure</title>
     <sect1><title>Device Drivers Base</title>
<!--
X!Iinclude/linux/device.h
-->
!Edrivers/base/driver.c
!Edrivers/base/core.c
!Edrivers/base/class.c
!Edrivers/base/firmware_class.c
!Edrivers/base/transport_class.c
<!-- Cannot be included, because
     attribute_container_add_class_device_adapter
 and attribute_container_classdev_to_container
     exceed allowed 44 characters maximum
X!Edrivers/base/attribute_container.c
-->
!Edrivers/base/sys.c
<!--
X!Edrivers/base/interface.c
```

```
-->
!Edrivers/base/platform.c
!Edrivers/base/bus.c
     </sect1>
     <sect1><title>Device Drivers Power Management</title>
!Edrivers/base/power/main.c
     </sect1>
     <sect1><title>Device Drivers ACPI Support</title>
<!-- Internal functions only
X!Edrivers/acpi/sleep/main.c
X!Edrivers/acpi/sleep/wakeup.c
X!Edrivers/acpi/motherboard.c
X!Edrivers/acpi/bus.c
-->
!Edrivers/acpi/scan.c
!Idrivers/acpi/scan.c
<!-- No correct structured comments
X!Edrivers/acpi/pci_bind.c
-->
     </sect1>
     <sect1><title>Device drivers PnP support</title>
!Idrivers/pnp/core.c
<!-- No correct structured comments
X!Edrivers/pnp/system.c
 -->
!Edrivers/pnp/card.c
!Idrivers/pnp/driver.c
!Edrivers/pnp/manager.c
!Edrivers/pnp/support.c
     </sect1>
     <sect1><title>Userspace IO devices</title>
!Edrivers/uio/uio.c
!Iinclude/linux/uio_driver.h
     </sect1>
  </chapter>

  <chapter id="parportdev">
     <title>Parallel Port Devices</title>
!Iinclude/linux/parport.h
!Edrivers/parport/ieee1284.c
!Edrivers/parport/share.c
!Idrivers/parport/daisy.c
  </chapter>

  <chapter id="message_devices">
        <title>Message-based devices</title>
     <sect1><title>Fusion message devices</title>
!Edrivers/message/fusion/mptbase.c
!Idrivers/message/fusion/mptbase.c
!Edrivers/message/fusion/mptscsih.c
!Idrivers/message/fusion/mptscsih.c
!Idrivers/message/fusion/mptctl.c
!Idrivers/message/fusion/mptspi.c
!Idrivers/message/fusion/mptfc.c
!Idrivers/message/fusion/mptlan.c
     </sect1>
```

```
     <sect1><title>I2O message devices</title>
!Iinclude/linux/i2o.h
!Idrivers/message/i2o/core.h
!Edrivers/message/i2o/iop.c
!Idrivers/message/i2o/iop.c
!Idrivers/message/i2o/config-osm.c
!Edrivers/message/i2o/exec-osm.c
!Idrivers/message/i2o/exec-osm.c
!Idrivers/message/i2o/bus-osm.c
!Edrivers/message/i2o/device.c
!Idrivers/message/i2o/device.c
!Idrivers/message/i2o/driver.c
!Idrivers/message/i2o/pci.c
!Idrivers/message/i2o/i2o_block.c
!Idrivers/message/i2o/i2o_scsi.c
!Idrivers/message/i2o/i2o_proc.c
     </sect1>
  </chapter>

  <chapter id="snddev">
     <title>Sound Devices</title>
!Iinclude/sound/core.h
!Esound/sound_core.c
!Iinclude/sound/pcm.h
!Esound/core/pcm.c
!Esound/core/device.c
!Esound/core/info.c
!Esound/core/rawmidi.c
!Esound/core/sound.c
!Esound/core/memory.c
!Esound/core/pcm_memory.c
!Esound/core/init.c
!Esound/core/isadma.c
!Esound/core/control.c
!Esound/core/pcm_lib.c
!Esound/core/hwdep.c
!Esound/core/pcm_native.c
!Esound/core/memalloc.c
<!-- FIXME: Removed for now since no structured comments in source
X!Isound/sound_firmware.c
-->
  </chapter>

  <chapter id="uart16x50">
     <title>16x50 UART Driver</title>
!Iinclude/linux/serial_core.h
!Edrivers/serial/serial_core.c
!Edrivers/serial/8250.c
  </chapter>

  <chapter id="fbdev">
     <title>Frame Buffer Library</title>

     <para>
       The frame buffer drivers depend heavily on four data structures.
       These structures are declared in include/linux/fb.h.  They are
```

```
      fb_info, fb_var_screeninfo, fb_fix_screeninfo and fb_monospecs.
      The last three can be made available to and from userland.
    </para>

    <para>
      fb_info defines the current state of a particular video card.
      Inside fb_info, there exists a fb_ops structure which is a
      collection of needed functions to make fbdev and fbcon work.
      fb_info is only visible to the kernel.
    </para>

    <para>
      fb_var_screeninfo is used to describe the features of a video card
      that are user defined.  With fb_var_screeninfo, things such as
      depth and the resolution may be defined.
    </para>

    <para>
      The next structure is fb_fix_screeninfo. This defines the
      properties of a card that are created when a mode is set and can't
      be changed otherwise.  A good example of this is the start of the
      frame buffer memory.  This "locks" the address of the frame buffer
      memory, so that it cannot be changed or moved.
    </para>

    <para>
      The last structure is fb_monospecs. In the old API, there was
      little importance for fb_monospecs. This allowed for forbidden things
      such as setting a mode of 800x600 on a fix frequency monitor. With
      the new API, fb_monospecs prevents such things, and if used
      correctly, can prevent a monitor from being cooked.  fb_monospecs
      will not be useful until kernels 2.5.x.
    </para>

    <sect1><title>Frame Buffer Memory</title>
!Edrivers/video/fbmem.c
    </sect1>
<!--
    <sect1><title>Frame Buffer Console</title>
X!Edrivers/video/console/fbcon.c
    </sect1>
-->
    <sect1><title>Frame Buffer Colormap</title>
!Edrivers/video/fbcmap.c
    </sect1>
<!-- FIXME:
  drivers/video/fbgen.c has no docs, which stuffs up the sgml.  Comment
  out until somebody adds docs.  KAO
    <sect1><title>Frame Buffer Generic Functions</title>
X!Idrivers/video/fbgen.c
    </sect1>
KAO -->
    <sect1><title>Frame Buffer Video Mode Database</title>
!Idrivers/video/modedb.c
!Edrivers/video/modedb.c
    </sect1>
```

```
     <sect1><title>Frame Buffer Macintosh Video Mode Database</title>
!Edrivers/video/macmodes.c
     </sect1>
     <sect1><title>Frame Buffer Fonts</title>
        <para>
           Refer to the file drivers/video/console/fonts.c for more information.
        </para>
<!-- FIXME: Removed for now since no structured comments in source
X!Idrivers/video/console/fonts.c
-->
     </sect1>
  </chapter>

  <chapter id="input_subsystem">
     <title>Input Subsystem</title>
     <sect1><title>Input core</title>
!Iinclude/linux/input.h
!Edrivers/input/input.c
!Edrivers/input/ff-core.c
!Edrivers/input/ff-memless.c
     </sect1>
     <sect1><title>Polled input devices</title>
!Iinclude/linux/input-polldev.h
!Edrivers/input/input-polldev.c
     </sect1>
     <sect1><title>Matrix keyboars/keypads</title>
!Iinclude/linux/input/matrix_keypad.h
     </sect1>
     <sect1><title>Sparse keymap support</title>
!Iinclude/linux/input/sparse-keymap.h
!Edrivers/input/sparse-keymap.c
     </sect1>
  </chapter>

  <chapter id="spi">
      <title>Serial Peripheral Interface (SPI)</title>
  <para>
        SPI is the "Serial Peripheral Interface", widely used with
        embedded systems because it is a simple and efficient
        interface:  basically a multiplexed shift register.
        Its three signal wires hold a clock (SCK, often in the range
        of 1-20 MHz), a "Master Out, Slave In" (MOSI) data line, and
        a "Master In, Slave Out" (MISO) data line.
        SPI is a full duplex protocol; for each bit shifted out the
        MOSI line (one per clock) another is shifted in on the MISO line.
        Those bits are assembled into words of various sizes on the
        way to and from system memory.
        An additional chipselect line is usually active-low (nCS);
        four signals are normally used for each peripheral, plus
        sometimes an interrupt.
  </para>
  <para>
        The SPI bus facilities listed here provide a generalized
        interface to declare SPI busses and devices, manage them
        according to the standard Linux driver model, and perform
        input/output operations.
```

At this time, only "master" side interfaces are supported,
where Linux talks to SPI peripherals and does not implement
such a peripheral itself.
(Interfaces to support implementing SPI slaves would
necessarily look different.)
</para>
<para>
The programming interface is structured around two kinds of driver,
and two kinds of device.
A "Controller Driver" abstracts the controller hardware, which may
be as simple as a set of GPIO pins or as complex as a pair of FIFOs
connected to dual DMA engines on the other side of the SPI shift
register (maximizing throughput). Such drivers bridge between
whatever bus they sit on (often the platform bus) and SPI, and
expose the SPI side of their device as a
<structname>struct spi_master</structname>.
SPI devices are children of that master, represented as a
<structname>struct spi_device</structname> and manufactured from
<structname>struct spi_board_info</structname> descriptors which
are usually provided by board-specific initialization code.
A <structname>struct spi_driver</structname> is called a
"Protocol Driver", and is bound to a spi_device using normal
driver model calls.
</para>
<para>
The I/O model is a set of queued messages. Protocol drivers
submit one or more <structname>struct spi_message</structname>
objects, which are processed and completed asynchronously.
(There are synchronous wrappers, however.) Messages are
built from one or more <structname>struct spi_transfer</structname>
objects, each of which wraps a full duplex SPI transfer.
A variety of protocol tweaking options are needed, because
different chips adopt very different policies for how they
use the bits transferred with SPI.
</para>
!Iinclude/linux/spi/spi.h
!Fdrivers/spi/spi.c spi_register_board_info
!Edrivers/spi/spi.c
</chapter>

<chapter id="i2c">
<title>I<superscript>2</superscript>C and SMBus Subsystem</title>

<para>
I<superscript>2</superscript>C (or without fancy typography, "I2C")
is an acronym for the "Inter-IC" bus, a simple bus protocol which is
widely used where low data rate communications suffice.
Since it's also a licensed trademark, some vendors use another
name (such as "Two-Wire Interface", TWI) for the same bus.
I2C only needs two signals (SCL for clock, SDA for data), conserving
board real estate and minimizing signal quality issues.
Most I2C devices use seven bit addresses, and bus speeds of up
to 400 kHz; there's a high speed extension (3.4 MHz) that's not yet
found wide use.
I2C is a multi-master bus; open drain signaling is used to
arbitrate between masters, as well as to handshake and to

```
synchronize clocks from slower clients.
      </para>

      <para>
         The Linux I2C programming interfaces support only the master
         side of bus interactions, not the slave side.
         The programming interface is structured around two kinds of driver,
         and two kinds of device.
         An I2C "Adapter Driver" abstracts the controller hardware; it binds
         to a physical device (perhaps a PCI device or platform_device) and
         exposes a <structname>struct i2c_adapter</structname> representing
         each I2C bus segment it manages.
         On each I2C bus segment will be I2C devices represented by a
         <structname>struct i2c_client</structname>.  Those devices will
         be bound to a <structname>struct i2c_driver</structname>,
         which should follow the standard Linux driver model.
         (At this writing, a legacy model is more widely used.)
         There are functions to perform various I2C protocol operations; at
         this writing all such functions are usable only from task context.
      </para>

      <para>
         The System Management Bus (SMBus) is a sibling protocol.  Most SMBus
         systems are also I2C conformant.  The electrical constraints are
         tighter for SMBus, and it standardizes particular protocol messages
         and idioms.  Controllers that support I2C can also support most
         SMBus operations, but SMBus controllers don't support all the protocol
         options that an I2C controller will.
         There are functions to perform various SMBus protocol operations,
         either using I2C primitives or by issuing SMBus commands to
         i2c_adapter devices which don't support those I2C operations.
      </para>

!Iinclude/linux/i2c.h
!Fdrivers/i2c/i2c-boardinfo.c i2c_register_board_info
!Edrivers/i2c/i2c-core.c
    </chapter>

</book>
```