CANONICAL

Ubuntu BIOS/UEFI Requirements

Canonical Services Ltd.

Revision: 602c945c (public build)
Date: 2012-05-22
Release: general

Table of Contents

1. Introduction

This document outlines a set of recommendations for system firmware teams producing both legacy BIOS and UEFI firmware images for consumer systems, intended to be released with Ubuntu preinstalled at the factory. The goal is to ensure that the system interoperates in a first-class manner with Ubuntu, which may lead to eventual Ubuntu certification of the system(s).

The technical recommendations cover ACPI, WMI, SMM, hotkeys, video, and UEFI-specific details.

This document also covers the Ubuntu release cycle, including how Ubuntu incorporates the Linux kernel. Finally, it discusses some of the tools that have been developed by Canonical and the Ubuntu Community for the purpose of debugging and/or qualifying BIOS / UEFI implementations.

2. Ubuntu development

Ubuntu releases new versions on a fixed six-month schedule. Ubuntu uses a release numbering convention *YY.MM*, where *YY* is the last two digits of the year, and *MM* indicates the month of the release. At the time of writing, the current release of Ubuntu is 12.04, which was released in April 2012. Ubuntu releases also are given a two-word informal name, comprised of a fanciful or rare animal, and an adjective. The informal name for 12.04 is “Precise Pangolin”.

The current development version of Ubuntu is 12.10, codenamed “Quantal Quetzal”, and is scheduled for release in october 2012.

For more information on the Ubuntu release cycle, please refer to https://wiki.ubuntu.com/ReleaseSchedule

2.1. LTS releases

In addition to the regular six-month release cycle, Ubuntu also has a release cycle for Long Term Support (LTS) releases. Every fourth Ubuntu release (which is every 2 years) is a LTS release. These releases receive security updates for a longer period (5 years) than a normal Ubuntu release (18 months).

At the time of writing, the most recent LTS release is Ubuntu 12.04 LTS. The next development release will be 14.04.

Previously (up to and including the 10.04 release), LTS releases were supported for 3 years. This has been extended to 5 years from the 12.04 release onwards.

For more information about Ubuntu LTS releases, please refer to: https://wiki.ubuntu.com/LTS

2.2. Reporting Ubuntu bugs

Ubuntu uses a website called Launchpad, hosted by Canonical, to track Ubuntu bugs. The main Launchpad site is at http://launchpad.net/. As Ubuntu itself is fully open-source, all bugs reported to Launchpad are publicly visible.

If you wish to file a public bug against an Ubuntu release, including bugs in Ubuntu’s ACPI or WMI implementation, driver bugs, or generic kernel bugs, please refer to the following page on the Ubuntu Wiki, at https://help.ubuntu.com/community/ReportingBugs.

If your bugs relate to pre-production hardware, Canonical is able to establish a private “project”, within Launchpad, for your company’s use. Bugs filed within this project will not be publicly accessible by default. Please contact your representative at Canonical if you would like to establish such an account.

Finally, you can always contact Canonical's Professional & Engineering Services organization for help. Please see Appendix A, *Contacting Canonical*.

2.3. Kernel source code access

Being an open-source project, the source code for the Ubuntu kernel is available for public access. The authoritative repositories for the source code are hosted on http://kernel.ubuntu.com/git/, using the git version control system.

The kernel for each release of Ubuntu is kept in a separate git repository, and will require use of the `git` application to access it. For example, to download the kernel source code used in the 12.04 release of Ubuntu (codename "precise"), run the following:

```
git clone git://kernel.ubuntu.com/ubuntu/ubuntu-precise.git
```

For access to the kernel sources used in other releases, replace `precise` with the codename for the appropriate release. For help using the `git` software, consult the git documentation at http://git-scm.com/documentation.

2.4. Where to find more information

- http://www.ubuntu.com/
- https://wiki.ubuntu.com/
- https://odm.ubuntu.com/

3. Firmware Test Suite

Canonical has developed the Firmware Test Suite (FWTS) to check BIOS / UEFI firmware for implementation bugs and divergences from relevant specifications. FTWS is open source software, originally based on Intel's Linux-ready Firmware Developer Kit. While development on Intel's Test code ceased on October 2007, FWTS development continues to refine and expand the test coverage.

FWTS is a command-line tool, to be run from with Ubuntu, which performs a series of tests against the currently installed BIOS and/or UEFI firmware. It offers a rich set of arguments which allow users to run individual ACPI tests, specify the number of cycles certain tests to be run, etc.

The test is hosted in two Launchpad Personal Package Archives (PPAs) belonging to the firmware-test-team on Launchpad (at https://launchpad.net/~firmware-testing-team). There are two PPAs: development and stable. To install the stable version:

```
sudo apt-add-repository ppa:firmware-testing-team/ppa-fwts-stable
sudo apt-get update
sudo apt-get install fwts
```

For more information on how to install Ubuntu packages from Launchpad PPAs, please see: https://help.launchpad.net/PPAQuickStart.

For more information on how to run/use FWTS, please refer to https://wiki.ubuntu.com/Kernel/Reference/fwts.

3.1. Reporting fwts bugs

If you discover problems with FWTS, bugs should be reported against the package in Ubuntu, at https://bugs.launchpad.net/ubuntu/+source/fwts.

3.2. Firmware Test Suite Live image

FWTS Live is a bootable USB image that will automatically boot and execute tests provided by Firmware Test Suite. The FWTS Live image is available for both 32 and 64 bit architectures and is capable of booting both legacy BIOS implementations as well as native UEFI (64 bit only). The test results are stored on the USB device and can be analysed on the fly or later on another computer.

For more information on the FWTS Live image, including installation information and a demo, please visit the FWTS Live home page at https://wiki.ubuntu.com/HardwareEnablementTeam/Documentation/FirmwareTestSuiteLive.

3.3. Reporting fwts-live bugs and results

If you discover problems with the FWTS Live image or have questions about your test results, bugs should be reported against the fwts-live project in Launchpad, at https://bugs.launchpad.net/fwts-live/+filebug.

4. ACPI

The Advanced Configuration and Power Interface (ACPI) specification provides an open-standard for configuration and power management on consumer computing devices, such as desktops, laptops, and all-in-ones. Table 1, “Supported ACPI versions” shows the versions supported by each Ubuntu release.

Table 1. Supported ACPI versions

Ubuntu release	Kernel version	ACPICA version	ACPI version
Precise (12.04)	3.2	20110623	4.0
Oneiric (11.10)	3.0	20110413	4.0
Natty (11.04)	2.6.38	20110112	4.0
Maverick (10.10)	2.6.35	20100428	4.0
Lucid (10.04 LTS)	2.6.32	20090903	4.0

4.1. ACPI 4.0 functionality

ACPI tables contain a very rich range of configuration data and some tables even contain executable ACPI Machine Language (AML) code that can implement machine specific custom features in a high level operating system neutral way. Not all are mandatory and hence there are different levels of implementation to handle the various tables.

Table 2. ACPI tables

ACPI Table	ACPI 4.0 §	Implemented	Notes
BERT	17.3.1	N	
BOOT	5.2.6	N	
CPEP	5.2.18	N	
Y: Yes, implemented; N: Not implemented, P: Partially implemented			

ACPI Table	ACPI 4.0 §	Implemented	Notes
DBGP	5.2.6	N	See Microsoft Debug Port Specification
DMAR	5.2.6	N	
DSDT	5.2.11	P	Some methods not fully implemented
ECDT	5.2.15	Y	
EINJ	17.5.1	Y	Ubuntu Maverick 10.10++
ERST	17.5	Y	Ubuntu Maverick 10.10++
ETDT	5.2.6	N	Superseded by HPET, now considered obsolete
FACS	5.2.10	Y	
FADT	5.2.9	Y	
HEST	17.3.2	N	
HPET	5.2.6	Y	
IBFT	5.2.6	N	
IVRS	5.2.6	Y	AMD specific, Ubuntu Lucid 10.04 LTS++
MADT	5.2.12	Y	
MCFG	5.2.6	Y	
MCHI	5.2.6	N	
MSCT	5.2.19	N	
OEMx	5.2.6	N	OEM specific tables
PSDT	5.2.11.3	Y	Treated like SSDT, PSDT now deprecated
RSDT	5.2.7	Y	
SBST	5.2.14	N	
SLIT	5.2.17	Y	
SPCR	5.2.6	N	
SPMI	5.2.6	N	
SRAT	5.2.16	Y	
SSDT	5.2.11.2	P	Some methods not fully implemented
TCPA	5.2.6	N	
UEFI	5.2.6	Y	Ubuntu Maverick 10.10++. Limited functionality
WAET	5.2.6	N	
WDRT	5.2.6	N	
XSDT	5.2.8	Y	
Y: Yes, implemented; N: Not implemented, P: Partially implemented			

4.2. Predefined ACPI names handled in Ubuntu

The DSDT and SSDT contain byte code that can implement a range of control methods and data objects as described in section 5.6.7 of version 4.0 of the ACPI specification [ACPI 4.0]. The names of these control methods and objects are “predefined ACPI names” and must be implemented in a manner that conforms to the ACPI specification.

The Ubuntu Linux ACPI driver can use these control methods and data objects in a variety of ways. Control methods are executable ACPI Machine Language code which gets executed by an AML interpreter. For example, the Ubuntu Linux ACPI driver may want to determine the current brightness level, by evaluating (executing) the `_BQC` control method, which returns a value back to the driver.

Less used are control methods that the ACPI firmware can call and the ACPI driver returns a result in response. An example of this the `_OSI` method which returns true or false depending on the argument passed to it.

The Ubuntu Linux ACPI driver can also evaluate ACPI data objects - an evaluated data object returns ACPI objects back to the Ubuntu Linux ACPI driver. For example, predefined battery information is returned as a package of integers and strings when the `_BIX` object is evaluated.

The firmware test suite can sanity check control methods and data objects using the “method” test. This will load the ACPI tables into the Intel ACPICA execution engine and evaluate a sample of frequently used methods and data objects. The test type checks the return data, sanity checks fixed return values (such as in the `_BIX` and `_BIF` methods) and also checks to see if any mutexes are left in an incorrect locked state. To run this test use:

```
sudo fwts method
```

The “method” test will check over 90 of the most commonly used methods and objects.

The Ubuntu Linux ACPI driver uses just subset of all the available control methods and data objects. Firmware may implement a full and complete set of these, however, a subset are just evaluated and used at run time. Conversely, the firmware may implement a small subset - the level of implementation is a choice left to the vendor. For a full feature rich implementation we recommend implementing as much functionality as is supported by Ubuntu.

The Linux kernel handles the predefined ACPI names in three ways:

4.2.1. Fully supported ACPI names

The Ubuntu Linux ACPI driver can evaluate the following names (methods and data objects) for specific operating system functionality.

Table 3. Fully supported ACPI names

Method/Object	**ACPI 4.0 §**	**Base version**
`_ADR`	6.1.1, B.6.1, 18.1.8	Dapper 6.06 LTS
`_BBN`	6.5.5	Jaunty 9.04
`_BCL`	B.6.2	Dapper 6.06 LTS
`_BCM`	B.6.3	Dapper 6.06 LTS
`_BFS`	7.3.1	Dapper 6.06 LTS
`_BIF`	10.2.2.1	Dapper 6.06 LTS
`_BIX`	10.2.2.2	Maverick 10.10
`_BQC`	B.6.4	Dapper 6.06 LTS
`_BST`	10.2.2.6	Dapper 6.06 LTS
`_BTP`	10.2.2.7	Dapper 6.06 LTS
`_CID`	6.1.2	Karmic 9.10
`_CRS`	6.2.2	Dapper 6.06 LTS
`_CRT`	11.4.4	Dapper 6.06 LTS
`_CST`	8.4.2.1	Dapper 6.06 LTS
`_DCK`	6.5.2	Hardy 8.04 LTS
`_DCS`	B.6.6	Dapper 6.06 LTS
`_DDC`	B.6.5	Dapper 6.06 LTS
`_DGS`	6.2.3	Dapper 6.06 LTS

Method/Object	ACPI 4.0 §	Base version
_DIS	6.2.3	Dapper 6.06 LTS
_DOD	B.4.2	Dapper 6.06 LTS
_DOS	B.4.1	Dapper 6.06 LTS
_DSS	B.6.8	Dapper 6.06 LTS
_DSW	7.2.1	Jaunty 9.04
_E*xx*	5.6.4.1	Dapper 6.06 LTS
_EC	1.12	Dapper 6.06 LTS
_EJD	6.3.2	Dapper 6.06 LTS
_EJ*x*	6.3.3	Hardy 8.04 LTS
_GHL	10.4.7	Lucid 10.04 LTS
_GL	5.7.1	Dapper 6.06 LTS
_GLK	6.5.7	Dapper 6.06 LTS
_GPD	B.4.4	Dapper 6.06 LTS
_GPE	5.3.1, 12.11	Dapper 6.06 LTS
_GTF	9.8.1.1	Dapper 6.06 LTS
_GTM	9.8.2.1.1	Dapper 6.06 LTS
_GTS	7.3.3	Dapper 6.06 LTS
_HOT	11.4.6	Dapper 6.06 LTS
_INI	6.5.1	Dapper 6.06 LTS
_IRC	7.2.13	Dapper 6.06 LTS
_L*xx*	5.6.4.1	Dapper 6.06 LTS
_LCK	6.3.4	Dapper 6.06 LTS
_LID	9.4.1	Dapper 6.06 LTS
_MAT	6.2.9	Hardy 8.04 LTS
_OFF	7.1.2	Dapper 6.06 LTS
_ON	7.1.3	Dapper 6.06 LTS
_OSC	6.2.10	Karmic 9.10
_OSI	5.7.2	Dapper 6.06 LTS
_OST	6.3.5	Lucid 10.04 LTS
_PCT	8.4.4.1	Dapper 6.06 LTS
_PDC	8.4.1	Dapper 6.06 LTS
_PMC	10.4.1	Lucid 10.04 LTS
_PMD	10.4.8	Lucid 10.04 LTS
_PMM	10.4.3	Lucid 10.04 LTS
_PPC	8.4.4.3	Dapper 6.06 LTS
_PR	5.3.1	Dapper 6.06 LTS
_PR0	7.2.7	Dapper 6.06 LTS
_PRS	6.2.11	Karmic 9.10
_PRW	7.2.11	Dapper 6.06 LTS

Method/Object	ACPI 4.0 §	Base version
_PS0	7.2.2	Dapper 6.06 LTS
_PS3	7.2.5	Dapper 6.06 LTS
_PSC	7.2.6	Dapper 6.06 LTS
_PSD	8.4.4.5	Hardy 8.04 LTS
_PSL	11.4.8	Dapper 6.06 LTS
_PSR	10.3.1	Dapper 6.06 LTS
_PSS	8.4.4.2	Dapper 6.06 LTS
_PSV	11.4.9	Dapper 6.06 LTS
_PSW	7.2.12	Dapper 6.06 LTS
_PTC	8.4.3.1	Dapper 6.06 LTS
_PTP	10.4.2	Lucid 10.04 LTS
_PTS	7.3.2	Dapper 6.06 LTS
_PUR	8.5.11	Lucid 10.04 LTS
_PXM	6.2.13	Dapper 6.06 LTS
_Q*xx*	5.6.4.1	Dapper 6.06 LTS
_REG	6.5.4	Dapper 6.06 LTS
_REV	5.7.4	Dapper 6.06 LTS
_RMV	6.3.6	Dapper 6.06 LTS
_ROM	B.4.3	Dapper 6.06 LTS
_SB	5.3.1	Dapper 6.06 LTS
_SBS	10.1.3	Dapper 6.06 LTS
_SCP	11.4.11	Dapper 6.06 LTS
_SDD	9.8.3.3.1	Hardy 8.04 LTS
_SEG	6.5.6	Dapper 6.06 LTS
_SHL	10.4.5	Lucid 10.04 LTS
_SPD	B.4.5	Dapper 6.06 LTS
_SRS	6.2.15	Dapper 6.06 LTS
_SST	9.1.1	Dapper 6.06 LTS
_STA	6.3.7, 7.1.4	Dapper 6.06 LTS
_STM	9.8.2.1.2	Hardy 8.04 LTS
_SUN	6.1.8	Jaunty 9.04
_T_*x*	18.2.1.1	Dapper 6.06 LTS
_TC1	11.4.12	Dapper 6.06 LTS
_TC2	11.4.13	Dapper 6.06 LTS
_TMP	11.4.14	Dapper 6.06 LTS
_TPC	8.4.3.3	Hardy 8.04 LTS
_TSD	8.4.3.4	Hardy 8.04 LTS
_TSP	11.4.17	Dapper 6.06 LTS
_TSS	8.4.3.2	Dapper 6.06 LTS

Method/Object	ACPI 4.0 §	Base version
_TZ	5.3.1	Dapper 6.06 LTS
_TZD	11.4.19	Dapper 6.06 LTS
_TZM	11.4.20	Karmic 9.10
_TZP	11.4.21	Dapper 6.06 LTS
_UID	6.1.9	Dapper 6.06 LTS
_VPO	B.4.6	Dapper 6.06 LTS
_WAK	7.3.7	Dapper 6.06 LTS
_W*xx*	5.6.4.2.2	Dapper 6.06 LTS

4.2.2. Checked ACPI names

These are ACPI methods or data objects that are not currently accessed in any way by the Ubuntu ACPI driver. However if they are referenced by calling methods the driver will sanity check return values when these are evaluated and issue an error message if they do not confirm to the ACPI specification.

Table 4. Checked ACPI names

Method/Object	ACPI 4.0 §	Base version
_AL*x*	11.4.2	Lucid 10.04 LTS
_ALC	9.2.4	Karmic 9.10
_ALI	9.2.2	Karmic 9.10
_ALN	18.1.8	Karmic 9.10
_ALP	9.2.6	Karmic 9.10
_ALR	9.2.5	Natty 11.04
_ALT	9.2.3	Karmic 9.10
_ART	11.4.3	Lucid 10.04 LTS
_BCT	10.2.2.9	Lucid 10.04 LTS
_BDN	6.5.3	Lucid 10.04 LTS
_BLT	9.1.3	Karmic 9.10
_BMA	10.2.2.4	Lucid 10.04 LTS
_BMC	10.2.2.11	Karmic 9.10
_BMD	10.2.2.10	Karmic 9.10
_BMS	10.2.2.5	Lucid 10.04 LTS
_BTM	10.2.2.8	Karmic 9.10
_CBA	n/a	Karmic 9.10
_CDM	6.2.1	Lucid 10.04 LTS
_CSD	8.4.2.2	Karmic 9.10
_DDN	6.1.3	Karmic 9.10
_DMA	6.2.4	Karmic 9.10
_DSM	9.14.1	Karmic 9.10
_DTI	11.4.5	Lucid 10.04 LTS
_EDL	6.3.1	Karmic 9.10

Method/Object	ACPI 4.0 §	Base version
_FDE	9.9.1	Maverick 10.10
_FDI	9.9.2	Karmic 9.10
_FDM	9.9.3	Karmic 9.10
_FIF	11.3.1.1	Lucid 10.04 LTS
_FIX	6.2.5	Karmic 9.10
_FPS	11.3.1.2	Lucid 10.04 LTS
_FSL	11.3.1.3	Lucid 10.04 LTS
_FST	11.3.1.4	Lucid 10.04 LTS
_GRA	18.1.8	Karmic 9.10
_GSB	6.2.6	Karmic 9.10
_HE	18.1.8	Karmic 9.10
_HPP	6.2.7	Karmic 9.10
_HPX	6.2.8	Karmic 9.10
_IFT	n/a	Karmic 9.10
_INT	18.1.8	Karmic 9.10
_LEN	18.1.8	Karmic 9.10
_LL	18.1.8	Karmic 9.10
_MBM	8.12.2.1	Lucid 10.04 LTS
_MLS	6.1.5	Karmic 9.10
_MSG	9.1.2	Karmic 9.10
_MSM	9.12.2.2	Lucid 10.04 LTS
_NTT	11.4.7	Lucid 10.04 LTS
_PAI	10.3.2	Lucid 10.04 LTS
_PCL	10.3.2	Karmic 9.10
_PDL	8.4.4.6	Lucid 10.04 LTS
_PIC	5.8.1	Karmic 9.10
_PIF	10.3.3	Lucid 10.04 LTS
_PLD	6.1.6	Karmic 9.10
_PPE	8.4.5	Karmic 9.10
_PR1	7.2.8	Karmic 9.10
_PR2	7.2.9	Karmic 9.10
_PR3	7.2.10	Karmic 9.10
_PRL	10.3.4	Lucid 10.04 LTS
_PRT	6.1.12	Karmic 9.10
_PS1	7.2.3	Karmic 9.10
_PS2	7.2.4	Karmic 9.10
_RT	18.1.8	Karmic 9.10
_RTV	11.4.10	Karmic 9.10
_S0	7.3.4.1	Dapper 6.06 LTS

Method/Object	ACPI 4.0 §	Base version
_S1	7.3.4.2	Dapper 6.06 LTS
_S2	7.3.4.2	Dapper 6.06 LTS
_S3	7.3.4.4	Dapper 6.06 LTS
_S4	7.3.4.5	Dapper 6.06 LTS
_S5	7.3.4.6	Dapper 6.06 LTS
_S1D	7.2.14	Dapper 6.06 LTS
_S2D	7.2.15	Dapper 6.06 LTS
_S3D	7.2.16	Dapper 6.06 LTS
_S4D	7.2.17	Dapper 6.06 LTS
_S0W	7.2.18	Karmic 9.10
_S1W	7.2.19	Karmic 9.10
_S2W	7.2.20	Karmic 9.10
_S3W	7.2.21	Karmic 9.10
_S4W	7.2.22	Karmic 9.10
_SI	5.3.1	Dapper 6.06 LTS
_SLI	6.2.14	Karmic 9.10
_SRV	n/a	Karmic 9.10
_STP	9.18.2	Lucid 10.04 LTS
_STR	6.1.7	Dapper 6.06 LTS
_STV	9.18.3	Lucid 10.04 LTS
_SWS	7.3.5	Karmic 9.10
_TIV	9.18.4	Lucid 10.04 LTS
_TIP	9.18.5	Lucid 10.04 LTS
_TPT	11.4.15	Karmic 9.10
_TRT	11.4.16	Karmic 9.10
_TTS	7.3.6	Jaunty 9.04
_UPC	9.13	Karmic 9.10
_UPD	9.16.1	Karmic 9.10
_UPP	9.16.2	Karmic 9.10

4.2.3. Unsupported ACPI names

The Ubuntu ACPI driver does not reference or sanity check the following methods or data objects in any way.

Table 5. Unsupported ACPI names

Method/Object	ACPI 4.0 §
_AC*x*	11.4
_ASI	18.1.8
_ASZ	18.1.8
_BAS	18.1.8

Method/Object	ACPI 4.0 §
_BM_	18.1.18
_DEC	18.1.18
_MAF	18.1.8
_MAX	18.1.8
_MEM	18.1.8
_MIF	18.1.8
_MIN	18.1.8
_RW_	18.1.8
_TDL	8.4.3.5
_TRA	18.1.8
_TRS	18.1.8
_TSF	18.1.8
_TTP	18.1.8
_TYP	18.1.8
_MTP	18.1.8
_RBO	18.1.8
_RBW	18.1.8
_RNG	18.1.8
_SHR	18.1.8
_SIZ	18.1.8

4.3. Supported ACPI device IDs

Certain integrated devices require support for some device-specific ACPI controls. Section 5.6.6 of version 4.0a of the ACPI specification [ACPI 4.0] lists these devices and their corresponding Plug-and-Play (PNP) IDs. The Ubuntu kernel supports a variety of these devices as described below. If the DSDT contains PNP _HIDs that are not supported, then expect Ubuntu to effectively ignore the description given in the DSDT. However, this doesn't necessarily mean the device is not supported, it just means that Ubuntu will ignore the device description/configuration.

Table 6. Checked ACPI names

PNP _HID	Description	Base support version
PNP0C09	Embedded Controller	Dapper 6.06 LTS
PNP0C0A	Contol Method Battery	Dapper 6.06 LTS
PNP0C0B	Fan	Dapper 6.06 LTS
ACPI_FPB	Power Button	Dapper 6.06 LTS
ACPI_FSB	Sleep Button	Dapper 6.06 LTS
PNP0C0C	Power Button	Dapper 6.06 LTS
PNP0C0D	Lid Device	Dapper 6.06 LTS
PNP0C0E	Sleep Button	Dapper 6.06 LTS
PNP0C0F	PCI interrupt link device	Dapper 6.06 LTS
PNP0C14	WMI	Jaunty 9.04
PNP0C80	Memory Device	No

PNP _HID	Description	Base support version
ACPI0001	SMBus 1.0 Host Controller	Dapper 6.06 LTS
ACPI0002	Smart Battery Subsystem	Dapper 6.06 LTS
ACPI0003	Power Resource Device	Dapper 6.06 LTS
ACPI0004	Modular Device	Dapper 6.06 LTS
ACPI0005	SMBus 2.0 Host Controller	Hardy 8.04 LTS
ACPI0006	GPE Block Device	No
ACPI0007	Processor Device	Karmic 9.10
ACPI0008	Ambient Light Sensor Device	No
ACPI0009	I/OxAPIC Device	No
ACPI000A	I/O APIC Device	No
ACPI000B	I/O SAPIC Device	No
ACPI000C	Processor Aggregator Device	Lucid 10.04 LTS
ACPI000D	Power Meter Device	Lucid 10.04 LTS
ACPI000E	Wake Alarm Device	No

4.4. 32- and 64-bit addresses (Generic Address Structure)

Some tables, such as the FADT, contain required 32-bit addresses and also a 64-bit extended address field in the form of a Generic Address Structure (GAS). For example, the 32-bit PM1a_EVT_BLK is superseded by the 64-bit X_PM1a_EVT_BLK.

The strict interpretation from version 4.0 of the ACPI specification is that the 64-bit address supersedes the 32-bit address, if the 64-bit address is non-zero. The assumption is that a non-zero 64-bit address should point to the same object as the 32-bit address. However, some firmware has been known to set different addresses (referencing different data), which is ambiguous.

Ubuntu will use 64-bit addresses if they are present and non-zero, 32-bit addresses otherwise.

4.5. ACPI table checksum

All ACPI tables contain an 8 bit checksum which should be set so that the 8 bit sum of data in each table is zero. For the Root System Description Pointer (RSDP) the checksum covers the first 20 bytes of the structure. For all other system description tables, a header contains a checksum field, which should again be set so that the 8 bit sum of all the data in each the table comes to zero.

Invalid checksums indicate a potentially corrupt table, and the kernel will complain with a warning. However, tables will still be loaded. We recommended that checksums are present and correct, otherwise it is impossible to differentiate between a correct table with a bad checksum and a table that contains errors because of firmware data corruption.

The firmware test suite contains an ACPI table checksum test which can be run as follows:

```
sudo fwts checksum
```

For more details, consult sections 5.2.5.3 and 5.2.6 of the ACPI 4.0 Specification [ACPI 4.0]

4.6. ACPI Machine Language (AML)

ACPI Machine Language (AML) is the byte code instruction found in the ACPI DSDT and SSDT tables. The Linux kernel fully supports the AML bytecode as specified in section 19 of version 4.0 of the

ACPI specification [ACPI 4.0]. ACPI control methods are compiled into AML and this code is executed inside the kernel context by the Linux ACPI driver. Generally the AML is compiled using either the Microsoft AML or Intel compilers and these can produce different output based on the same source code. Ubuntu uses the Intel AML compiler.

- We recommended that the code should not contain infinite loops. All loops should contain a loop counter or a timeout to break out of a loop at some point. Infinite loops can lock up the ACPI driver's thread of execution and lead to excessive CPU load and potential lockups on serialized code paths.
- Deep recursion of methods should be avoided. The Linux ACPI driver will detect and halt recursions deeper than 255 levels, this leads to undefined execution behaviour.
- Methods should always return the expected return types according to the ACPI specification for methods that are defined by the ACPI specification. If the method is not defined by the ACPI specification the method should always return the type expected by the caller. We have observed that sometimes methods have multiple return control paths and some of these neglect to return the expected return types on all the return paths.
- Methods should never return packages containing zero elements.
- Package lists and tables returned by methods should always be the correct expected size.
- Methods should always be called with the correct number of arguments with the correct type.
- Field accesses outside a defined buffer or memory region are illegal and ignored. This leads to undefined behaviour and hence unexpected results.
- No illegal AML op-codes are allowed. This will result in undefined behaviour at run time.

The Microsoft compiler seems to be less strict than the Intel AML compiler, and so we recommend that the source is compiled using the Intel compiler to check for any illegal code. The Intel compiler also contains some semantic checking and can even catch some subtle bugs such as mutex Acquires with missing timeout failures.

4.7. `_OSI`(Linux)

Section 5.7.2 of version 4.0a of the ACPI specification [ACPI 4.0] describes the `_OSI` (Operating System Interfaces) object. This object provides the firmware with the ability to query the operating system to determine the set of ACPI related interfaces, behaviors, or features that the operating system supports. For example, Windows Vista requires the latest ACPI backlight functionality (see Appendix B of the ACPI specification) and the firmware can use `_OSI` to detect this version of the operating system to enable this extra functionality.

Linux attempts to be compatible with the latest version of Windows, and will always return true to `_OSI` with all known Windows version strings. The intention is to make it impossible for the firmware to tell if the machine is running Linux. The implementation of `_OSI` can be found in the kernel sources in `drivers/acpi/osl.c`

`_OSI` has been used to detect an operating system version to try to work around bugs in the operating system. **Trying to work around a Linux bug by detecting an operating system version using _OSI should be avoided at all costs**. These workarounds can break with new versions of the kernel. The best approach is to engage with Linux developers and fix the problem in the kernel.

4.7.1. `_OSI` in detail

The `_OSI` method has one argument and one return value. The argument is an OS vendor defined string representing a set of OS interfaces and behaviours or an ACPI defined string representing an operating system.

Ubuntu Linux will return `0xffffffff` (i.e. feature is supported) for the following arguments to `_OSI`:

Table 7. `_OSI` support

_OSI argument	Windows version	Supported in Ubuntu
Windows 2000	Windows 2000	Pre-Dapper 6.06 LTS
Windows 2001	Windows XP	Pre-Dapper 6.06 LTS
Windows 2001	Windows XP SP1	Pre-Dapper 6.06 LTS
Windows 2001.1	Windows Server 2003	Pre-Dapper 6.06 LTS
Windows 2001 SP2	Windows XP SP2	Pre-Dapper 6.06 LTS
Windows 2001.1 SP1	Windows Server 2003 SP1	Hardy 8.04 LTS
Windows 2006	Windows Vista	Hardy 8.04 LTS
Windows 2006.1	Windows Server 2008	Lucid 10.04 LTS
Windows 2006 SP1	Windows Vista SP1	Lucid 10.04 LTS
Windows 2006 SP2	Windows Vista SP2	Natty 11.04
Windows 2009	Windows 7 and Server 2008 R2	Lucid 10.04 LTS

`_OSI` will always return 0 (feature not supported) for "*Linux*". Ubuntu Linux will return true (`0xffffffff`) to the most recent Windows `_OSI` string. The "*Linux*" `_OSI` argument is meaningless and should never be expected to work or do anything useful.

The DSDT and SSDT tables contain ACPI Machine Language (AML) code that has access to a wide range of I/O ports, memory regions and memory mapped I/O. The ACPI driver will ban the AML code from accessing certain port I/O operations at run time depending on which which OS behaviour compatibility string is passed to `_OSI`. See Table 8, “Banned I/O ports” for the list of I/O ports that are always banned to AML byte code.

Table 8. Banned I/O ports

Port range	Description
0x0020-0x0021	PIC0: Programmable Interrupt Controller (8259_a)
0x00a0-0x00a1	PIC1: Cascaded PIC
0x04d0-0x04d1	ELCR: PIC edge/level registers

However, for Windows XP and higher, different port ranges are banned to AML byte code. Table 9, “Windows XP+ banned I/O ports” lists these ranges.

Table 9. Windows XP+ banned I/O ports

Port range	Description
0x0000-0x000F	DMA: DMA controller
0x0040-0x0043	PIT1: System Timer 1
0x0048-0x004b	PIT2: System Timer 2 failsafe
0x0070-0x0071	RTC: Real-time clock
0x0074-0x0076	CMOS: Extended CMOS
0x0081-0x0083	DMA1: DMA 1 page registers
0x0087-0x0087	DMA1L: DMA 1 Ch 0 low page
0x0089-0x008b	DMA2: DMA 2 page registers
0x008f-0x008f	DMA2L: DMA 2 low page refresh

Port range	Description
0x0090-0x0091	ARBC: Arbitration control
0x0093-0x0094	SETUP: Reserved system board setup
0x0096-0x0097	POS: POS channel select
0x00c0-0x00df	IDMA: ISA DMA
0x0cf8-0x0cff	PCI: PCI configuration space

So avoid accessing these ports by AML byte code - it will not work and it results in an kernel error message ("Denied AML access to port"), and the AML will execute incorrectly. This could lead to undefined behaviour since port reads and writes will not be executed. Failed port reads can potentially return uninitialised random data found on the stack or heap - an error will be flagged and usually the port reading caller will skip or abort execution.

To access registers in devices using these I/O spaces, one needs to declare devices in ASL.

The following is an example of PCI(e) device written in ASL. By declaring this device in the PCI bridge or PCIe Root Port, the BIOS is able to read the Vendor ID (VID) and Device ID (DID) in PCI configuration space without directly accessing I/O ports `0xCF8` and `0xCFC`. The low-level hardware accesses will be handled by the Linux kernel.

```
Device (PDEV) {   // Device 0x01, Function 0x02
   Name (_ADR, 0x00010002)
   OperationRegion (CNFG, PCI_Config, 0x0, 0x100)
   Field (CNFG, DWordAcc, NoLock, Preserve) {
         VID,   16,
         DID,   16,
   }
}
```

Similarly, the BIOS can read from or write to CMOS registers in a RTC device without directly accessing I/O Port 0x70 and 0x71. The example below demonstrates how the BIOS can declares Seconds, Minutes and Hours registers in a RTC device. More details for different RTC devices can be found in section 9.15 of ACPI specification [ACPI 4.0].

```
Device (RTC) {   // PC/AT-compatible RTC Device

   Name (_HID, EisaId ("PNP0B00"))
   Name (_CRS, ResourceTemplate () {
      IO (Decode16,
         0x0070,
         0x0070,
         0x01,
         0x08,
      )
      IRQNoFlags ()
         {8}
   })

   OperationRegion(CMS1, SystemCMOS, 0, 0x40)
   Field(CMS1, ByteAcc, NoLock, Preserve) {
      SECD, 8,
      , 8,
      MINT, 8,
      , 8,
      HOUR, 8
```

```
        }
    }
```

Port access violations caused by AML code (which occur very rarely) can be detected by the firmware test suite using the `klog` test, run FWTS as follows:

```
sudo fwts klog
```

4.8. Battery

Battery information is of value to Linux on mobile platforms such as laptops and netbooks. The ACPI specification describes two interfaces, “Smart Battery” (Section 10.1) and “Control Method Batteries” (Section 10.2). Few systems implement “Smart Battery”, and although Linux provides a driver, it has not been tested on a wide range of machines. The preferred interface is “Control Method Batteries”, but Canonical can assist if your hardware or firmware requires Smart Battery support.

It is recommended that static battery information should be provided by either one of the `_BIF` (Battery Information) or `_BIX` (Battery Information Extended) objects. Linux can handle either object, however `_BIF` is deprecated in ACPI 4.0 so `_BIX` is the preferred choice. Sections 10.2.2.1 and 10.2.2.2 of ACPI specification 4.0a describe these objects in detail. Please ensure that the objects are packages that comply with the specification in terms of typing and ranges. Most specifically:

1. Power Unit is 0x00000000 (mWh) or 0x00000001 (mAh).
2. Design Capacity is between 0x00000000 and 0x7fffffff if known and 0xffffffff if unknown.
3. Last Full Capacity is between 0x00000000 and 0x7fffffff if known and 0xffffffff if unknown.
4. Battery Technology is 0x00000000 or 0x00000001.
5. Design Voltage is either 0x00000000-0x7fffffff if known and 0xffffffff if unknown.
6. Design capacity of Warning is 0x00000000-0x7fffffff
7. Cycle Count is either 0x00000000-0xfffffffe if known and 0xffffffff if unknown.
8. Measurement Accuracy is in thousands of a percent, 0x00000000-0x000186a0
9. Min and Max Sampling Times are 0xffffffff if unavailable.
10. Model Number, Serial Number, Battery Type and OEM Information strings are defined. Null entries make it practically impossible to identify battery hardware, so please ensure they are defined correctly.

The Linux kernel will report these values with minimal filtering of any incorrect data. Incorrect or out of range values can potentially confuse the higher levels of power management.

Where as the `_BIX` and `_BIX` objects are generally static information, the `_BST` (Battery Status) control method is used to determine the current battery status. We expect this control method to be implemented for mobile platforms.

This method is described in section 10.2.2.6 of version 4.0a of the ACPI specification. This needs to return a package of all DWORD integers which contain values conforming to table 10-7 (_BST Return Package Values) in the specification.

It is essential that the Battery State Bits 0..2 contain the correct discharging/charging/critical state settings and Battery Present Rate. Note that bits 0 and bit 1 in are mutually exclusive.

Battery Remaining Capacity and Battery Present Voltage fields contain correct and reliable data in the ranges 0x00000000-0x7fffffff if known and 0xffffffff if unknown.

4.9. Mutexes

ASL provides mutex primitives via the `Acquire()` and `Release()` operators. Mutexes are used to provide synchronization around critical data to avoid race conditions. The executing thread is suspended until the mutex is released or a timeout occurs. The timeout value can be 0xffff (or greater) to indicate an indefinite wait, or a value less than 0xffff indicates a timeout in milliseconds.

The `Acquire` operator returns True if a timeout occurs and hence the mutex was not acquired. Thus for timeouts less than 0xffff it is required that mutex acquire failures are checked and the error condition is handled appropriately. We class any AML code that contains non-indefinite waits without checking for timeout failures as a critical bug - the AML code contains potentially hazardous race conditions and will result in undefined incorrect execution behaviour.

It is also important to balance each `Acquire()` with a mutex `Release()`. Sometimes methods contain multiple return paths and some of these do not release acquire mutexes. This causes subtle lock-ups during execution of the methods and we class these as critical bugs.

4.10. Thermal zones

Linux has a mature ACPI Thermal Zone driver that allows proactive system cooling policies as described by section 11 of the ACPI specification. Providing ACPI thermal zones in the firmware allows Linux to monitor and control cooling decisions based on CPU loading and thermal heuristics. ACPI Thermal Zones can be implemented as a complete system thermal zone, or a system can be partitioned into multiple thermal zones (e.g. per CPU, device, etc.) for finer control.

It is recommended to provide reasonable and sensible trip points and polling intervals within the limits provided by the ACPI specification. If the hardware cannot generate asynchronous notifications to detect temperature changes, then one is required to specify sensible polling intervals. Polling intervals should not be too short to overload the CPU and also not too infrequent as to miss critical thermal levels.

Ensure that critical trip points are correctly set (in degrees Kelvin) so that Linux can trigger graceful shutdowns.

It is our observation that ACPI Thermal Zones are not implemented in the firmware of the majority of mobile platforms and instead System Management Mode (SMM) seems to be the preferred mechanism for handling fan control and critical thermal trip points. While this solution may work, it does mean that non-maskable System Management Interrupts can preempt the kernel and hence affect real time performance. Therefore, if possible, use ACPI thermal zones in preference to SMM.

5. Windows Management Instrumentation (WMI)

Windows Management Instrumentation (WMI) is a complex set of proprietary extensions to the Windows Driver Model that provides an OS interface to allow instrumented components to provide information and notifications.

Typically we are interested in WMI if a laptop or netbook has implemented hotkey events using WMI. In this case, we need to write a driver or extend and existing driver to capture the appropriate WMI events and map these onto key events.

Our recommendation is to implement hot keys by generating scan codes that can be interpreted at the input layer without the need of a WMI driver. However, if hotkeys must be implemented using WMI, then we recommend:

1. Use existing WMI implementations, so that drivers do not need to be written or extended.

2. WMI interfaces are thoroughly documented. New WMI interfaces require a specification provided during the planning phase.

We require a description of the following:

1. The WMI GUIDs and Notifier IDs
2. A complete description of the data return on a notifier event (e.g. the data returned by the _WED control method).
3. A description of the expected return codes and the keys they map to. E.g. “code 0x0013 maps to the brightness down key.”
4. Where possible, the Managed Object Format (MOF) source before it is compiled into a WMI WQxx binary object. This will help us to understand the higher level semantics of the WMI GUIDs.

5.1. Common Errors

5.1.1. Incorrect GUID generation

It is important not to reuse GUIDs. Copying and pasting GUIDs from elsewhere (for example, from sample code) will cause GUID conflicts, which makes it impossible for a driver to distinguish WMI devices uniquely. This makes fixing bugs impossible, as the designs of two WMI devices are likely to be different.

It is also important not to generate a GUID by modifying existing GUIDs in any way. It is a common mis-understanding that modifying a GUID (eg. adding 1) can avoid conflict. This is not how GUIDs work; all GUIDs must be generated independently.

A number of tools are available for proper GUID generation:

- For Ubuntu, the `uuidgen` utility will generate GUIDs quickly and easily.
- For Windows, Microsoft's GuidGen tool can be used to generate GUIDs. This tool is shipped with MS Visual C++, but is also available at https://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17252.
- There are various web services available to generate GUIDs online, such as http://guid.us/.

6. System Management Mode (SMM)

System Management Mode (SMM) is a mechanism that allows the processor to temporarily jump into a high privilege mode and execute specialised (firmware) assist code.

To enter System Management Mode, a System Management Interrupt (SMI) is generated either by hardware signals on a pin on the processor, or by a software SMI triggered via a SMI port (eg, port `0xb2` on Intel platforms), or by a I/O write to a port that is configured to generate an SMI.

SMIs cannot be blocked or disabled and effectively suspend execution of the operating system while they are being handled. This can disrupt the operating system in several ways:

- From the operating system’s viewpoint, clock ticks are mysteriously lost
- SMIs can disrupt real time performance, due to unexpected latency injection
- SMM code can make assumptions on the way specific hardware is configured (such as APICs) and these may be incompatible with the way Ubuntu Linux handles the hardware

Therefore, we recommend that SMIs are used only where absolutely necessary, such as critical CPU temperature behaviour, where the functionality provided in the SMI handler must be available regardless of the state of the operating system. In situations where other hardware service facilities are feasible, we recommend using those over SMI-based implementations.

When they are used, we recommend that that SMIs take no longer than 150 microseconds to complete (ie., return control to the operating system). SMI latencies longer than 150 microseconds potentially risk operating system timeouts. Delays greater than 300 microseconds are considered inappropriate (will definitely cause operating system timeouts) and must be avoided.

Intel's BIOSBITS (BIOS Implementation Test Suite) (http://biosbits.org/) is a useful test suite that can detect long SMI latencies. We recommend that firmware passes the SMI test.

6.1. High Precision Event Timer (HPET)

The HPET can cause issues with suspend (S3) because Ubuntu Linux uses a tickless HZ timer, whereas Windows uses a periodic clock. We have observed that System Management Mode SCIs that jump into the BIOS can cause long hangs, possibly because of small delays based on 64 bit timers than may have 32 bit counter wrap-around, and the firmware does not take this into consideration. We recommend that BIOS vendors consider the ramifications of the tickless HZ timer that Ubuntu Linux uses when handling SMIs.

7. System Management BIOS (SMBIOS)

The System Management BIOS (SMBIOS) specification [SMBIOS 2.71] describes how systems and motherboard vendors structure management information in a standard way. This information describes the hardware and is intended to allow operating systems and applications to identify hardware without the need to probe system hardware which can be difficult, error prone and unreliable. Unfortunately, firmware can reach the market with poorly constructed SMBIOS information which makes it difficult or impossible to determine hardware configurations because of empty (null) fields or fields containing meaningless default values or text strings.

Therefore, we recommend that fields comply with the following rules:

- No empty or invalid fields
- Serial numbers must be defined and not left to a default such as 0123456789
- Asset tags must be defined and not left to a default such as 0123456789
- UUIDs need to be defined and not defaulted to `0A0A0A0A-0A0A-0A0A-0A0A-0A0A0A0A0A0A0A0A`
- No fields should contain defaults such as "To Be Filled By O.E.M."

7.1. Common Errors

7.1.1. System Enclosure or Chassis (Type 3)

SMBIOS defines attributes of mechanical enclosures in section 7.4 [SMBIOS 2.71]; specifically, the System Enclosure Indicator (offset `0x00`) specifies the chassis type. It is common that this field does not match the actual hardware. We recommend that the Type 3 data is programmed correctly.

In addition to SMBIOS, ACPI defines Preferred_PM_Profile in Fixed ACPI Description Table (FADT) [ACPI 4.0]. It is a very common error that the SMBIOS Type 3 data and ACPI Preferred_PM_Profile does not match. We recommend that these two attributes are consistent.

7.1.2. Portable Battery (Type 22)

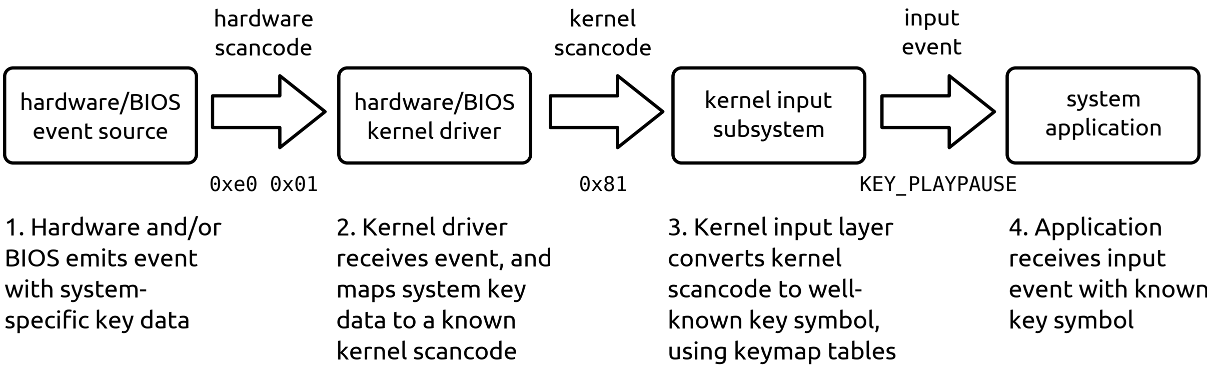
Both SMBIOS and ACPI specifications define structures for portable batteries. Because SMBIOS is static and ACPI is dynamic, it is not practical to match their attributes. However, we recommend always including SMBIOS Type 22 when systems support portable batteries, i.e. the ACPI battery is declared in AML, even if no battery is attached during BIOS POST.

8. Hotkeys

8.1. Hotkey mappings

Because of the wide range of hardware that the Ubuntu kernel supports, and the different applications that expect to receive input events, there is a translation between hardware event data and the input event codes that are generated from these hardware events. Figure 1, “Hardware-to-input-event mapping” shows the process used to map hardware events to events that applications can understand.

Figure 1. Hardware-to-input-event mapping



While the first stage of mapping is generally hardcoded in the driver, the second stage is configurable through a standard kernel interface. **Where possible, existing mappings should be re-used**. This is best achieved by using a standard set of hardware scancodes across multiple SKUs.

Tables of existing hotkey mappings can be found in Appendix B, *Existing hotkey mappings*.

The mappings are defined by udev rules, and can be found in the `/lib/udev/keymaps/` directory. The format is as follows:

`kernel-scancode keycode-symbolic-name #comments`

For example:

```
0x81 playpause # Play/Pause
0x82 stopcd # Stop
0x83 previoussong # Previous song
0x84 nextsong # Next song
```

The scancode in the file is in the encoded form used by Linux kernel. For example, the scancode `0xe0 0x01` is encoded as `0x81` and it is mapped to the symbolic name KEY_PLAYPAUSE. The available keycode symbolic names are listed in `/usr/include/linux/input.h`.

The relevant mapping file will generally be called `/lib/udev/keymaps/`*`vendor-name`*. The appropriate keymap is loaded at runtime, by the udev rules in `/lib/udev/rules.d/95-keymap.rules`. If no suitable file exists, Canonical can work with you to create a new one. However, **we highly recommend that existing mappings are re-used**. New mappings (for keys that are not already present in a suitable keymap) can be easily added to this file, but should be kept consistent across product lines.

8.2. Brightness controls

Brightness control hotkeys on mobile platforms are implemented in various ways on different platforms. The recommended method is to implement the ACPI brightness control methods, detailed in section B.6 of the ACPI specification [ACPI 4.0]. We also recommend that the BIOS notifies the Linux kernel as defined in section B.7 of the specification. The supported notification values are listed in Table 10, “Notification Values for Brightness Control”.

Table 10. Notification Values for Brightness Control

Notification Value	Description
0x86	Increase Brightness
0x87	Decrease Brightness

Brightness control hotkeys should not be an internal BIOS implementation that excludes operating system participation. Some BIOSes require specific `_OSI` levels before they enable ACPI brightness controls - Linux will always support ACPI brightness control support, so use this, no matter what `_OSI` reports.

8.3. WMI hotkeys

These should be avoided, as they usually require a custom WMI event mapper to translate WMI events into key scan codes. See Section 5, “Windows Management Instrumentation (WMI)” for more details.

8.4. Keyboard BIOS hotkeys / Embedded Controller hotkeys

Sometimes known as “Keyboard BIOS hotkeys”, or “Embedded Controller hotkeys”, this is a mechanism where the scancodes are sent directly via the keyboard controller. The Linux kernel directly receives keyboard codes on the keyboard input device, so a special hotkey driver is not required. However, the hotkey keyboard codes need to be remapped to meaningful key codes via udev keymaps.

We recommend that hotkeys implemented in this manner should re-use already-existing keymaps, keeping consistent with previous SKUs. Refer to Section 8.1, “Hotkey mappings” for the mapping process, and Appendix B, *Existing hotkey mappings* for details of existing mappings used by Ubuntu.

A well-behaved EC for this hotkey should send both key press and release scan codes.

8.5. Vendor-specific ACPI device HID hotkeys

A vendor specific device HID is an alternative mechanism for implementing hotkeys in ACPI. Typically a vendor specific device is implemented in the DSDT with some arbitrary device HID and hotkeys generate GPE edge or level triggered event(s) which cause notify events that need to be handled. A Linux platform specific driver needs to be written to handle this unique device.

Such hotkey implementations should be avoided. However, if it cannot be avoided then full details about the notify events and how they map onto hotkeys, the HID and any methods that need to be evaluated to operate this interface are required. Failure to disclose this information requires the engineer to reverse engineer the driver from the DSDT by observing notify events on key presses.

8.6. PNP device HID hotkeys

Table 11, “Hotkey PNP devices” describes the Hotkey PNP Devices supported by Ubuntu.

Table 11. Hotkey PNP devices

PNP device	Description
ACPI_FPB	Power Button
ACPI_FSB	Sleep Button
PNP0C0C	Power Button
PNP0C0D	Lid Device
PNP0C0E	Sleep Button

8.7. Video output hotkeys

Most laptops have a video out hotkey (generally Fn+F1 or Fn+F7) which causes the system to cycle through the following external monitor options when an external monitor is attached to one of the system’s video outputs (eg. HDMI, DP or VGA). This key cycles the video output mode through the following four states:

- Same image on both ("clone")
- Builtin Display only (LVDS)
- External Display only
- Extended Display (both displays active, desktop extended over both displays)

Newer laptops released with support for Windows 7 now sends a new keycode, “Mod4 + P” (Mod4 is the Windows key modifier).

Video mode switching is handled by the OS system application `gnome-settings-daemon`. This daemon responds to the keycodes and uses the `xrandr` application to affect the desired mode changes. This requires that the X graphics driver supports the xrandr protocol, version 1.2 or later.

8.7.1. Discrete NVIDIA graphic switching

At the time this document was written, the closed source NVIDIA driver does not support xrandr 1.2, so the standard Ubuntu utilites are unable to control the monitor configuration when this driver is in use.

Specifically, the driver relies on an ACPI event, `NVIF_NOTIFY_DISPLAY_DETECT` (value `0xcb`) to be generated by the BIOS on display reconfiguration. This event is enabled through the `NVIF` ACPI method.

8.8. RF killswitches

RF killswitches are hotkeys that disable/re-enable radio devices such as Wifi and Bluetooth. We recommend that these keys generate a standard vendor-specific key scancode that can be remapped to the wlan key via the udev keymapping. These key events can then be used to inform the driver to disable/re-enable wireless, via the Ubuntu kernel rfkill interface.

An example of the mapping for Dell laptops is the Fn+F2 key, which emits scancode `0xe0 0x08`. This scancode is mapped to the wlan key event via udev rules in `/lib/udev/keymaps/dell`.

Note

If a driver provided by an OEM/ODM/IHV does not support the standard kernel rfkill interface, and the hardware design of the hotkey directly controls power to the device, Canonical will not accept responsibility for bugs tied to the wireless hotkey.

8.9. Touchpad killswitches

Touchpad kill switches are hotkeys that enable/disable touchpads. Touchpads that use the underlying PS/2 mouse protocol typically implement the kill switch at the embedded controller firmware layer. When the kill switch disables the touchpad, no further PS/2 packets are emitted from the 8042 keyboard controller until the kill switch toggles it back on again.

We recommend that a touchpad toggle key event is generated, so that a notification can be displayed to the user. The key code is vendor-specific and requires remapping through udev keymaps.

For example, Dell laptops emit the scan code `0xe0 0x1e` or `0xe0 0x59` when the rfkill key is pressed. This maps to the F21 key event (touchpad toggle) via the udev rules in `/lib/udev/keymaps/dell`.

The touchpad toggle key event is processed by `gnome-settings-daemon`, which will trigger the on-screen notification. It also will track the state of the touchpad toggle, and persist this state using the GConf key `/desktop/gnome/peripherals/touchpad/touchpad_enabled`. This GConf key allows the system to keep track of the touchpad state across suspend, hibernate, or reboot.

Neither the Linux kernel, nor gnome-settings-daemon explicitly handle the actual enable/disable logic, nor the any associated LED which indicates the status of the touchpad. Both of these functions are the responsibility of the BIOS/Embedded Controller.

9. UEFI

Canonical is currently working on a base-level compatibility between Ubuntu and UEFI firmware. We are expecting that most OEM machines will ship with a firmware that complies with version 2.3.1 of the UEFI standard.

9.1. Legacy BIOS compatibility

Ubuntu's UEFI compatibility has mainly been tested with legacy BIOS compatibility mode (known as a Compatibility Support Module, or “CSM”) enabled. However, Ubuntu is known to work with non-CSM firmware: Ubuntu 12.04 has been tested on EDK Build Version 10 in native UEFI mode (no-CSM), on Intel SDP hardware.

In the future, we expect for CSM to be disabled. Provided that native-UEFI drivers are working (in particular, those providing EFI_GRAPHICS_OUTPUT_PROTOCOL support), UEFI firmware with configurable CSM mode should be configured to disable CSM.

9.2. UEFI boot services

Table 12, “Boot services used” lists the boot services that may be referenced by the Ubuntu bootloader. Any optional services that are not implemented by the firmware must return `EFI_UNSUPPORTED`.

Table 12. Boot services used

Service	UEFI §	Compliance
EFI_INSTALL_CONFIGURATION_TABLE	6.5	Required

Service	UEFI §	Compliance
EFI_LOCATE_PROTOCOL	6.3	Required
EFI_LOCATE_HANDLE	6.3	Required
EFI_OPEN_HANDLE	6.3	Required
EFI_STALL	6.5	Required
EFI_EXIT	6.4	Required
EFI_SET_WATCHDOG_TIMER	6.5	Optional, watchdog is disabled
EFI_ALLOCATE_PAGES	6.2	Required
EFI_FREE_PAGES	6.2	Required
EFI_GET_MEMORY_MAP	6.2	Required
EFI_EXIT_BOOT_SERVICES	6.4	Required
EFI_UNLOAD_IMAGE	6.4	Required
EFI_START_IMAGE	6.4	Required
EFI_LOAD_IMAGE	6.4	Required

Table 13, “Boot protocols used” lists the boot protocols that may be referenced by the Ubuntu bootloader.

Table 13. Boot protocols used

Protocol	UEFI §	Compliance
`EFI_DEVICE_PATH_PROTOCOL`	9.1	Required
`EFI_GRAPHICS_OUTPUT_PROTOCOL`	11.9	Required
`EFI_DISK_IO_PROTOCOL`	12.7	Required
`EFI_BLOCK_IO_PROTOCOL`	12.8	Required
`EFI_GRAPHICS_OUTPUT_PROTOCOL`	11.9	Required
`EFI_SIMPLE_NETWORK_PROTOCOL`	21.1	Optional, required for netboot
`EFI_PXE_BASE_CODE_PROTOCOL`	21.3	Optional, required for netboot

9.3. UEFI runtime services

Table 14, “Runtime services used” lists the runtime services that the Ubuntu bootloader and kernel may use, after `ExitBootServices()` has been invoked. Any optional services that are not implemented by the firmware must return `EFI_UNSUPPORTED`.

Table 14. Runtime services used

Service	UEFI §	Compliance
`EFI_GET_TIME`	7.3	Required
`EFI_SET_TIME`	7.3	Required
`EFI_GET_WAKEUP_TIME`	7.3	Optional, required for RTC wakeup
`EFI_SET_WAKEUP_TIME`	7.3	Optional, required for RTC wakeup
`EFI_SET_VIRTUAL_ADDRESS_MAP`	7.4	Required
`EFI_GET_VARIABLE`	7.2	Required
`EFI_GET_NEXT_VARIABLE_NAME`	7.2	Required
`EFI_RESET_SYSTEM`	7.5.1	Required

Due to issues with existing UEFI implementations, the memory used for EFI boot services is not reused by the operating system until after `EFI_SET_VIRTUAL_ADDRESS_MAP` has been invoked. However, runtime services should *not* depend on the presence of boot-services memory for any other runtime services.

9.4. UEFI configuration tables

The Ubuntu kernel currently makes use of the `ACPI20` and `SMBIOS` configuration tables; these must be present and correct.

Usage of the `ACPI` table (ie, `EFI_ACPI_TABLE_GUID`) has been superseded by the `ACPI20` table (ie, `EFI_ACPI_20_TABLE_GUID`). If both are present, the `ACPI` table will be ignored by Ubuntu.

9.5. Secure boot

Section 27 of the UEFI specification [UEFI 2.3.1] defines “Secure Boot”, a mechanism for authenticating boot images loaded by UEFI firmware. Although the description of the secure boot mechanism is comprehensive, it does not define any policy for ownership of authentication information.

Canonical, in conjunction with industry partners, has released a whitepaper [UEFI-SB] detailing the issues surrounding UEFI secure boot and Linux-based operating systems.

Canonical will provide keys and signed boot images for use with secure boot functionality. The signing key will be provided as an x.509-encapsulated 2048-bit RSA public key. OEMs must embed this key in the KEK and db signature databases, as an entry of type `EFI_CERT_X509_GUID`. The PK is left for the OEM to define.

Any machine shipped with Ubuntu must support reconfiguration of the keys used in the secure boot process, to allow users to use secure boot with their own keys and custom boot images. The firmware interface should allow a physically-present user to enter the machine in to setup mode, or manually load KEK, db and dbx entries from disk or removable storage. This requirement is compatible with the Windows 8 Hardware Certification Requirements [WIN8HCR], § System.Fundamentals.Firmware.UEFISecureBoot, item 20.

Any machine shipped with Ubuntu must allow a physically-present user to disable and re-enable secure boot verification functionality. This requirement is compatible with the Windows 8 Hardware Certification Requirements [WIN8HCR], § System.Fundamentals.Firmware.UEFISecureBoot, item 21.

Systems shipping with secure boot enabled must not use a CSM module for legacy BIOS compatibility.

Due to the very limited availability of UEFI implementations with secure boot functionality, Canonical requires additional testing effort for any SKUs that are required to support secure boot. We require that a sample SKU be provided early in the enablement process, to allow for this additional testing.

For more information on enabling Ubuntu on a system supporting secure boot, please contact Canonical.

9.6. Graphics output protocol

The Ubuntu boot process relies on the UEFI Graphics Output Protocol (GOP) for early access to display hardware. Therefore, UEFI firmware must implement this protocol for proper functionality during system boot.

Firmware GOP drivers should *not* rely on legacy-BIOS compatibility to function. Legacy VGA drivers that implement communication between software and GPU using interrupts (INT10h) and the VGA/

VBE interface, should be ported to use the UEFI Protocols as per UEFI 2.3.1 specification § 11.9, “Graphics Output Protocol”.

References

[ACPI 4.0] *Advanced Configuration and Power Interface Specification*. 4.0. Hewlett-Packard Corporation. Intel Corporation. Microsoft Corporation. Phoenix Technologies Ltd.. Toshiba Corporation. December 6, 2011. http://www.acpi.info/spec40a.htm.

[SMBIOS 2.71] *System Management BIOS Reference Specification*. 2.71. Distributed Management Task Force. January 26, 2011. http://dmtf.org/sites/default/files/standards/documents/DSP0134_2.7.1.pdf.

[UEFI 2.3.1] *Unified Extensible Firmware Interface Specification*. 2.3.1. United EFI, Inc. April 6, 2011. http://www.uefi.org/specs/.

[UEFI-SB] *UEFI Secure Boot Impact on Linux*. Jeremy Kerr. Matthew Garrett. James Bottomley. October 28, 2011. http://ozlabs.org/docs/uefi-secure-boot-impact-on-linux.pdf.

[WIN8HCR] *Windows 8 Hardware Certification Requirements*. Microsoft. December 16, 2011. http://msdn.microsoft.com/library/windows/hardware/hh748188.

[UDEV] *udev(7) manual page*. Greg Kroah-Hartman. Kay Sievers. July 11, 2011. http://manpages.ubuntu.com/manpages/precise/man7/udev.7.html.

A. Contacting Canonical

Canonical has offices in the United States, China, Taiwan, the United Kingdom, Canada, Brazil, and the Isle of Man.

For questions about this document, or clarification on technical items, send email to <`hwe-docs@lists.launchpad.net`>.

For sales and other enquiries, contact us via https://forms.canonical.com/sales/, or contact one of our local offices:

Boston office

Canonical USA Inc.
Suite 212 Lexington Corporate Center
10 Maguire Road
Lexington, MA 02421
USA

Phone: +1 781 761 9080
Fax: +1 781 862 5514

Shanghai office

Canonical China

Unit 2763, 27th Floor, K.Wah Center
No.1010 Huaihai Zhong Road
Shanghai, 200031
China

上海市淮海中路1010号嘉华中心27楼2763单元

Phone: +86 21 6103 1234

Taipei office

Canonical Limited Taiwan Branch
Room d, 46F
No. 7, Xin Yi Rd., Sec. 5
Taipei 101
Taiwan

台北市信義路5段7號46樓D室 (台北101大樓)

Phone: +886 2 8729 6888
Fax: +886 2 2723 9288

For additional contact information please see http://www.canonical.com/about-canonical/contact.

B. Existing hotkey mappings

The following tables are a selection of the current scancode to key symbol mappings used by Ubuntu. The tables are generated directly from the keymap data used in the latest version of Ubuntu.

Each table is preceeded by the conditions that are checked before loading the keymap. If the conditions match the device, then the keymap is used. Otherwise, the keymap is not used. Details about the format of these conditions are available in the udev documentation [UDEV].

1. Hotkey mapping tables

dell

`ENV{DMI_VENDOR}=="Dell*"`

scancode	keysym	notes
0xe0 0x01	KEY_PLAYPAUSE	Play/Pause
0xe0 0x02	KEY_STOPCD	Stop
0xe0 0x03	KEY_PREVIOUSSONG	Previous song
0xe0 0x04	KEY_NEXTSONG	Next song
0xe0 0x05	KEY_BRIGHTNESSDOWN	Fn+Down arrow Brightness Down
0xe0 0x06	KEY_BRIGHTNESSUP	Fn+Up arrow Brightness Up

scancode	keysym	notes
0xe0 0x07	KEY_BATTERY	Fn+F3 battery icon
0xe0 0x08	KEY_UNKNOWN	Fn+F2 Turn On/Off Wireless - handled in hardware
0xe0 0x09	KEY_EJECTCLOSECD	Fn+F10 Eject CD
0xe0 0x0a	KEY_SUSPEND	Fn+F1 hibernate
0xe0 0x0b	KEY_SWITCHVIDEOMODE	Fn+F8 CRT/LCD (high keycode: "displaytoggle")
0xe0 0x0c	KEY_F23	Fn+Right arrow Auto Brightness
0xe0 0x0f	KEY_SWITCHVIDEOMODE	Fn+F7 aspect ratio
0xe0 0x10	KEY_PREVIOUSSONG	Front panel previous song
0xe0 0x11	KEY_PROG1	Wifi Catcher (DELL Specific)
0xe0 0x12	KEY_MEDIA	MediaDirect button (house icon)
0xe0 0x13	KEY_F23	Fn+Left arrow Auto Brightness
0xe0 0x15	KEY_CAMERA	Shutter button Takes a picture if optional camera available
0xe0 0x17	KEY_EMAIL	Tablet email button
0xe0 0x18	KEY_F21	Tablet screen rotatation
0xe0 0x19	KEY_NEXTSONG	Front panel next song
0xe0 0x1a	KEY_SETUP	Tablet tools button
0xe0 0x1b	KEY_SWITCHVIDEOMODE	Display Toggle button
0xe0 0x1e	KEY_F21	touchpad toggle
0xe0 0x22	KEY_PLAYPAUSE	Front panel play/pause
0xe0 0x24	KEY_STOPCD	Front panel stop
0xe0 0x6d	KEY_MEDIA	MediaDirect button
0xe0 0x58	KEY_SCREENLOCK	Tablet lock button
0xe0 0x59	KEY_F21	touchpad toggle

dell-latitude-xt2

```
ENV{DMI_VENDOR}=="Dell*"
    && ATTR{[dmi/id]product_name}=="Latitude XT2"
```

scancode	keysym	notes
0xe0 0x1b	KEY_UP	tablet rocker up
0xe0 0x1e	KEY_ENTER	tablet rocker press
0xe0 0x1f	KEY_BACK	tablet back
0xe0 0x23	KEY_DOWN	tablet rocker down

lenovo-ideapad

```
ENV{DMI_VENDOR}=="LENOVO*"
    && ATTR{[dmi/id]product_version}=="*IdeaPad*"
ENV{DMI_VENDOR}=="LENOVO*"
    && ATTR{[dmi/id]product_name}=="S10-*"
```

scancode	keysym	notes
0xe0 0x01	KEY_RFKILL	does nothing in BIOS
0xe0 0x03	KEY_DISPLAY_OFF	BIOS toggles screen state
0xe0 0x39	KEY_BRIGHTNESSUP	does nothing in BIOS
0xe0 0x3a	KEY_BRIGHTNESSDOWN	does nothing in BIOS
0xe0 0x71	KEY_CAMERA	BIOS toggles camera power
0xe0 0x72	KEY_F21	touchpad toggle (key alternately emits f2 and f3)
0xe0 0x73	KEY_F21	

lenovo-thinkpad_x200_tablet

```
ENV{DMI_VENDOR}=="LENOVO*"
    && ATTR{[dmi/id]product_version}=="ThinkPad X2[02]* Tablet*"
    && ATTR{[dmi/id]product_version}=="* Tablet"
```

scancode	keysym	notes
0xe0 0x5d	KEY_MENU	
0xe0 0x63	KEY_FN	
0xe0 0x66	KEY_SCREENLOCK	
0xe0 0x67	KEY_CYCLEWINDOWS	bezel circular arrow
0xe0 0x68	KEY_SETUP	bezel setup / menu
0xe0 0x6c	KEY_DIRECTION	rotate screen

lenovo-thinkpad_x6_tablet

```
ENV{DMI_VENDOR}=="LENOVO*"
    && ATTR{[dmi/id]product_version}=="ThinkPad X6*"
    && ATTR{[dmi/id]product_version}=="* Tablet"
```

scancode	keysym	notes
0xe0 0x6c	KEY_F21	rotate
0xe0 0x68	KEY_SCREENLOCK	screenlock
0xe0 0x6b	KEY_ESC	escape
0xe0 0x6d	KEY_RIGHT	right on d-pad
0xe0 0x6e	KEY_LEFT	left on d-pad
0xe0 0x71	KEY_UP	up on d-pad
0xe0 0x6f	KEY_DOWN	down on d-pad
0xe0 0x69	KEY_ENTER	enter on d-pad

module-lenovo

```
ENV{DMI_VENDOR}=="LENOVO*"
    && KERNELS=="input*"
    && ATTRS{name}=="ThinkPad Extra Buttons"
```

scancode	**keysym**	**notes**
0xe0 0x01	KEY_SCREENLOCK	Fn+F2
0xe0 0x02	KEY_BATTERY	Fn+F3
0xe0 0x03	KEY_SLEEP	Fn+F4
0xe0 0x04	KEY_WLAN	Fn+F5
0xe0 0x06	KEY_SWITCHVIDEOMODE	Fn+F7
0xe0 0x07	KEY_F21	Fn+F8 touchpadtoggle
0xe0 0x08	KEY_F24	Fn+F9 undock
0xe0 0x0b	KEY_SUSPEND	Fn+F12
0xe0 0x0f	KEY_BRIGHTNESSUP	Fn+Home
0xe0 0x10	KEY_BRIGHTNESSDOWN	Fn+End
0xe0 0x11	KEY_KBDILLUMTOGGLE	Fn+PgUp - ThinkLight
0xe0 0x13	KEY_ZOOM	Fn+Space
0xe0 0x14	KEY_VOLUMEUP	
0xe0 0x15	KEY_VOLUMEDOWN	
0xe0 0x16	KEY_MUTE	
0xe0 0x17	KEY_PROG1	ThinkPad/ThinkVantage button (high key-code: "vendor")
0xe0 0x1a	KEY_MICMUTE	Microphone mute