



Ubuntu BIOS & UEFI Requirements

Canonical Ltd.

Revision: f113f73a (public build)

Date: 2021-04-30

Release: general

Table of Contents

1. Introduction	2
2. Ubuntu development	2
2.1. LTS releases	2
2.2. Reporting Ubuntu bugs	2
2.3. Kernel source code access	3
2.4. Where to find more information	3
3. Firmware Test Suite	3
3.1. Reporting FWTS bugs	3
3.2. Firmware Test Suite Live image	4
3.3. Reporting FWTS Live bugs and results	4
4. ACPI	4
4.1. ACPI functionality	5
4.2. Predefined ACPI names handled in Ubuntu	6
4.3. Supported ACPI device IDs	13
4.4. 32- and 64-bit addresses (Generic Address Structure)	14
4.5. ACPI table checksum	15
4.6. ACPI Machine Language (AML)	15
4.7. _OSI(Linux)	16
4.8. _REV	19
4.9. Battery	19
4.10. Mutexes	20
4.11. Thermal zones	20
4.12. Common Errors	21
5. Windows Management Instrumentation (WMI)	21
5.1. Common Errors	22
6. System Management Mode (SMM)	22
6.1. High Precision Event Timer (HPET)	23
7. System Management BIOS (SMBIOS)	23
7.1. Common Errors	23
8. Hotkeys	24
8.1. Keyboard BIOS hotkeys / Embedded Controller hotkeys	24
8.2. Hotkey mappings	24
8.3. ACPI Events	25
8.4. KBC scancodes	27
9. UEFI	28
9.1. Legacy BIOS compatibility	28
9.2. UEFI boot services	28
9.3. UEFI runtime services	29
9.4. UEFI configuration tables	30
9.5. UEFI boot manager	30
9.6. Secure boot	30
9.7. Graphics output protocol	33
9.8. Firmware capsule update	34
References	34
A. Contacting Canonical	34
B. Existing hotkey mappings	35
1. Hotkey mapping tables	36
C. Secure boot certificate details	39
1. Ubuntu master CA certificate	39
2. Microsoft UEFI CA certificate	40

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 pre-installed 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. The current release of Ubuntu is 20.04, which was released in April 2020. Ubuntu releases are also given a two-word informal name, comprised of a fanciful or rare animal, and an adjective. The informal name for 20.04 is “Focal Fossa”.

The current development version of Ubuntu is 21.10, codenamed “Impish Idri”, and is scheduled for release in October 2021.

Please refer to <https://wiki.ubuntu.com/ReleaseSchedule> for more information on the Ubuntu release cycle.

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 an LTS release. These releases receive security updates for a longer period (5 years) than a normal Ubuntu release (9 months).

The most recent LTS release is Ubuntu 20.04. The next LTS release will be Ubuntu 22.04. Please refer to <https://wiki.ubuntu.com/LTS> for more information about Ubuntu LTS releases.

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 a Ubuntu release, including bugs in Ubuntu’s ACPI or WMI implementation, driver bugs, or generic kernel bugs, please refer to <https://help.ubuntu.com/community/ReportingBugs> on Ubuntu Wiki.

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 Ubuntu 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 the use of the git application to access it. For example, to download the kernel source code used in the 16.04 release of Ubuntu (codename "xenial"), run the following command:

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

For access to the kernel sources used in other releases, replace `xenial` with the codename for the appropriate release. For help using the git software, please consult the git documentation¹.

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. FWTS 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 suite is hosted in a Launchpad Personal Package Archive (PPA)² belonging to `firmware-test-team`³. To install the latest 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 visit <https://help.launchpad.net/Packaging/PPA>.

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

3.1. Reporting FWTS bugs

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

¹ <http://git-scm.com/documentation>

² <https://launchpad.net/~firmware-testing-team/+archive/ubuntu/ppa-fwts-stable>

³ <https://launchpad.net/~firmware-testing-team>

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 page⁴.

3.3. Reporting FWTS Live bugs and results

If you discover any 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
Hirsute (21.04)	5.11	20200528	6.3
Groovy (20.10)	5.8	20200528	6.3
Focal (20.04)	5.4	20190816	6.3
Eoan (19.10)	5.3	20190703	6.3
Disco (19.04)	5.0	20181213	6.2
Cosmic (18.10)	4.18	20180531	6.2
Bionic (18.04)	4.15	20170831	6.2
Artful (17.10)	4.13	20170531	6.2
Zesty (17.04)	4.10	20160930	6.1
Yakkety (16.10)	4.8	20160422	6.1
Xenial (16.04)	4.4	20150930	6.0
Wily (15.10)	4.2	20150619	6.0
Vivid (15.04)	3.19	20141107	5.1
Utopic (14.10)	3.16	20140424	5.0
Trusty (14.04)	3.13	20131115	5.0
Saucy (13.10)	3.11	20130517	5.0
Raring (13.04)	3.8	20121018	5.0
Quantal (12.10)	3.5	20120320	5.0
Precise (12.04)	3.2	20110623	4.0
Oneiric (11.10)	3.0	20110413	4.0

⁴ <https://wiki.ubuntu.com/FirmwareTestSuite/FirmwareTestSuiteLive>

Ubuntu release	Kernel version	ACPICA version	ACPI version
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 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	Implemented?	Notes
APIC	Y	
BERT	N	
BGRT	Y	
BOOT	N	
CPEP	N	
CSRT	Y	
DBG2	N	See Microsoft Debug Port Specification 2
DBGP	N	See Microsoft Debug Port Specification
DMAR	Y	
DRTM	N	
DSDT	P	Some methods not fully implemented
ECDT	Y	
EINJ	Y	Ubuntu Maverick 10.10++
ERST	Y	Ubuntu Maverick 10.10++
ETDT	N	Superseded by HPET, now considered obsolete
FACS	Y	
FADT	Y	
FPDT	N	
GTDT	P	Required on arm64, not implemented on x86/x64
HEST	Y	
HPET	Y	x86/x64 only, not supported on arm64
IBFT	Y	
IORT	N	
IVRS	Y	AMD specific, Ubuntu Lucid 10.04 LTS++
LPIT	N	
MADT	Y	
MCFG	Y	
MCHI	N	
MPST	N	
Y: Implemented, N: Not implemented, P: Partially implemented		

ACPI Table	Implemented?	Notes
MSCT	N	
MSDM	N	
NFIT	Y	
OEMx	N	OEM specific tables
PCCT	Y	
PDTT	N	
PMTT	N	
PPTT	Y	
PSDT	Y	Treated like SSDT, PSDT now deprecated
RASF	N	
RSDT	Y	
SBST	N	
SLIC	N	
SLIT	Y	
SPCR	N	
SPMI	Y	
SRAT	Y	
STAO	Y	
SSDT	P	Some methods not fully implemented
TCPA	Y	
TPM2	Y	
UEFI	Y	Ubuntu Maverick 10.10++. Limited functionality
WAET	N	
WDAT	N	
WDRT	N	
WPBT	N	
XENV	N	
XSDT	Y	
Y: Implemented, N: Not implemented, P: Partially implemented		

4.2. Predefined ACPI names handled in Ubuntu

DSDT and SSDT contain byte codes that can implement a range of control methods and data objects as described in Section 5.6.8 of the ACPI specification [ACPI 6.2a]. 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 Linux ACPI driver in Ubuntu 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 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 is 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.

FWTS 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 checks over 180 of the most commonly used methods and objects.

The Ubuntu Linux ACPI driver uses just a 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.

Ubuntu kernels handle each predefined ACPI name in one of the three ways as described in the following sections.

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	Base version
<code>_ACx</code>	Lucid 10.04 LTS
<code>_ADR</code>	Dapper 6.06 LTS
<code>_AEI</code>	Quantal 12.10
<code>_ALx</code>	Lucid 10.04 LTS
<code>_ALI</code>	Quantal 12.10
<code>_ART</code>	Vivid 15.04
<code>_BBN</code>	Jaunty 9.04
<code>_BCL</code>	Dapper 6.06 LTS
<code>_BCM</code>	Dapper 6.06 LTS
<code>_BIF</code>	Dapper 6.06 LTS
<code>_BIX</code>	Maverick 10.10
<code>_BQC</code>	Dapper 6.06 LTS
<code>_BST</code>	Dapper 6.06 LTS
<code>_BTP</code>	Dapper 6.06 LTS
<code>_CBA</code>	Karmic 9.10
<code>_CCA</code>	Xenial 16.04 LTS
<code>_CID</code>	Karmic 9.10
<code>_CLS</code>	Xenial 16.04 LTS
<code>_CPC</code>	Xenial 16.04 LTS
<code>_CRS</code>	Dapper 6.06 LTS

Ubuntu BIOS & UEFI Requirements

Method/Object	Base version
_CRT	Dapper 6.06 LTS
_CST	Dapper 6.06 LTS
_DCK	Hardy 8.04 LTS
_DCS	Dapper 6.06 LTS
_DDC	Dapper 6.06 LTS
_DEP	Wily 15.10
_DGS	Dapper 6.06 LTS
_DIS	Dapper 6.06 LTS
_DOD	Dapper 6.06 LTS
_DOS	Dapper 6.06 LTS
_DSD	Vivid 15.04
_DSM	Karmic 9.10
_DSS	Dapper 6.06 LTS
_DSW	Jaunty 9.04
_Exx	Dapper 6.06 LTS
_EC	Dapper 6.06 LTS
_EJD	Dapper 6.06 LTS
_EJx	Hardy 8.04 LTS
_EVT	Yakkety 16.10
_FIF	Utopic 14.10
_FPS	Utopic 14.10
_FSL	Utopic 14.10
_FST	Utopic 14.10
_GAI	Maverick 10.10
_GHL	Lucid 10.04 LTS
_GL	Dapper 6.06 LTS
_GLK	Dapper 6.06 LTS
_GPD	Dapper 6.06 LTS
_GPE	Dapper 6.06 LTS
_GSB	Karmic 9.10
_GTF	Dapper 6.06 LTS
_GTM	Dapper 6.06 LTS
_HID	Natty 11.04
_HOT	Dapper 6.06 LTS
_HPP	Karmic 9.10
_HPX	Karmic 9.10
_HRV	Yakkety 16.10
_IFT	Maverick 10.10
_INI	Dapper 6.06 LTS

Ubuntu BIOS & UEFI Requirements

Method/Object	Base version
_IRC	Dapper 6.06 LTS
_Lxx	Dapper 6.06 LTS
_LCK	Dapper 6.06 LTS
_LID	Dapper 6.06 LTS
_LPI	Yakkety 16.10
_LSI	Bionic 18.04 LTS
_LSR	Bionic 18.04 LTS
_LSW	Bionic 18.04 LTS
_MAT	Hardy 8.04 LTS
_OFF	Dapper 6.06 LTS
_ON	Dapper 6.06 LTS
_OSC	Karmic 9.10
_OSI	Dapper 6.06 LTS
_OST	Lucid 10.04 LTS
_PAI	Maverick 10.10
_PCT	Dapper 6.06 LTS
_PDC	Dapper 6.06 LTS
_PIC	Karmic 9.10
_PLD	Quantal 12.10
_PMC	Lucid 10.04 LTS
_PMD	Lucid 10.04 LTS
_PMM	Lucid 10.04 LTS
_PPC	Dapper 6.06 LTS
_PR	Dapper 6.06 LTS
_PR0	Dapper 6.06 LTS
_PR1	Karmic 9.10
_PR2	Karmic 9.10
_PR3	Karmic 9.10
_PRS	Karmic 9.10
_PRT	Karmic 9.10
_PRW	Dapper 6.06 LTS
_PS0	Dapper 6.06 LTS
_PS1	Karmic 9.10
_PS2	Karmic 9.10
_PS3	Dapper 6.06 LTS
_PSC	Dapper 6.06 LTS
_PSD	Hardy 8.04 LTS
_PSL	Dapper 6.06 LTS
_PSR	Dapper 6.06 LTS

Ubuntu BIOS & UEFI Requirements

Method/Object	Base version
_PSS	Dapper 6.06 LTS
_PSV	Dapper 6.06 LTS
_PSW	Dapper 6.06 LTS
_PTC	Dapper 6.06 LTS
_PTP	Lucid 10.04 LTS
_PTS	Dapper 6.06 LTS
_PUR	Lucid 10.04 LTS
_PXM	Dapper 6.06 LTS
_Qxx	Dapper 6.06 LTS
_REG	Dapper 6.06 LTS
_REV	Dapper 6.06 LTS
_RMV	Dapper 6.06 LTS
_ROM	Dapper 6.06 LTS
_RST	Wily 15.10
_S0	Dapper 6.06 LTS
_S1	Dapper 6.06 LTS
_S2	Dapper 6.06 LTS
_S3	Dapper 6.06 LTS
_S4	Dapper 6.06 LTS
_S5	Dapper 6.06 LTS
_S1D	Dapper 6.06 LTS
_S2D	Dapper 6.06 LTS
_S3D	Dapper 6.06 LTS
_S4D	Dapper 6.06 LTS
_S0W	Karmic 9.10
_S1W	Karmic 9.10
_S2W	Karmic 9.10
_S3W	Karmic 9.10
_S4W	Karmic 9.10
_SB	Dapper 6.06 LTS
_SBS	Dapper 6.06 LTS
_SCP	Dapper 6.06 LTS
_SDD	Hardy 8.04 LTS
_SEG	Dapper 6.06 LTS
_SHL	Lucid 10.04 LTS
_SI	Dapper 6.06 LTS
_SPD	Dapper 6.06 LTS
_SRS	Dapper 6.06 LTS
_SST	Dapper 6.06 LTS

Method/Object	Base version
_STA	Dapper 6.06 LTS
_STM	Hardy 8.04 LTS
_STR	Dapper 6.06 LTS
_SUB	Raring 13.04
_SUN	Jaunty 9.04
_T_x	Dapper 6.06 LTS
_TC1	Dapper 6.06 LTS
_TC2	Dapper 6.06 LTS
_TMP	Dapper 6.06 LTS
_TPC	Hardy 8.04 LTS
_TRT	Vivid 15.04
_TSD	Hardy 8.04 LTS
_TSP	Dapper 6.06 LTS
_TSS	Dapper 6.06 LTS
_TTS	Jaunty 9.04
_TZ	Dapper 6.06 LTS
_TZD	Dapper 6.06 LTS
_TZM	Karmic 9.10
_TZP	Dapper 6.06 LTS
_UID	Dapper 6.06 LTS
_UPC	Karmic 9.10
_VP0	Dapper 6.06 LTS
_WAK	Dapper 6.06 LTS
_Wxx	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	Base version
_ALC	Karmic 9.10
_ALP	Karmic 9.10
_ALR	Natty 11.04
_ALT	Karmic 9.10
_BCT	Lucid 10.04 LTS
_BDN	Lucid 10.04 LTS
_BLT	Karmic 9.10
_BMA	Lucid 10.04 LTS

Ubuntu BIOS & UEFI Requirements

Method/Object	Base version
_BMC	Karmic 9.10
_BMD	Karmic 9.10
_BMS	Lucid 10.04 LTS
_BTH	Wily 15.10
_BTM	Karmic 9.10
_CDM	Lucid 10.04 LTS
_CSD	Karmic 9.10
_CWS	Saucy 13.10
_DDN	Karmic 9.10
_DLM	Saucy 13.10
_DMA	Karmic 9.10
_DPL	Quantal 12.10
_DRS	Precise 12.04
_DTI	Lucid 10.04 LTS
_EDL	Karmic 9.10
_FDE	Maverick 10.10
_FDI	Karmic 9.10
_FDM	Karmic 9.10
_FIX	Karmic 9.10
_GCP	Saucy 13.10
_GRT	Saucy 13.10
_GWS	Saucy 13.10
_MBM	Lucid 10.04 LTS
_MLS	Karmic 9.10
_MSG	Karmic 9.10
_MSM	Lucid 10.04 LTS
_MTL	Wily 15.10
_NBS	Eoan 19.10
_NCH	Eoan 19.10
_NIC	Eoan 19.10
_NIH	Eoan 19.10
_NIG	Eoan 19.10
_NTT	Lucid 10.04 LTS
_PCL	Karmic 9.10
_PDL	Lucid 10.04 LTS
_PIF	Lucid 10.04 LTS
_PPE	Karmic 9.10
_PRE	Utopic 14.10
_PRL	Lucid 10.04 LTS

Method/Object	Base version
_PRR	Wily 15.10
_PSE	Saucy 13.10
_RDI	Saucy 13.10
_RTV	Karmic 9.10
_RXL	Saucy 13.10
_SLI	Karmic 9.10
_SLV	Quantal 12.10
_SRT	Saucy 13.10
_SRV	Karmic 9.10
_STB	Quantal 12.10
_STP	Lucid 10.04 LTS
_STV	Lucid 10.04 LTS
_SWS	Karmic 9.10
_TDL	Natty 11.04
_TFP	Wily 15.10
_TIP	Lucid 10.04 LTS
_TIV	Lucid 10.04 LTS
_TPT	Karmic 9.10
_TSN	Wily 15.10
_TST	Saucy 13.10
_TXL	Quantal 12.10
_UPD	Karmic 9.10
_UPP	Karmic 9.10

4.2.3. Deprecated ACPI names

4.2.3.1. _GTS and _BFS

_GTS (Going to Sleep) and _BFS (Back from Sleep) control methods are introduced in ACPI specification 2.0. However, they are rarely seen in any systems in the field. As a result, proposal of removing them in the next revision of ACPI specification has been submitted, and support for _GTS and _BFS had already been removed in Linux kernel.

4.3. Supported ACPI device IDs

Certain integrated devices require support for some device-specific ACPI controls. Section 5.6.7 of the ACPI specification [ACPI 6.2a] lists these devices and their corresponding Plug-and-Play (PNP) IDs. 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 does not necessarily mean that the device is not supported, it just means that Ubuntu will ignore the device description/configuration.

Table 5. ACPI Device IDs

PNP _HID	Description	Base support version
PNP0A05	Generic Container Device	Hardy 8.04 LTS
PNP0A06	Generic Container Device	Hardy 8.04 LTS
PNP0C09	Embedded Controller	Dapper 6.06 LTS
PNP0C0A	Contol Method Battery	Dapper 6.06 LTS
PNP0C0B	Fan	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	Vivid 15.04
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	Wily 15.10
ACPI0009	I/OxAPIC Device	Wily 15.10
ACPI000A	I/O APIC Device	Wily 15.10
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
ACPI000F	User Presence Detection Device	No
ACPI0010	User Presence Detection Device	Yakkety 16.10
ACPI0011	Generic Buttons Device	No
ACPI0012	NVDIMM Root Device	Wily 15.10
ACPI0013	Generic Event Device	Yakkety 16.10

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 the ACPI specification is that the 64-bit address supersedes the 32-bit address, if the 64-bit address is non-zero. ACPI specification 6.1 explicitly states 32-bit and 64-bit addresses should be mutually exclusive, i.e. one must be zero if the other one is non-zero. However, some firmware sets both field and 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 recommend 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.

FWTS 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 Specification [ACPI 6.2a].

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 20 of the ACPI specification [ACPI 6.2a]. ACPI control methods are compiled into AML and this code is executed inside the kernel context by the Linux ACPI driver. Usually, AML is compiled using either the Microsoft AML or Intel compilers, but they can produce different output from the same source code. Ubuntu uses the Intel AML compiler.

- We recommend 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 the ACPI specification [ACPI 6.2a] 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, behaviours, 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 `drivers/acpi/osl.c` of the kernel source code.

_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**. 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 6. _OSI support

_OSI argument	Windows version	Supported in Ubuntu
<i>Windows 2000</i>	Windows 2000	Pre-Dapper 6.06 LTS
<i>Windows 2001</i>	Windows XP	Pre-Dapper 6.06 LTS
<i>Windows 2001</i>	Windows XP SP1	Pre-Dapper 6.06 LTS
<i>Windows 2001.1</i>	Windows Server 2003	Pre-Dapper 6.06 LTS
<i>Windows 2001 SP2</i>	Windows XP SP2	Pre-Dapper 6.06 LTS
<i>Windows 2001.1 SP1</i>	Windows Server 2003 SP1	Hardy 8.04 LTS
<i>Windows 2006</i>	Windows Vista	Hardy 8.04 LTS
<i>Windows 2006.1</i>	Windows Server 2008	Lucid 10.04 LTS
<i>Windows 2006 SP1</i>	Windows Vista SP1	Lucid 10.04 LTS
<i>Windows 2006 SP2</i>	Windows Vista SP2	Natty 11.04
<i>Windows 2009</i>	Windows 7 and Server 2008 R2	Quantal 12.10
<i>Windows 2012</i>	Windows 8 and Server 2012	Raring 13.04
<i>Windows 2013</i>	Windows 8.1 and Windows Server 2012 R2	Utopic 14.10
<i>Windows 2015</i>	Windows 10	Wily 15.10
<i>Windows 2016</i>	Windows 10 (1607)	Cosmic 18.10
<i>Windows 2017</i>	Windows 10 (1703)	Cosmic 18.10
<i>Windows 2017.2</i>	Windows 10 (1709)	Cosmic 18.10
<i>Windows 2018</i>	Windows 10 (1803)	Disco 19.04

_OSI argument	Windows version	Supported in Ubuntu
<i>Windows 2018.2</i>	Windows 10 (1809)	Disco 19.04
<i>Windows 2019</i>	Windows 10 (1903)	Eoan 19.10

_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 OS behaviour compatibility string is passed to _OSI. See Table 7, “Banned I/O ports” for the list of I/O ports that are always banned to AML byte code.

Table 7. 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 8, “Windows XP+ banned I/O ports” lists these ranges.

Table 8. 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
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 a 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 an RTC device without directly accessing I/O ports 0x70 and 0x71. The example below demonstrates how the BIOS can declare Seconds, Minutes and Hours registers in an RTC device. More details for different RTC devices can be found in Section 9.15 of ACPI specification [ACPI 6.2a].

```
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 FWTS using the klog test, run FWTS as follows:

```
sudo fwts klog
```

4.7.2. _OSI OEM interface

Linux tries to be compatible with Windows, nevertheless it is not Windows. As BIOS needs to act differently in some cases when running in Linux and Windows, there is an update⁵ in Linux kernel v4.10 which outlines how to create a custom _OSI string as summarized below. More details can be found in Documentation/acpi/osi.txt in Linux kernel source code.

- Use _OSI("Linux-OEM-my_interface_name") in BIOS ACPI ASL.

⁵ <https://github.com/torvalds/linux/commit/bc18461816bd3a6a141e472f68c886bb932a6c2e>

- Add a "acpi_osi=Linux-OEM-my_interface_name" kernel parameter to Ubuntu OEM images.
- Submit a kernel patch to include "Linux-OEM-my_interface_name" for future compatibilities.

ACPI ASL sample code is as below. It is important to as specific as possible when using `_OSI` OEM interface because this can break future compatibilities easily. Please contact your representative if `_OSI` OEM interface is needed in your BIOS image.

```
If(_OSI("Linux-XYZ-USB")) {  
    // Linux USB-specific functions for OEM XYZ  
} Else {  
    // Windows USB-specific functions for OEM XYZ  
}
```

4.8. `_REV`

Section 5.7.4 of the ACPI specification 5.1 describes the `_REV` (Revision Data Object) object. This object originally provides firmware the ability to evaluate the revisions of the ACPI specification to determine the set of ACPI related interfaces, behaviours or features. Ubuntu 15.10 reports its value to be 5, and Windows returns 2. As a result, many systems abuse this difference as a way to distinguish Linux from Windows.

Starting from ACPI specification 6, `_REV`'s definition was changed to the length of an Integer supported by ACPI specifications. Ubuntu 16.04 and after reports `_REV` to be 2 (supports 32 and 64 bits). This is an attempt to be compatible with latest versions of Windows, and you can find the rationale from this blog post⁶ of the Linux developer who made this change. The implementation of `_REV` can be found in `drivers/acpi/osl.c` in the kernel source.

The implication of this change is you can no longer exploit `_REV` to determine whether the running operating system is Linux or Windows. Although we do not recommend you to do so, but if you have such the need, please refer to Section 4.7.2, "`_OSI` OEM interface".

4.9. 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 [ACPI 6.2a] 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.

⁶ <https://mjg59.dreamwidth.org/34542.html>

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 0x00000000-0xffffffffe 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 `_BIF` 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 the ACPI specification [ACPI 6.2a]. 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 bit 0 and bit 1 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.

4.10. 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.

4.11. 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.

4.12. Common Errors

4.12.1. Brightness Levels

ACPI specification [ACPI 6.2a] defines the `_BCL` object to return a package containing a list of the supported brightness levels. Specifically, **the first element represents the brightness level on full power and the second element represents the brightness level on batteries**. The remaining elements represent brightness level supported by hotkeys. However, it is commonly seen that the first two elements are omitted, causing problems with the Linux kernel. It is important to specify the first two elements for brightness levels for full-power and for on-batteries according to the ACPI specification.

4.12.2. Current Brightness Level

ACPI specification [ACPI 6.2a] defines the `_BQC` object for an operating system to query the current brightness level. In practice, a number of common errors are observed and listed below:

- `_BQC` returns zero at boot time. Ubuntu kernel uses `_BQC` to initialize LCD brightness at boot time, and a value of zero will dim the LCD to an unreadable level. We recommend that the BIOS either sets a default value or restores the previous brightness level. Restoration of the previous brightness level is highly preferred.
- `_BQC` always returns a fixed value. In this case, the BIOS is telling the Ubuntu kernel that the brightness level never changes.
- `_BQC` returns a value not present in `_BCL`. A common error is that the BIOS returns a *level* of brightness (for example, 5th level) instead of the *value* of the brightness (for example, 50). This inconsistent return value means that the Ubuntu kernel cannot determine the current brightness level, nor the new brightness level to set.

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 an 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. For example, "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 an SMI port (e.g. port 0xb2 on Intel platforms), or by an 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 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 (i.e. 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 BIOS BITS (BIOS Implementation Test Suite)⁷ 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 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 uses when handling SMIs.

7. System Management BIOS (SMBIOS)

The System Management BIOS (SMBIOS) specification [SMBIOS 3.2] 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 - 0A0A0A0A0A0A0A0A0A
- 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 3.2]; 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.

⁷ <https://biosbits.org/>

In addition to SMBIOS, ACPI defines Preferred_PM_Profile in Fixed ACPI Description Table (FADT) [ACPI 6.2a]. It is a very common error that the SMBIOS Type 3 data and ACPI Preferred_PM_Profile do 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.

7.1.3. Out of range value (Multiple SMBIOS Types)

It is commonly seen that many SMBIOS structures including (but not limited to) Types 3, 17 and 26 to have out of range values in particular fields. It can be the results of default values that need updating by BIOS during POST but BIOS fails to do so. It is recommended to run FWTS using the `dmicheck` test:

```
sudo fwts dmicheck
```

8. Hotkeys

8.1. Keyboard BIOS hotkeys / Embedded Controller hotkeys

Hotkeys can be implemented in two different approaches: as BIOS ACPI events or KBC scancodes.

BIOS ACPI events (sometimes known as “BIOS hotkeys”) are implemented by triggering GPE events and executing BIOS AML. Ubuntu supports many standard ACPI events to handle BIOS hotkeys.

KBC scancodes (sometimes known as “Embedded Controller hotkeys”) are a mechanism where scancodes are sent directly via the keyboard controller. The Linux kernel receives keyboard codes directly from 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.

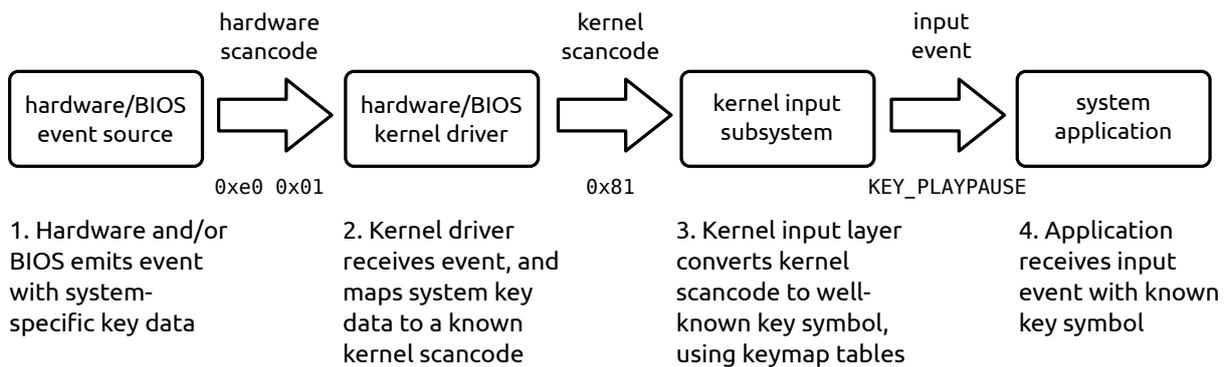
We recommend that hotkeys implemented in this manner should re-use standard ACPI events and already-existing KBC scancodes and keymaps, keeping consistent with previous SKUs. Refer to Section 8.3, “ACPI Events” for common ACPI hotkey events supported by Ubuntu. Refer to Section 8.2, “Hotkey mappings” for the mapping process, and Appendix B, *Existing hotkey mappings* for details of existing mappings used by Ubuntu. Some functions can be implemented by either ACPI events or by KBC scancodes, but existing approaches should be prioritized.

A well-behaved EC for this hotkey should send both key press (make) and release (break) scan codes in the correct order.

8.2. 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*.

8.3. ACPI Events

8.3.1. Brightness interfaces and hotkeys

Brightness interfaces and hotkeys on mobile platforms are implemented in various ways on different platforms. The common approaches are either by native backlights such as intel_backlight and amdgpu_bl0, or to implement the ACPI brightness control methods, detailed in Section B.6 of the ACPI specification [ACPI 6.2a]. Native backlights are similar to Windows approaches and are the default in Ubuntu Linux; however, it is up to hardware designers to choose an appropriate brightness interface. ACPI brightness can be chosen as default by the kernel parameter **acpi_backlight=video**. 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 9, “Notification Values for Brightness Control”.

Table 9. 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.

8.3.2. Video output hotkeys

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

- Same image on both displays (“clone” or “mirror”)
- Builtin display only (LVDS)

- External display only
- Extended display (both displays active, desktop extended over both displays)

ACPI defines standard approaches for BIOS to notify the Ubuntu kernel that a VGA switch event occurs, as listed in Table 9, “Notification Values for Brightness Control” and detailed in Appendix B of [ACPI 6.2a].

Table 10. ACPI VGA notifications for hotkeys

Notification value	VGA function
0x80	ACPI_VIDEO_NOTIFY_SWITCH
0x81	ACPI_VIDEO_NOTIFY_PROBE
0x82	ACPI_VIDEO_NOTIFY_CYCLE
0x83	ACPI_VIDEO_NOTIFY_NEXT_OUTPUT
0x84	ACPI_VIDEO_NOTIFY_NEXT_OUTPUT

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

Video mode switching is handled by the OS system application `unity-settings-daemon`. This daemon responds to the appropriate 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.

8.3.2.1. 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 utilities 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.

8.3.3. 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.

8.3.4. 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.

8.3.5. 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_FPFB	Power Button
ACPI_FSB	Sleep Button
PNP0C0C	Power Button
PNP0C0D	Lid Device
PNP0C0E	Sleep Button

8.4. KBC scancodes

8.4.1. Multimedia hotkeys

Keyboard controllers (KBC) or embedded controllers (EC) are commonly used to support multimedia hotkeys. Some USB keyboards also provide similar functionality.

Ubuntu is capable of handling a number of common scan codes for multimedia hotkeys, as listed in Table 12, “Scan codes for multimedia hotkeys”. We strongly recommend that system designers to re-use these existing mappings.

Table 12. Scan codes for multimedia hotkeys

Key Name	Make code (press)	Break code (release)
Next Track	0xE0 0x19	0xE0 0x99
Previous Track	0xE0 0x10	0xE0 0x90
Stop	0xE0 0x24	0xE0 0xA4
Play/Pause	0xE0 0x22	0xE0 0xA2
Mute	0xE0 0x20	0xE0 0xA0
Volume up	0xE0 0x30	0xE0 0xB0
Volume down	0xE0 0x32	0xE0 0xB2

8.4.2. 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. Udev keymapping is defined in `/lib/udev/hwdb.d/60-keyboard.hwdb` (in previous Ubuntu releases before systemd migration, mappings are define in files under `/lib/udev/keymaps/`). These key events can then be used to inform the driver to disable/re-enable wireless, via the Ubuntu kernel rkill interface.

An example of the mapping for Dell laptops is the Fn+F2 key, which emits scancode `0xe0 0x08`.

Note

If a driver provided by an OEM/ODM/IHV does not support the standard kernel rkill 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.

8.4.3. 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 rkill key is pressed. This maps to the F21 key event (touchpad toggle) via the udev rules in `/lib/udev/hwdb.d/60-keyboard.hwdb`.

The touchpad toggle key event is processed by `unity-settings-daemon`, which will trigger the on-screen notification. It will also 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 `unity-settings-daemon` explicitly handle the actual enable/disable logic, nor 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 of the UEFI standard.

9.1. Legacy BIOS compatibility

Ubuntu has been tested with UEFI-only mode BIOS and is also known to work with legacy BIOS compatibility mode (known as a Compatibility Support Module, or “CSM”) enabled. However, enabling legacy support will affect secure boot functionality, because it needs native-UEFI drivers and to be signed for validation during booting up.

9.2. UEFI boot services

Table 13, “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 13. Boot services used

Service	UEFI §	Compliance
<code>EFI_INSTALL_CONFIGURATION_TABLE</code>	6.5	Required
<code>EFI_LOCATE_PROTOCOL</code>	6.3	Required
<code>EFI_LOCATE_HANDLE</code>	6.3	Required
<code>EFI_OPEN_HANDLE</code>	6.3	Required
<code>EFI_STALL</code>	6.5	Required
<code>EFI_EXIT</code>	6.4	Required
<code>EFI_SET_WATCHDOG_TIMER</code>	6.5	Optional, watchdog is disabled
<code>EFI_ALLOCATE_PAGES</code>	6.2	Required
<code>EFI_FREE_PAGES</code>	6.2	Required

Service	UEFI §	Compliance
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 14, “Boot protocols used” lists the boot protocols that may be referenced by the Ubuntu bootloader.

Table 14. 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 15, “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 15. 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_SET_VARIABLE	7.2	Required
EFI_GET_NEXT_VARIABLE_NAME	7.2	Required
EFI_RESET_SYSTEM	7.5.1	Required
EFI_UPDATE_CAPSULE	7.5.3	Optional, required for firmware update
EFI_QUERY_CAPSULE_CAPABILITIES	7.5.3	Optional, required for firmware update

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

The latest SMBIOS3.0 has been released in 2015, unlike the original SMBIOS 2.1 (32-bit) entry point, allows the SMBIOS structure table to reside anywhere in 32-bit physical address space (that is, below 4 GB), the SMBIOS 3.0 (64-bit) entry point allows the SMBIOS structure table to reside anywhere in 64-bit memory. If an implementation provides both a 32-bit and a 64-bit entry point and they point to distinct SMBIOS structure tables, the 32-bit table must be a consistent subset of the 64-bit table: for any structure type (between 0 and 125) that exists in the 32-bit table, there must be a corresponding structure in the 64-bit table. The 64-bit table may contain structure types not found in the 32-bit table. On UEFI-based systems, the SMBIOS Entry Point structure can be located by looking in the EFI Configuration Table for the `SMBIOS_TABLE_GUID`, the SMBIOS3.0 Entry Point structure can be located by looking in the EFI Configuration Table for the `SMBIOS3_TABLE_GUID`.

9.5. UEFI boot manager

During Ubuntu installation, a bootloader option (i.e. `Boot#### ubuntu "\EFI\ubuntu\shimx64.efi"`) is created. This option is set as the default boot option through the `BootOrder` setting.

Firmware should not enforce any boot policy other than the mechanism specified in Section 3 of the UEFI specification [UEFI 2.7a]. Specifically, **firmware should not modify boot behaviour when the removable media (i.e. `\EFI\BOOT\BOOT{machine type short-name}.EFI`) exists as well.**

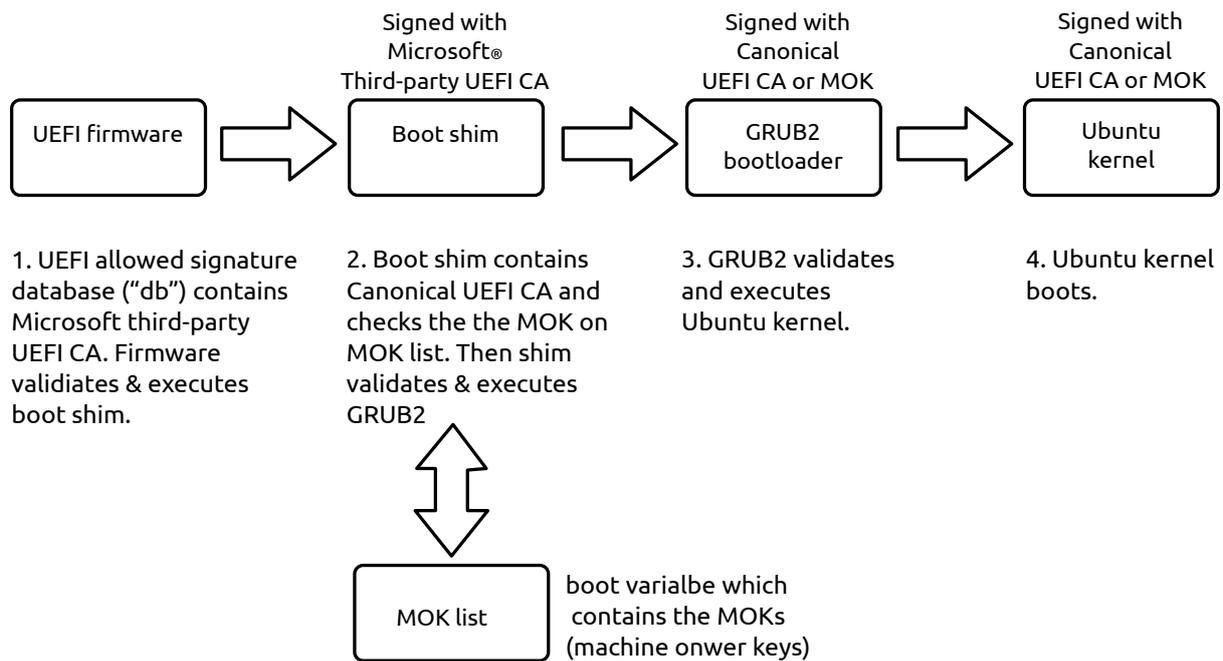
9.6. Secure boot

Section 31 of the UEFI specification [UEFI 2.7a] 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.

When deployed in secure boot configurations, the Ubuntu boot process uses a small “boot shim”, which allows compatibility with the Microsoft Third-party CA certificate, and also provides the MOK (Machine owner keys) database mechanism, which allows machine owner to enroll their own keys for their own signed kernels or bootloaders. Figure 2, “UEFI secure boot process” shows the trust configuration of the boot chain.

Figure 2. UEFI secure boot process



Canonical will provide keys and signed boot images for use with secure boot functionality.

Ideally, **platforms should be provisioned in setup mode**. The Ubuntu first-boot process will initialise the firmware signature databases as required, and transition the machine from setup mode to user mode.

If shipping machines in setup mode is not possible, Ubuntu requires the following signature database configuration:

- The Ubuntu certificate should be present in KEK. Exact details of this certificate are provided in Appendix C, Section 1, “Ubuntu master CA certificate”.
- The Microsoft UEFI CA certificate should be present in db. Exact details of this certificate are provided in Appendix C, Section 2, “Microsoft UEFI CA certificate”.

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.

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

9.6.1. Ubuntu secure boot certificate

The master Ubuntu certificate is a 2048-bit RSA public key, encapsulated in a self-signed x.509 certificate, DER-encoded. This certificate allows Ubuntu machines to be updated with new key database entries when they become available.

To ensure that you have the correct certificate data, calculate the SHA-256 checksum of the certificate data, using the `sha256sum` command at the Linux terminal, and compare your output to the following:

```
$ sha256sum < ubuntu-master-public.der
ed1fe72cb9ca31c9af5b757afcd733323d675825032e6ced7fe1ae9eb767998c -
```

9.6.1.1. Embedding requirements

UEFI firmware should embed this certificate in the Key Exchange Key signature database (EFI variable "KEK"). The certificate should appear as an entry of type `EFI_CERT_X509_GUID`. The following definitions describe an appropriate `EFI_SIGNATURE_LIST` structure to contain the certificate:

```
/* EFI GUID for Canonical */
#define CANONICAL_GUID \
    {0x6dc40ae4, 0x2ee8, 0x9c4c, \
     {0xa3, 0x14, 0x0f, 0xc7, 0xb2, 0x00, 0x87, 0x10}}

/* Size of the DER-encoded x.509 Ubuntu certificate */
#define UBUNTU_SIGNATURE_SIZE 1080

/* Signature database entry */
EFI_SIGNATURE_LIST {
    SignatureType = EFI_CERT_X509_GUID,
    SignatureListSize = 1124, /* = sizeof(EFI_SIGNATURE_LIST) */
                               /* + sizeof(EFI_SIGNATURE_DATA) */
                               /* + UBUNTU_SIGNATURE_SIZE */
    SignatureHeaderSize = 0,
    SignatureSize = 1096, /* = sizeof(EFI_SIGNATURE_DATA) */
                          /* + UBUNTU_SIGNATURE_SIZE */
    SignatureHeader[] = [], /* empty */
    Signatures = EFI_SIGNATURE_DATA {
        SignatureOwner = CANONICAL_GUID,
        SignatureData = [<signature-file-data>]
    }
}
```

Like all GUIDs referenced in the EFI specification, `CANONICAL_GUID` is encoded as little-endian. The EFI-standard textual representation of the GUID is:

```
6dc40ae4-2ee8-9c4c-a314-0fc7b2008710
```

Therefore, the actual bytes encoded in `SignatureOwner` will be:

```
const unsigned char canonical_guid_bytes[] = {
    0xe4, 0x0a, 0xc4, 0x6d, 0xe8, 0x2e, 0x4c, 0x9c,
    0xa3, 0x14, 0x0f, 0xc7, 0xb2, 0x00, 0x87, 0x10
};
```

9.6.1.2. Firmware requirements

UEFI Firmware must be able to follow the following trust path:

1. UEFI boot image is signed with a RSA2048 signing key.
2. A certificate containing this signing key is embedded in the PE/COFF PKCS7 signedData structure, in the certificates list. This certificate is signed by the master key.
3. A certificate containing the master certificate (the Microsoft UEFI CA) is embedded in the UEFI db signature database.

Additionally, the firmware **must** be able to iterate all potential trust paths in the system. For instance, if there are multiple certificates embedded in the PE/COFF image, then the firmware must attempt to authenticate the image with all of these certificates before rejecting the image. Specifically, the firmware must not abort verification on the first failed certificate.

This support for multiple trust paths will allow certificate change-over periods, when the signing certificate may be signed by multiple master certificates. Since only one of these master certificates may be present in the UEFI signature database, firmware must not abort authentication until all potential trust paths fail.

9.6.2. Stricter secure boot enforcement

Since Ubuntu 16.04, secure boot policy enforcement extends to the kernel, which means when secure boot is enabled, unsigned kernel will no longer be launched and unsigned kernel modules will not be loaded. As a result, this change also affects the usage of Dynamic Kernel Module Support⁸ (DKMS). A shim-signed dialogue prompts end users to disable secure boot while DKMS packages are being installed, it allows users to disable shim validation, like what the command `mokutil --disable-validation` does, so that end users can keep using the DKMS packages needed for their system devices.

If DKMS packages are required for the system, here are some solutions so that DKMS packages can be preloaded:

- Turn off secure boot in system firmware, which disables the secure boot processes defined in Section 31 of the UEFI specification [UEFI 2.7a].
- Disable shim validation while keeping secure boot in system firmware enabled. Shim validation can be enabled/disabled by the `mokutil` tool with these commands:

```
mokutil --disable-validation
```

```
mokutil --enable-validation
```

These commands basically add/delete the UEFI variable “MokSBState”. Shim checks the value of “MokSBState” to decide whether to perform signature validation or not. Adding the “MokSBState” variable by default on the system can disable shim validation.

For more information on disabling shim validation, please contact Canonical.

9.7. 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. GOP must support minimum resolution 1024x768, with or without EDID. Without GOP, Ubuntu cannot use X Window with framebuffer.

UEFI firmware must detect all output devices connected to the POST display adapter. UEFI firmware must install GOP with or without physical output device for proper Ubuntu installation process.

⁸ <https://github.com/dell/dkms>

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, per UEFI specification § 11.9, “Graphics Output Protocol”.

9.8. Firmware capsule update

Ubuntu has been tested with firmware capsule update. For supporting capsule update, firmware must have the implementation below:

- Capsule related runtime services in Section 8.5 of the UEFI specification [UEFI 2.7a], including UpdateCapsule, QueryCapsuleCapabilities and also the ability of exchanging information between the OS and firmware (i.e. through the OsIndicationsSupported and the OsIndications variables).
- The EFI System Resource Table (ESRT) and protocols defined in Section 23 of the UEFI specification [UEFI 2.7a].

There are two tools which can be used to update capsules under Ubuntu:

- `fwupdate`, it uses ESRT and UpdateCapsule to apply firmware updates locally. Please refer to <https://github.com/rhinstaller/fwupdate> for more information.
- `fwupd` (firmware update daemon), it allows updating capsules locally or from Linux Vendor Firmware Service (LVFS). Using `fwupd`, firmware needs to be packaged up in a **cab** file which includes an **inf** file that describes the update in more detail and also add extra metadata that it can have fully localized update descriptions. Please refer to <http://fwupd.org/> for more information.

References

[ACPI 6.2a] *Advanced Configuration and Power Interface Specification*. 6.2. Unified EFI Forum, Inc. September, 2017. <http://uefi.org/specifications>.

[SMBIOS 3.2] *System Management BIOS Reference Specification*. 3.2. Distributed Management Task Force. February 12, 2015. <https://www.dmtf.org/standards/smbios>.

[UEFI 2.7a] *Unified Extensible Firmware Interface Specification*. 2.7a. United EFI Forum, Inc. September, 2017. <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>.

[HWDB] *hwdb(7) manual page*. <http://manpages.ubuntu.com/manpages/xenial/man7/hwdb.7.html>.

A. Contacting Canonical

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

For questions about this document, or clarification on technical items, send email to [<hwe-docs@lists.launchpad.net>](mailto: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 below. For additional contact information, please visit <https://www.canonical.com/about#contact-row>.

London office

Canonical Group Ltd.
5th Floor Blue Fin Building
110 Southwark Street
London, SE1 0SU
UK

Phone: +44 020 7630 2400

Beijing office

Canonical China Limited, Beijing Branch
Unit 1118-19, Level 11, China World Office 1
Jianguomenwai Ave., Chaoyang District
Beijing, 100004
China

北京市朝阳区建国门外大街1号国贸写字楼1座11层1118-1119室

Phone: +86 (10) 5829 1826

Taipei office

Canonical Group 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

B. Existing hotkey mappings

The mappings are defined by udev rules, and can be found in `/lib/udev/hwdb.d/60-keyboard.hwdb`. The format is as follows:

```
KEYBOARD_KEY_<hex scan code>=<key code identifier> #comments
```

For example:

```
KEYBOARD_KEY_81=playpause           # Play/Pause
KEYBOARD_KEY_82=stopcd              # Stop
KEYBOARD_KEY_83=previous song       # Previous song
```

```
KEYBOARD_KEY_84=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/uapi/linux/input-event-codes.h.

If you look at /lib/udev/hwdb.d/60-keyboard.hwdb, you can find there are sections for each OEM. Canonical can work with you to create new rules in the file and submit to upstream. 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.

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 LTS version of Ubuntu.

Each table is preceded by the conditions that are checked before loading the keymap. If the conditions match the device, then the keymap is used. Details about the format of these conditions are available in the udev's hwdb documentation [HWDB].

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
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

scancode	keysym	notes
0xe0 0x17	KEY_EMAIL	Tablet email button
0xe0 0x18	KEY_F21	Tablet screen rotation
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_name}=="* 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	

scancode	keysym	notes
0xe0 0x16	KEY_MUTE	
0xe0 0x17	KEY_PROG1	ThinkPad/ThinkVantage button (high keycode: "vendor")
0xe0 0x1a	KEY_MICMUTE	Microphone mute

C. Secure boot certificate details

The following sections provide exact details of secure boot certificates referenced in Section 9.6, "Secure boot"

1. Ubuntu master CA certificate

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 13348991040521802343 (0xb94124a0182c9267)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=GB, ST=Isle of Man, L=Douglas, O=Canonical Ltd., CN=Canonical
Ltd. Master Certificate Authority
    Validity
      Not Before: Apr 12 11:12:51 2012 GMT
      Not After : Apr 11 11:12:51 2042 GMT
    Subject: C=GB, ST=Isle of Man, L=Douglas, O=Canonical Ltd.,
CN=Canonical Ltd. Master Certificate Authority
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:bf:5b:3a:16:74:ee:21:5d:ae:61:ed:9d:56:ac:
        bd:de:de:72:f3:dd:7e:2d:4c:62:0f:ac:c0:6d:48:
        08:11:cf:8d:8b:fb:61:1f:27:cc:11:6e:d9:55:3d:
        39:54:eb:40:3b:b1:bb:e2:85:34:79:ca:f7:7b:bf:
        ba:7a:c8:10:2d:19:7d:ad:59:cf:a6:d4:e9:4e:0f:
        da:ae:52:ea:4c:9e:90:ce:c6:99:0d:4e:67:65:78:
        5d:f9:d1:d5:38:4a:4a:7a:8f:93:9c:7f:1a:a3:85:
        db:ce:fa:8b:f7:c2:a2:21:2d:9b:54:41:35:10:57:
        13:8d:6c:bc:29:06:50:4a:7e:ea:99:a9:68:a7:3b:
        c7:07:1b:32:9e:a0:19:87:0e:79:bb:68:99:2d:7e:
        93:52:e5:f6:eb:c9:9b:f9:2b:ed:b8:68:49:bc:d9:
        95:50:40:5b:c5:b2:71:aa:eb:5c:57:de:71:f9:40:
        0a:dd:5b:ac:1e:84:2d:50:1a:52:d6:e1:f3:6b:6e:
        90:64:4f:5b:b4:eb:20:e4:61:10:da:5a:f0:ea:e4:
        42:d7:01:c4:fe:21:1f:d9:b9:c0:54:95:42:81:52:
        72:1f:49:64:7a:c8:6c:24:f1:08:70:0b:4d:a5:a0:
        32:d1:a0:1c:57:a8:4d:e3:af:a5:8e:05:05:3e:10:
        43:a1
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Subject Key Identifier:
        AD:91:99:0B:C2:2A:B1:F5:17:04:8C:23:B6:65:5A:26:8E:34:5A:63
      X509v3 Authority Key Identifier:
  
```

keyid:AD:91:99:0B:C2:2A:B1:F5:17:04:8C:23:B6:65:5A:26:8E:34:5A:63

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Key Usage:

Digital Signature, Certificate Sign, CRL Sign

X509v3 CRL Distribution Points:

Full Name:

URI:http://www.canonical.com/secure-boot-master-ca.crl

Signature Algorithm: sha256WithRSAEncryption

3f:7d:f6:76:a5:b3:83:b4:2b:7a:d0:6d:52:1a:03:83:c4:12:
a7:50:9c:47:92:cc:c0:94:77:82:d2:ae:57:b3:99:04:f5:32:
3a:c6:55:1d:07:db:12:a9:56:fa:d8:d4:76:20:eb:e4:c3:51:
db:9a:5c:9c:92:3f:18:73:da:94:6a:a1:99:38:8c:a4:88:6d:
c1:fc:39:71:d0:74:76:16:03:3e:56:23:35:d5:55:47:5b:1a:
1d:41:c2:d3:12:4c:dc:ff:ae:0a:92:9c:62:0a:17:01:9c:73:
e0:5e:b1:fd:bc:d6:b5:19:11:7a:7e:cd:3e:03:7e:66:db:5b:
a8:c9:39:48:51:ff:53:e1:9c:31:53:91:1b:3b:10:75:03:17:
ba:e6:81:02:80:94:70:4c:46:b7:94:b0:3d:15:cd:1f:8e:02:
e0:68:02:8f:fb:f9:47:1d:7d:a2:01:c6:07:51:c4:9a:cc:ed:
dd:cf:a3:5d:ed:92:bb:be:d1:fd:e6:ec:1f:33:51:73:04:be:
3c:72:b0:7d:08:f8:01:ff:98:7d:cb:9c:e0:69:39:77:25:47:
71:88:b1:8d:27:a5:2e:a8:f7:3f:5f:80:69:97:3e:a9:f4:99:
14:db:ce:03:0e:0b:66:c4:1c:6d:bd:b8:27:77:c1:42:94:bd:
fc:6a:0a:bc

2. Microsoft UEFI CA certificate

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

61:08:d3:c4:00:00:00:00:00:04

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation,
CN=Microsoft Corporation Third Party Marketplace Root

Validity

Not Before: Jun 27 21:22:45 2011 GMT

Not After : Jun 27 21:32:45 2026 GMT

Subject: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation,
CN=Microsoft Corporation UEFI CA 2011

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

00:a5:08:6c:4c:c7:45:09:6a:4b:0c:a4:c0:87:7f:
06:75:0c:43:01:54:64:e0:16:7f:07:ed:92:7d:0b:
b2:73:bf:0c:0a:c6:4a:45:61:a0:c5:16:2d:96:d3:
f5:2b:a0:fb:4d:49:9b:41:80:90:3c:b9:54:fd:e6:
bc:d1:9d:c4:a4:18:8a:7f:41:8a:5c:59:83:68:32:
bb:8c:47:c9:ee:71:bc:21:4f:9a:8a:7c:ff:44:3f:
8d:8f:32:b2:26:48:ae:75:b5:ee:c9:4c:1e:4a:19:

7e:e4:82:9a:1d:78:77:4d:0c:b0:bd:f6:0f:d3:16:
d3:bc:fa:2b:a5:51:38:5d:f5:fb:ba:db:78:02:db:
ff:ec:0a:1b:96:d5:83:b8:19:13:e9:b6:c0:7b:40:
7b:e1:1f:28:27:c9:fa:ef:56:5e:1c:e6:7e:94:7e:
c0:f0:44:b2:79:39:e5:da:b2:62:8b:4d:bf:38:70:
e2:68:24:14:c9:33:a4:08:37:d5:58:69:5e:d3:7c:
ed:c1:04:53:08:e7:4e:b0:2a:87:63:08:61:6f:63:
15:59:ea:b2:2b:79:d7:0c:61:67:8a:5b:fd:5e:ad:
87:7f:ba:86:67:4f:71:58:12:22:04:22:22:ce:8b:
ef:54:71:00:ce:50:35:58:76:95:08:ee:6a:b1:a2:
01:d5

Exponent: 65537 (0x10001)

X509v3 extensions:

1.3.6.1.4.1.311.21.1:

.....

1.3.6.1.4.1.311.21.2:

....k..wSJ.%7.N.&{. p.

X509v3 Subject Key Identifier:

13:AD:BF:43:09:BD:82:70:9C:8C:D5:4F:31:6E:D5:22:98:8A:1B:D4

1.3.6.1.4.1.311.20.2:

.S.u.b.C.A

X509v3 Key Usage:

Digital Signature, Certificate Sign, CRL Sign

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Authority Key Identifier:

keyid:45:66:52:43:E1:7E:58:11:BF:D6:4E:9E:23:55:08:3B:3A:22:6A:A8

X509v3 CRL Distribution Points:

Full Name:

URI:http://crl.microsoft.com/pki/crl/products/MicCorThiParMarRoo_2010-10-05.crl

Authority Information Access:

CA Issuers -

URI:http://www.microsoft.com/pki/certs/MicCorThiParMarRoo_2010-10-05.crt

Signature Algorithm: sha256WithRSAEncryption

35:08:42:ff:30:cc:ce:f7:76:0c:ad:10:68:58:35:29:46:32:
76:27:7c:ef:12:41:27:42:1b:4a:aa:6d:81:38:48:59:13:55:
f3:e9:58:34:a6:16:0b:82:aa:5d:ad:82:da:80:83:41:06:8f:
b4:1d:f2:03:b9:f3:1a:5d:1b:f1:50:90:f9:b3:55:84:42:28:
1c:20:bd:b2:ae:51:14:c5:c0:ac:97:95:21:1c:90:db:0f:fc:
77:9e:95:73:91:88:ca:bd:bd:52:b9:05:50:0d:df:57:9e:a0:
61:ed:0d:e5:6d:25:d9:40:0f:17:40:c8:ce:a3:4a:c2:4d:af:
9a:12:1d:08:54:8f:bd:c7:bc:b9:2b:3d:49:2b:1f:32:fc:6a:
21:69:4f:9b:c8:7e:42:34:fc:36:06:17:8b:8f:20:40:c0:b3:
9a:25:75:27:cd:c9:03:a3:f6:5d:d1:e7:36:54:7a:b9:50:b5:
d3:12:d1:07:bf:bb:74:df:dc:1e:8f:80:d5:ed:18:f4:2f:14:
16:6b:2f:de:66:8c:b0:23:e5:c7:84:d8:ed:ea:c1:33:82:ad:
56:4b:18:2d:f1:68:95:07:cd:cf:f0:72:f0:ae:bb:dd:86:85:
98:2c:21:4c:33:2b:f0:0f:4a:f0:68:87:b5:92:55:32:75:a1:
6a:82:6a:3c:a3:25:11:a4:ed:ad:d7:04:ae:cb:d8:40:59:a0:
84:d1:95:4c:62:91:22:1a:74:1d:8c:3d:47:0e:44:a6:e4:b0:
9b:34:35:b1:fa:b6:53:a8:2c:81:ec:a4:05:71:c8:9d:b8:ba:

e8:1b:44:66:e4:47:54:0e:8e:56:7f:b3:9f:16:98:b2:86:d0:
68:3e:90:23:b5:2f:5e:8f:50:85:8d:c6:8d:82:5f:41:a1:f4:
2e:0d:e0:99:d2:6c:75:e4:b6:69:b5:21:86:fa:07:d1:f6:e2:
4d:d1:da:ad:2c:77:53:1e:25:32:37:c7:6c:52:72:95:86:b0:
f1:35:61:6a:19:f5:b2:3b:81:50:56:a6:32:2d:fe:a2:89:f9:
42:86:27:18:55:a1:82:ca:5a:9b:f8:30:98:54:14:a6:47:96:
25:2f:c8:26:e4:41:94:1a:5c:02:3f:e5:96:e3:85:5b:3c:3e:
3f:bb:47:16:72:55:e2:25:22:b1:d9:7b:e7:03:06:2a:a3:f7:
1e:90:46:c3:00:0d:d6:19:89:e3:0e:35:27:62:03:71:15:a6:
ef:d0:27:a0:a0:59:37:60:f8:38:94:b8:e0:78:70:f8:ba:4c:
86:87:94:f6:e0:ae:02:45:ee:65:c2:b6:a3:7e:69:16:75:07:
92:9b:f5:a6:bc:59:83:58