NAME

dmidecode − DMI table decoder

SYNOPSIS

dmidecode [**OPTIONS**]

DESCRIPTION

dmidecode is a tool for dumping a computer's DMI (some say SMBIOS) table contents in a human-readable format. This table contains a description of the system's hardware components, as well as other useful pieces of information such as serial numbers and BIOS revision. Thanks to this table, you can retrieve this information without having to probe for the actual hardware. While this is a good point in terms of report speed and safeness, this also makes the presented information possibly unreliable.

The DMI table doesn't only describe what the system is currently made of, it also can report the possible evolutions (such as the fastest supported CPU or the maximal amount of memory supported).

SMBIOS stands for System Management BIOS, while DMI stands for Desktop Management Interface. Both standards are tightly related and developed by the DMTF (Desktop Management Task Force).

As you run it, **dmidecode** will try to locate the DMI table. It will first try to read the DMI table from sysfs, and next try reading directly from memory if sysfs access failed. If**dmidecode** succeeds in locating a v alid DMI table, it will then parse this table and display a list of records like this one:

Handle 0x0002, DMI type 2, 8 bytes. Base Board Information
Manufacturer: Intel
Product Name: C440GX+
Version: 727281-001
Serial Number: INCY92700942

Each record has:

- A handle. This is a unique identifier, which allows records to reference each other. For example, processor records usually reference cache memory records using their handles.
- A type. The SMBIOS specification defines different types of elements a computer can be made of. In this example, the type is 2, which means that the record contains "Base Board Information".
- A size. Each record has a 4-byte header (2 for the handle, 1 for the type, 1 for the size), the rest is used by the record data. This value doesn't take text strings into account (these are placed at the end of the record), so the actual length of the record may be (and is often) greater than the displayed value.
- Decoded values. The information presented of course depends on the type of record. Here, we learn about the board's manufacturer, model, version and serial number.

OPTIONS

-d, **--dev-mem FILE**
Read memory from device **FILE** (default: **/dev/mem**)

-q, **--quiet**
Be less verbose. Unknown, inactive and OEM-specific entries are not displayed. Meta-data and handle references are hidden.

-s, **--string KEYWORD**
Only display the value of the DMI string identified by **KEYWORD**. **KEYWORD** must be a keyword from the following list: **bios-vendor**, **bios-version**, **bios-release-date**, **bios-revision**, **firmware-revision**, **system-manufacturer**, **system-product-name**, **system-version**, **system-serial-number**, **system-uuid**, **system-sku-number**, **system-family**, **baseboard-manufacturer**, **baseboard-product-name**, **baseboard-version**, **baseboard-serial-number**, **baseboard-asset-tag**, **chassis-manufacturer**, **chassis-type**, **chassis-version**, **chassis-serial-number**, **chassis-asset-tag**, **processor-family**, **processor-manufacturer**, **processor-version**, **processor-frequency**.

Each keyword corresponds to a given DMI type and a given offset within this entry type. Not all strings may be meaningful or even defined on all systems. Some keywords may return more than one result on some systems (e.g. **pr ocessor-version** on a multi-processor system). If **KEYWORD** is not provided or not valid, a list of all valid keywords is printed and **dmidecode** exits with an error. This option cannot be used more than once.

Note: on Linux, most of these strings can alternatively be read directly from **sysfs**, typically from files under */sys/devices/virtual/dmi/id*. Most of these files are even readable by regular users.

-t, **--type TYPE**
Only display the entries of type **TYPE**. **TYPE** can be either a DMI type number, or a comma-separated list of type numbers, or a keyword from the following list: **bios**, **system**, **baseboard**, **chassis**, **processor**, **memory**, **cache**, **connector**, **slot**. Refer to the DMI TYPES section below for details. If this option is used more than once, the set of displayed entries will be the union of all the given types. If **TYPE** is not provided or not valid, a list of all valid keywords is printed and **dmidecode** exits with an error.

-H, **--handle HANDLE**
Only display the entry whose handle matches **HANDLE**. **HANDLE** is a 16-bit integer.

-u, **--dump**
Do not decode the entries, dump their contents as hexadecimal instead. Note that this is still a text output, no binary data will be thrown upon you. The strings attached to each entry are displayed as both hexadecimal and ASCII. This option is mainly useful for debugging.

--dump-bin FILE
Do not decode the entries, instead dump the DMI data to a file in binary form. The generated file is suitable to pass to **--from-dump** later.

--from-dump FILE
Read the DMI data from a binary file previously generated using **--dump-bin**.

--no-sysfs
Do not attempt to read DMI data from sysfs files. This is mainly useful for debugging.

--oem-string N
Only display the value of the OEM string number **N**. The first OEM string has number 1. With special value "count", return the number of OEM strings instead.

-h, **--help**
Display usage information and exit

-V, **--version**
Display the version and exit

Options --string, --type, --dump-bin and --oem-string determine the output format and are mutually exclusive.

Please note in case of **dmidecode** is run on a system with BIOS that boasts new SMBIOS specification, which is not supported by the tool yet, it will print out relevant message in addition to requested data on the very top of the output. Thus informs the output data is not reliable.

DMI TYPES

The SMBIOS specification defines the following DMI types:

Type	Information
0	BIOS
1	System
2	Baseboard
3	Chassis
4	Processor

5 Memory Controller
6 Memory Module
7 Cache
8 Port Connector
9 System Slots
10 On Board Devices
11 OEM Strings
12 System Configuration Options
13 BIOS Language
14 Group Associations
15 System Event Log
16 Physical Memory Array
17 Memory Device
18 32-bit Memory Error
19 Memory Array Mapped Address
20 Memory Device Mapped Address
21 Built-in Pointing Device
22 Portable Battery
23 System Reset
24 Hardware Security
25 System Power Controls
26 Voltage Probe
27 Cooling Device
28 Temperature Probe
29 Electrical Current Probe
30 Out-of-band Remote Access
31 Boot Integrity Services
32 System Boot
33 64-bit Memory Error
34 Management Device
35 Management Device Component
36 Management Device Threshold Data
37 Memory Channel
38 IPMI Device
39 Power Supply
40 Additional Information
41 Onboard Devices Extended Information
42 Management Controller Host Interface

Additionally, type 126 is used for disabled entries and type 127 is an end-of-table marker. Types 128 to 255 are for OEM-specific data. **dmidecode** will display these entries by def ault, but it can only decode them when the vendors have contributed documentation or code for them.

Keywords can be used instead of type numbers with **--type**. Each keyword is equivalent to a list of type numbers:

Keyword	Types
bios	0, 13
system	1, 12, 15, 23, 32
baseboard	2, 10, 41
chassis	3
processor	4
memory	5, 6, 16, 17

cache	7
connector	8
slot	9

Keywords are matched case-insensitively. The following command lines are equivalent:

- dmidecode --type 0 --type 13
- dmidecode --type 0,13
- dmidecode --type bios
- dmidecode --type BIOS

BINARY DUMP FILE FORMAT

The binary dump files generated by --dump-bin and read using --from-dump are formatted as follows:

- The SMBIOS or DMI entry point is located at offset 0x00. It is crafted to hard-code the table address at offset 0x20.
- The DMI table is located at offset 0x20.

UUID FORMAT

There is some ambiguity about how to interpret the UUID fields prior to SMBIOS specification version 2.6. There was no mention of byte swapping, and RFC 4122 says that no byte swapping should be applied by default. However, SMBIOS specification version 2.6 (and later) explicitly states that the first 3 fields of the UUID should be read as little-endian numbers (byte-swapped). Furthermore, it implies that the same was already true for older versions of the specification, even though it was not mentioned. In practice, many hardware vendors were not byte-swapping the UUID. So, in order to preserve compatibility, it was decided to interpret the UUID fields according to RFC 4122 (no byte swapping) when the SMBIOS version is older than 2.6, and to interpret the first 3 fields as little-endian (byte-swapped) when the SMBIOS version is 2.6 or later. The Linux kernel follows the same logic.

FILES

/dev/mem
/sys/firmware/dmi/tables/smbios_entry_point (Linux only)
/sys/firmware/dmi/tables/DMI (Linux only)

BUGS

More often than not, information contained in the DMI tables is inaccurate, incomplete or simply wrong.

AUTHORS

Alan Cox, Jean Delvare

SEE ALSO

biosdecode(8), **mem**(4), **ownership**(8), **vpddecode**(8)