4 Machine description

To describe the resources within a virtual machine (or logical domain), a data structure called a machine description (MD) is made available to the guest running in each logical domain / virtual machine environment.

This section describes the transport format for the machine description (MD).

This format is provided for the contract between the producer of the MD (typically the Service Entity) and the consumers in the logical domains (for example, OBP boot firmware and the Solaris OS.)

4.1 Requirements

The format of the machine description is designed so that any consumer may either elect to read and transform it into an internal representation, or merely use it in place. For the latter, the encoding needs to be easily readable with an efficient encoding. Similarly a simple encoding requirement also exists for the system software responsible for generating a particular machine description.

A hypervisor will provide a machine description as a whole to a guest operating system upon request in response to an API call. The machine description is written into a buffer owned by the guest, and not shared with any other guest or with the hypervisor. Once provided it is truly private to the guest. Therefore, there is no requirement that the encoding format support any form of dynamic update or extension. Updates to a machine description are indicated by providing a complete new machine description.

4.2 Sections

The machine description is provided in four sections as illustrated below and described below.

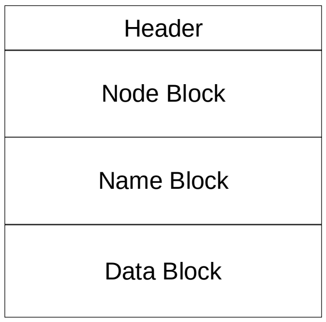

Figure 1Machine Description sections

These sections are linearly concatenated together to provide a single machine description.

4.3 Encoding

Unless otherwise specified, all fields described herein are encoded in network byte order (big-endian).

Unless otherwise specified, all fields are packed without intervening padding, and have no required byte alignment.

Where alignment is specified, it is defined in relation to the first byte of the machine description header.

4.4 Header

The format for the machine description header is defined below:

Byte offset	Size in bytes	Field name	Description
0	4	transport_version	Transport Version number
4	4	node_blk_sz	Size in bytes of node block
8	4	name_blk_sz	Size in bytes of name block
12	4	data_blk_sz	Size in bytes of data block

The header is easily described by the following packed C structure for a big-endian machine:

```
struct MD_HEADER {
        uint32_t     transport_version;
        uint32_t     node_blk_sz;
        uint32_t     name_blk_sz;
        uint32_t     data_blk_sz;
};
```

The transport_version specifies the version encoding that applies to this MD. The transport version is a 32bit integer value. The upper 16bits correspond to a major version number, the lower 16bits correspond to a minor version number change.

4.4.1 Version numbering

The `transport_version` number for this specification is 0x10000, namely version 1.0.

An increase in the minor number of the transport version corresponds to the compatible addition or removal of information encoded in the machine description. This includes, but is not limited to, the removal of certain property types, or the addition of new property types. Guests can expect to be able to decode some, but not all of the Machine Description, and must handle this expectation accordingly by ignoring unknown types.

Future specification revisions defining new element types found outside a node encapsulation (e.g. between NODE_END and NODE) are considered incompatible and require an increase in the major version number of the MD transport header.

4.4.2 Size fields

- Each size field describes the size in bytes of the remaining three blocks in the machine description.
- The node block follows immediately after the section header.
- The name block starts at byte offset: 16+ node_blk_sz.
- The data block starts at byte offset: 16 + node_blk_sz + name_blk_sz.
- All sizes are multiples of 16 bytes.
- The total size of the MD is 16 + node_blk_sz + name_blk_sz + data_blk_sz.

- Each section (sizes; node_blk_sz, name_blk_sz, data_blk_sz) may be a maximum of 2^{32}-16 bytes in length.

Note: The name block and data block sections are described below first, to assist in understanding of the subsequent node block description.

4.5 Name Block

The name block provides name strings to be used for node entry naming. Legal name strings are defined as follows:

A name string is a human readable string comprised of an unaligned linear array of bytes (characters) terminated by a zero byte (nul '\0' character). Null termination enables the use of C functions such as `strcmp`(3) for comparison.

Character encoding consists of all human readable letters and symbols from ISO standard 8859-1 not including: blanks, "/", "\", ";", "[", "]", "@".

Each name string is referenced by its starting byte offset within the name block.

Name string lengths are stored along with the byte offset in the node elements, limiting name length to 255 bytes, not including the terminating null character.

There may not be duplicate strings in the name block; a given name string may appear only once in the name block. Thus the offset within the name block becomes a unique identifier for a given name string within a machine description.

A single name string may be referenced from more than one node element.

The name block is padded with zero bytes to ensure that the subsequent data block is aligned on a 16 byte boundary relative to the start of the machine description. These pad bytes are included in the name block size.

Note: The name block contains name strings that are held independently from the data block section in order to assist with accelerated string lookups. This technique is described later in section .

4.6 Data Block

The data block provides raw data that may be referenced by nodes in the node block.

Raw data associated with node block elements is simply a linear concatenation of the raw data itself and has no further intrinsic structure. The size, location and content of each data element is identified by the referring element in the node block.

Data block contents are unaligned unless specified as part of the referring property's requirements. When alignment is required it is considered relative to the first byte of the overall machine description. Alignment is achieved by preceeding a data element with zero bytes in the data block.

The producer of a machine description is required to arrange that data requiring a specific alignement in the MD is placed on an appropriate alignment boundary relative to the start of the MD. The consumer of an MD is required to read the machine description into a buffer aligned correctly for the largest alignment requirement the consumer may have, or be prepared to handle unaligned data references correctly.

4.7 Node Block

The node block is comprised of a linear array of 16 byte elements aligned on a 16byte boundary relative to the first byte of the entire machine description.

The node block elements have specific types and are grouped as defined below so as to form “nodes” of data. Each element is of fixed length, and each element may be uniquely identified by its index within the node block array.

Any element A may refer to another element B simply by using the array index for the location of element B. For example, the first element of the node block has index value 0, the second has index 1, and so on.

4.7.1 Element format

Elements within the node block have a fixed 16byte length format comprised of big-endian fields described below:

Byte offset	Size in bytes	Field name	Description
0	1	tag	Type of element
1	1	name_len	Length in bytes of element name. Element name is located in the name block.
2	2	_reserved_field	reserved field (contains bytes of value 0)
4	4	name_offset	Location offset of name associated with this element relative to start of name block.
8	8	val	64 bit value for elements of tag type “NODE”, “PROP_VAL” or “PROP_ARC”
8	4	data_len	Length in bytes of data in data block for elements of type “PROP_STR” and of type “PROP_DATA”
12	4	data_offset	Location offset of data associated with this element relative to start of data block for elements of tag type “PROP_STR” and “PROP_DATA”

For a big-endian machine this is illustrated by the packed C structure below:

```
struct MD_ELEMENT {
      uint8_t      tag;
      uint8_t      name_len;
      uint16_t     _reserved_field;
      uint32_t     name_offset;
      union {
            struct  {
                  uint32_t     data_len;
                  uint32_t     data_offset;
            } y;
            uint64_t     val;
      } d;
};
```

The tag field defines how each element should be interpreted.

The name associated with this element is given by the name_offset and name_len fields giving the offset within the name block and length of the node name not including the terminating null character.

The remainder of the node element has two formats depending upon the node tag field.

The node element either contains a 64bit immediate data value, or (for elements requiring an extended data or string) it consists of two 32bit values providing the size and offset of the relevant data within the data block.

4.7.2 Tag definitions

Note: Element tag enumerations are chosen so that an ASCII dump of the node section will reveal each element type thus aiding debugging.

The following element tag types are defined:

Tag Value	ASCII equiv	Name	Description	Value field
0x0	\0	LIST_END	End of element list	-
0x4e	'N'	NODE	Start of node definition	64bit index to next node in list of nodes
0x45	'E'	NODE_END	End of node definition	-
0x20	' '	NOOP	NOOP list element - to be ignored	0
0x61	'a'	PROP_ARC	Node property arc'ing to another node	64bit index of node referenced
0x76	'v'	PROP_VAL	Node property with an integer value	64bit integer value for property
0x73	's'	PROP_STR	Node property with a string value	offset and length of string within data block
0x64	'd'	PROP_DATA	Node property with a block of data	offset and length of property data with in the data block

4.7.3 Nodes

The array of elements in the node block form a sequence of “nodes” terminated by a single LIST_END element.

- A node is itself a linear sequence of two or more elements whose first element is NODE and whose last element is NODE_END.
- Between NODE and NODE_END there are zero or more elements that define properties for that node. These are PROP_* elements. The ordering of these elements (between NODE and NODE_END) does not confer meaning.
- The name given to a NODE element is non-unique and defines the binding of property elements that may be encapsulated within that node.
- The NOOP element is provided so that an entire node may be removed by overwriting all of its constituent elements with NOOP. A NODE link that arrives at a NOOP element is equivalent to the next NODE or LIST_END element after the sequence of NOOP elements.
- The PROP_ARC element is used to denote an arc in a DAG, therefore a PROP_ARC element may only reference a NODE element.
    - *Note: A node referenced by any PROP_ARC element cannot be removed by use of NOOP element unless all the referring PROP_ARC elements are removed. PROP_ARC elements may be removed by conversion to a NOOP element.*

- The element index of a "NODE" element is serves as a unique identification of a complete node and its encapsulated properties.
- The value field associated with a "NODE" element (elem_ptr->d.val) holds the element index to the next "NODE" element within the MD.
    - *A reader may skip from one node to the next without having to scan within each node for the "NODE_END" by using this index value to locate the next NODE element in the node block.*