NVMe Namespace Data Generator

**Table of Contents**

[1 Overview 3](#_Toc428452192)

[2 Command line format 3](#_Toc428452193)

[3 Format Of Namespace Specification file 3](#_Toc428452194)

[3.1 Number formats 3](#_Toc428452195)

[3.2 Comments 3](#_Toc428452196)

[3.3 Namespace Specifier 3](#_Toc428452197)

[3.3.1 nsze: namespace size 4](#_Toc428452198)

[3.3.2 ncap: namespace capacity 4](#_Toc428452199)

[3.3.3 nuse: namespace utilization 4](#_Toc428452200)

[3.3.4 nsfeat: Namespace Features 4](#_Toc428452201)

[3.3.5 mc: Metadata capabilities 5](#_Toc428452202)

[3.3.6 dpc: End to end Data protection Capabilities 5](#_Toc428452203)

[3.3.7 dps: End to end data protection settings 6](#_Toc428452204)

[3.3.8 nmic: Namespace Multi-path I/O and Namespace Sharing capabilities 7](#_Toc428452205)

[3.3.9 rescap: reservation Capabilities 7](#_Toc428452206)

[3.3.10 eui64: Extended Unique Identifier 8](#_Toc428452207)

[3.3.11 Euioff 8](#_Toc428452208)

[3.3.12 Euiman 8](#_Toc428452209)

[3.3.13 lbaf: LBA Format 8](#_Toc428452210)

[3.3.14 vs: Vendor specific 9](#_Toc428452211)

[3.4 Controller Assignments 9](#_Toc428452212)

[3.5 RAM aliases 10](#_Toc428452213)

[3.6 Assignment Policy 10](#_Toc428452214)

[4 Sample namespace specification file 10](#_Toc428452215)

# Overview

The NVMe namespace generator (nsgen) is a text based generator that takes a namespace description file and delivers a C language constant file that gives the equivalent information in C compliable form.

# Command line format

Use the following command line format:

nsgen <infile> <outfile>

The infile file is the text file used to generate the namespace. The outfile is the output file that is generated. The names are used exactly as specified. Recommend you use a name like x.c for the output file so that it can be compiled.

# Format Of Namespace Specification file

The input file consists of a series of namespace specifiers, controller assignments, ram alias statements, or policy statements, with optional intermixed comments. Each will be described in turn.

## Number formats

Where numerical formats are specified, any of the following are acceptable:

123 Decimal

0x123 Hex

8m Mega (the number \* 1048576)

8g Giga (the number \* 1073741824)

## Comments

Anywhere within the file can appear a comment of the form:

! <comment>

This is the “!” character, followed by any characters up to the end of the line. All characters on the line up to and including the “!” char are simply discarded.

## Namespace Specifier

A namespace specifier is of the form:

**namespace** [field]… **nameend**

That is, each namespace that appears starts with the word **namespace** and ends with **nameend**. The format is free flowing, that is, it is up to you to have statements appear on the same line, or multiple lines.

A series of fields appear within the namespace and are assigned to the namespace. If a field appears more than one time, it simply overwrites the previous contents of the field. The exception to this rule is the field **lbaf**, for which all values that appear are collected and used to form the LBA format table.

Note that there are often default values for fields in a namespace. These will be specified on a per field basis.

### nsze: namespace size

Format:

**nsze** <size>

Size is a number, in either decimal or hex, that gives the total size of the namespace in terms of LBAs. The LBA indicated is the one marked as the formatter LBA (see **lbaf** below).

Default:

None. It is an error if this field is not defined.

### ncap: namespace capacity

Format:

**ncap** <size>

Size is a number, in either decimal or hex, that gives the total size of the namespace in terms of LBAs. The LBA indicated is the one marked as the formatter LBA (see lbaf below).

Default:

Automatically set to the same size as the nsze field.

### nuse: namespace utilization

Format:

**nuse** <size>

Size is a number, in either decimal or hex, that gives the total size of the allocated blocks in the namespace in terms of LBAs. The LBA indicated is the one marked as the formatter LBA (see **lbaf** below).

Default:

Automatically set to 0

### nsfeat: Namespace Features

Format:

**nsfeat** [<flags>]…

Gives a series of flags:

**dealloc** Indicates that the controller supports the Deallocated or Unwritten Logical Block error for this namespace. If the flag does not appear, then the controller does not support the Deallocated or Unwritten Logical Block error for this namespace.

**altflag** Indicates that the fields NAWUN, NAWUPF, and NACWU are defined for this namespace and should be used by the host for this namespace instead of the AWUN, AWUPF, and ACWU fields in the Identify Controller data structure. If the flag does not appear, then the controller does not support the fields NAWUN, NAWUPF, and NACWU for this namespace. In this case, the host should use the AWUN, AWUPF, and ACWU fields defined in the Identify Controller data structure.

**thin** Indicates that the namespace supports thin provisioning. Specifically, the Namespace Capacity reported may be less than the Namespace Size. When this feature is supported and the Dataset Management command is supported then deallocating LBAs shall be reflected in the Namespace Utilization field. If the flag does not appear, it indicates that thin provisioning is not supported and the Namespace Size and Namespace Capacity fields report the same value.

Default:

Defaults to 0 (no flags set).

### mc: Metadata capabilities

Format:

**mc** [<flags>]…

Gives a series of flags. The flags are:

**extdata** Indicates the namespace supports the metadata being transferred as part of a separate buffer that is specified in the Metadata Pointer. If the flag does not appear, it indicates that the namespace does not support the metadata being transferred as part of a separate buffer.

**extlba** Indicates that the namespace supports the metadata being transferred as part of an extended data LBA. If the flag does not appear, it indicates that the namespace does not support the metadata being transferred as part of an extended data LBA.

Default:

Defaults to 0 (no flags set).

### dpc: End to end Data protection Capabilities

Format:

**dpc** [<flags>]…

Gives a series of flags. The flags are:

**last8** Indicates that the namespace supports protection information transferred as the last eight bytes of metadata. If the flag does not appear, it indicates that the namespace does not support protection information transferred as the last eight bytes of metadata.

**first8** Indicates that the namespace supports protection information transferred as the first eight bytes of metadata. If the flag does not appear, it indicates that the namespace does not support protection information transferred as the first eight bytes of metadata.

**prot3** Indicates that the namespace supports Protection Information Type 3. If the flag does not appear, it indicates that the namespace does not support Protection Information Type 3.

**prot2** Indicates that the namespace supports Protection Information Type 2. If the flag does not appear, it indicates that the namespace does not support Protection Information Type 2.

**prot1** Indicates that the namespace supports Protection Information Type 1. If the flag does not appear, it indicates that the namespace does not support Protection Information Type 1.

Default:

Defaults to 0 (no flags set).

### dps: End to end data protection settings

Format:

**dps** [<flags>]…

Gives a series of flags. The flags are:

**first8** Indicates that the protection information, if enabled, is transferred as the first eight bytes of metadata. If the flag does not appear, it indicates that the protection information, if enabled, is transferred as the last eight bytes of metadata.

**prot1** Enable protection information, type 1.

**prot2** Enable protection information, type 2.

**prot3** Enable protection information, type 3.

Default:

Defaults to 0 (no flags set).

### nmic: Namespace Multi-path I/O and Namespace Sharing capabilities

Format:

**nmic** [<flags>]…

Gives a series of flags. The flags are:

**share** Indicates the namespace may be accessible by two or more controllers in the NVM subsystem (i.e., may be a shared namespace). If the flag does not appear, it indicates the namespace is a private namespace and may only be accessed by the controller that returned this namespace data structure.

Default:

Defaults to 0 (no flags set).

### rescap: reservation Capabilities

Format:

**rescap** [<flags>]…

Gives a series of flags. The flags are:

**allexc** Indicates that the namespace supports the Exclusive Access – All Registrants reservation type. If the flag does not appear, then the namespace does not support the Exclusive Access – All Registrants reservation type.

**wrtexc** Indicates that the namespace supports the Write Exclusive – All Registrants reservation type. If the flag does not appear, then the namespace does not support the Write Exclusive – All Registrants reservation type.

**excreg** Indicates that the namespace supports the Exclusive Access – Registrants Only reservation type. If the flag does not appear, then the namespace does not support the Exclusive Access – Registrants Only reservation type.

**wrtexcreg** Indicates that the namespace supports the Write Exclusive – Registrants Only reservation type. If the flag does not appear, then the namespace does not support the Write Exclusive – Registrants Only reservation type.

**excres** Indicates that the namespace supports the Exclusive Access reservation type. If the flag does not appear, then the namespace does not support the Exclusive Access reservation type.

**wrtexcres** Indicates that the namespace supports the Write Exclusive reservation type. If the flag does not appear, then the namespace does not support the Write Exclusive reservation type.

**pwrl** Indicates that the namespace supports the Persist Through Power Loss capability. If the flag does not appear, then the namespace does not support the Persist Through Power Loss Capability.

Default:

Defaults to 0 (no flags set).

### eui64: Extended Unique Identifier

Format:

Eui64 <eui number>

The EUI a globally unique number for the namespace. It is a concatenation of the manufacturer number and an ID number for the namespace. The EUI is specified as a 64 bit value.

Default:

The EUI is automatically generated by taking the logical number of the namespace (which is automatically enumerated from the definition file from 1 to N), adding the euioff value, and merging with the manufacturer value. Thus, it can be normally left for the EUI value to be automatically generated in the file.

### Euioff

Format:

Euioff <euioff>

The EUI offset is used to generate EUI64 values for namespaces. See the eui64 statement for further information. The euioff number is specified.

### Euiman

Format:

Euiman <euiman>

The EUI manufacturer is used to generate EUI64 values for namespaces. See the eui64 statement for further information. The euiman number is specified.

Default:

The EUI is set by default to the manufacturer number for Cavium Networks, 0x000FB7.

### lbaf: LBA Format

Format:

Lbaf [‘\*’]<lbasize>[<perform>][<metadatasize>]

The LBA format gives up to 16 LBA format entries. The lbasize is the size of each lba (sector) given as a power of 2. It has a minimum value of 512. Typical values are 512, 1024, 2048, 4096, etc.

Perform is a performance ranking of this LBA entry as:

**Best** Best performance

**Better** Better performance

**Good** Good performance

**Degraded** Degraded Performance

The metadatasize is the total number of bytes provided per LBA.

If the LBA entry is the one used to format the drive, it is preceeded by “\*” (star character). If no LBA is so marked, the default will be the first LBA entered, or the default LBA.

The total set of LBA formats for a namespace is the concatenation, in order, of all of the lbaf statements in the namespace block.

If no LBA statement appears, the default declaration is:

Lbaf \*512,best,0

### vs: Vendor specific

Format:

**vs** [<datas>|string]…

This field is used to store up to 3712 bytes of vendor specific data. A series of either data bytes or string values can be used to fill the space. A string consists of a series of character values enclosed in single or double quotes. C style escapes are allowed.

If multiple vs statements appear in a namespace, they are concatenated.

Default:

Defaults to all zeros. Any part of the vendor space that is not defined is set to zero.

## Controller Assignments

Namespace to controller assignments appear outside of a namespace definition block. They have the format:

**assign** <controller> [<namespace>]…

Where the controller and namespace are logical controller and namespace numbers. Controllers are numbed from 0 to N. Namespaces are numbered from 1 to N. The controller number is 0 for PF, 1 for the first VF, etc. The maximum controller number is determined by implementation and/or controller setup registers. The namespace number is 1 for the first namespace block, 2 for the second, etc. It is an error if not that many namespaces appear in the definition file.

Multiple assignments to the same controller concatenate.

## RAM aliases

Namespace RAM aliases allow two namespaces to share the same RAM block, or a subset of that RAM block. The format is:

**alias** <dnamespace> <snamespace>

Both dnamespace and snamespace are logical namespace numbers. Controllers are numbed from 0 to N. Namespaces are numbered from 1 to N. The controller number is 0 for PF, 1 for the first VF, etc. The maximum controller number is determined by implementation and/or controller setup registers. The namespace number is 1 for the first namespace block, 2 for the second, etc. It is an error if not that many namespaces appear in the definition file.

The dnamespace gets the same RAM block as the snamespace. It is an error if the dnamespace RAM block is larger than the snamespace RAM block. It is an error to double alias a block, that is, an alias of an alias.

## Assignment Policy

The assignment policy controls how namespaces are mapped to controllers. The format is:

**policy** [ **onetoall** | **onetoone** | **pertable** ]

The meanings are:

**onetoall** Assign namespaces defined, to all controllers.

**onetoone** Assign each namespace in turn to a controller, i.e., controller 0 gets namespace 1, controller 1 gets namespace 2, etc.

**pertable** Use the assignment table, i.e., obey the **assign** statements.

Note: if the assignment policy is not **pertable**, none of the assign statements will have effect.

# Sample namespace specification file

The following is a typical namespace generation file.

namespace nsze 8m nameend

namespace nsze 1g nameend

Assign 0 1 2 ! give the pf namespaces 1 and 2

Assign 1 1 ! the first vf gets namespace 1

Alias 1 2 ! share the 8mb space with the 1gb space

Note the considerable use of default values here.