



P4 Language Support in DPL

Table of contents

Introduction

P4 Language Features

Identifiers

Data Types

Derived Types

Statements and Expressions

Operators

Variables

Control Apply Block

Actions

This section contains information on the P4 language features that are supported, known issues and deviations from the [P4 Language Specification](#). All references to the spec refer to P4₁₆ language specification version v1.2.4.

Introduction

The NVIDIA BlueField networking platform (DPU or SuperNIC) is a specialized Data Processing Unit engineered to significantly enhance data center performance. It achieves this by efficiently offloading, accelerating, and isolating critical tasks related to networking, storage, and security. BlueField-3 merges the advantages of dedicated accelerators with the versatility of general-purpose processors, all within an ASIC-based system-on-a-chip architecture. It boasts impressive connectivity speeds of up to 400G, making it an ideal choice for environments demanding high levels of AI and high-performance computing capabilities.

To ensure BlueField delivers on its promise of optimized functionality, its P4 language implementation has been tailored to leverage the strengths of high-performance ASIC hardware. This strategic focus on performance and efficiency, however, means that BlueField supports DOCA Pipeline Language (DPL), which does not include every feature outlined in the P4 language specification. This is primarily due to the inherent differences in flexibility and programmability between ASICs and other types of hardware, such as CPUs and FPGAs, and the unique pipeline model of the BlueField DPU. This section outlines the P4 Language features that are currently supported in this release of the NVIDIA DPL compiler.

Note

This document refers to features as being **supported** or **unsupported**. Supported features have been tested and should work according to the P4₁₆ language specification, subject to any caveats described in this document. Unsupported features have not been fully tested and should not be relied upon. In most cases, the compiler will reject programs that use unsupported features. However, in some cases the compiler may accept a program that uses unsupported features if the feature is not necessary to implement the program. For example, if a program contains an expression that uses an unsupported operator but its operands can be computed at compile-time, the compiler may choose to compute the value of the expression at compile-time and accept the program. This behavior

should not be used as an indicator of whether the feature is supported.

P4 Language Features

Identifiers

Identifiers starting with `__` are reserved for internal compiler use. Otherwise, identifiers described in the P4 language spec section *6.4.1. Identifiers* are allowed.

Data Types

See *Operators* section for support of operations on values with these types.

- Bool
 - Supported
- Arbitrary-precision Integer
 - Supported only for literals. See spec [7.1.6.5. Arbitrary-precision integers](#).
- Signed integer
 - Unsupported
- Strings literals
 - No operations are allowed or validity checks performed. See spec [6.4.3.3](#).
- Bit strings
 - Supported, limited by available hardware resources

Derived Types

- Enum

- Enumeration types are supported as described in section [7.2.1](#) of the P4₁₆ spec, allowing the P4 programmer to either specify an underlying representation or allow the compiler to choose the representation. Note that the set of allowed types for the underlying representation is limited to those otherwise supported by the DPL compiler.
- Header
 - Header types are supported as described in section [7.2.2](#) of the P4₁₆ spec using field types otherwise supported by the DPL compiler with the exception of `varbit<>` fields. Variable-length header types are supported only using the `NvOptionParser` extern type.
- Header stacks
 - Unsupported
- Structs
 - Struct types are supported as described in section [7.2.5](#) of the P4₁₆ spec using field types otherwise supported by the DPL compiler.
- Unions
 - Unsupported
- Tuple/List
 - Tuple types are supported as described in section [7.2.6](#) of the P4₁₆ spec using component types otherwise supported by the DPL compiler.
- Extern types
 - Extern types, including both extern functions and extern objects, as described in section [7.2.9](#) of the P4₁₆ spec are supported only for those declared in the P4 headers distributed with the DPL compiler. P4 programs cannot declare additional extern types.
- Type specialization
 - Type specialization is supported as described in section [7.2.10](#) of the P4₁₆ spec.

Statements and Expressions

- Assignment
 - An L-value cannot be used in a method call expression, packet out metadata, flex-header field, standard metadata field, or as an action parameter. Not all fixed header fields can be an L-value of an assignment statement. Please refer to the chart below.
- Conditional
 - Conditional statements are only supported within control apply blocks. Its expression must evaluate to a bit or bool type.
- switch statement
 - The switch statement is only supported within control apply block. Its expression must evaluate to a bool type.
 - The compiler supports empty switch statement, fall through, default case, and non-default cases. See spec section [11.7 Switch statement](#) for details.

The following tables describe the compiler support for expressions using the built in header fields and standard metadata as L-values and R-values. Note, this is separate of header fields that can be used as match keys.

Note

In the default hardware parser, some fields that are mutually exclusive are extracted to the same buffer location (referred to in the table as an alias). Assignments to and copy from these fields can use either of the aliased field names.

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.ethernet.dst_addr</code>	✓	✓	

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.ethernet.src_addr</code>	✓	✓	
<code>headers.ethernet.ether_type</code>	☐	☐	Last extracted outer <code>etherType</code> value
<code>headers.vlan.vlan_pcp</code>	☐	☐	
<code>headers.vlan.vlan_dei</code>	☐	☐	
<code>headers.vlan.vlan_id</code>	✓	✓	
<code>headers.vlan.vlan_ether_type</code>	☐	☐	Last extracted outer <code>etherType</code> value
<code>headers.inner_ethernet.dst_addr</code>	☐	✓	
<code>headers.inner_ethernet.src_addr</code>	☐	✓	
<code>headers.inner_ethernet.ether_type</code>	☐	☐	Last extracted inner <code>etherType</code> value
<code>headers.inner_vlan.vlan_pcp</code>	☐	☐	
<code>headers.inner_vlan.vlan_dei</code>	☐	☐	
<code>headers.inner_vlan.vlan_id</code>	☐	☐	
<code>headers.inner_vlan.vlan_ether_type</code>	☐	☐	Last extracted inner <code>etherType</code> value
<code>headers.ipv4.version</code> Alias with <code>headers.ipv6.version</code>	☐	☐	
<code>headers.ipv4.ihl</code>	☐	☐	
<code>headers.ipv4.diffserv</code> Alias with <code>headers.ipv6.diffserv</code>	☐	✓	Can be set through <code>nv_set_ip_dscp</code> extern

Fixed Header Fields	Assignab le	Copyabl e	Notes
headers.ipv4.ecn Alias with headers.ipv6.ecn	☐	✓	Can be set through nv_set_ip_ecn extern
headers.ipv4.total_len	☐	☐	Value is write only by hardware
headers.ipv4.identificati on	☐	☐	
headers.ipv4.flags	☐	☐	
headers.ipv4.frag_offset	☐	☐	
headers.ipv4.ttl Alias with headers.ipv6.hop_limit	☐	✓	Can be set through nv_set_ip_ttl extern
headers.ipv4.protocol Alias with headers.ipv6.next_header	☐	☐	
headers.ipv4.hdr_checksum	☐	☐	Value is write only by hardware
headers.ipv4.src_addr	✓	✓	
headers.ipv4.dst_addr	✓	✓	
headers.inner_ipv4.versio n Alias with headers.inner_ipv6.versio n	☐	☐	
headers.inner_ipv4.ihl	☐	☐	
headers.inner_ipv4.diffse rv Alias with headers.inner_ipv6.diffse rv	☐	✓	
headers.inner_ipv4.ecn Alias with headers.inner_ipv6.ecn	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.inner_ipv4.total_len</code>	☐	☐	
<code>headers.inner_ipv4.identification</code>	☐	☐	
<code>headers.inner_ipv4.flags</code>	☐	☐	
<code>headers.inner_ipv4.frag_offset</code>	☐	☐	
<code>headers.inner_ipv4.ttl</code> Alias with <code>headers.inner_ipv6.hop_limit</code>	☐	✓	
<code>headers.inner_ipv4.protocol</code> Alias with <code>headers.inner_ipv6.protocol</code>	☐	☐	
<code>headers.inner_ipv4.hdr_checksum</code>	☐	☐	
<code>headers.inner_ipv4.src_addr</code>	☐	✓	
<code>headers.inner_ipv4.dst_addr</code>	☐	✓	
<code>headers.ipv6.flow_label</code>	☐	☐	
<code>headers.ipv6.payload_length</code>	☐	☐	
<code>headers.ipv6.src_addr</code>	✓	✓	
<code>headers.ipv6.dst_addr</code>	✓	✓	
<code>headers.inner_ipv6.flow_label</code>	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.inner_ipv6.payload_length</code>	☐	☐	
<code>headers.inner_ipv6.src_addr</code>	☐	✓	
<code>headers.inner_ipv6.dst_addr</code>	☐	✓	
<code>headers.mpls.label</code>	✓	✓	
<code>headers.mpls.tc</code>	✓	✓	
<code>headers.mpls.bos</code>	✓	✓	
<code>headers.mpls.ttl</code>	✓	✓	
<code>headers.inner_mpls.label</code>	☐	✓	
<code>headers.inner_mpls.tc</code>	☐	✓	
<code>headers.inner_mpls.bos</code>	☐	✓	
<code>headers.inner_mpls.ttl</code>	☐	✓	
<code>headers.icmp.type</code>	☐	☐	
<code>headers.icmp.code</code>	☐	☐	
<code>headers.icmp.checksum</code>	☐	☐	
<code>headers.icmp.identifier</code>	☐	☐	
<code>headers.icmp.sequence_number</code>	☐	☐	
<code>headers.inner_icmp.type</code>	☐	☐	
<code>headers.inner_icmp.code</code>	☐	☐	
<code>headers.inner_icmp.checksum</code>	☐	☐	
<code>headers.inner_icmp.identifier</code>	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.inner_icmp.sequence_number</code>	☐	☐	
<code>headers.icmpv6.type</code>	☐	☐	
<code>headers.icmpv6.code</code>	☐	☐	
<code>headers.icmpv6.checksum</code>	☐	☐	
<code>headers.icmpv6.payload_1</code>	☐	☐	
<code>headers.icmpv6.payload_2</code>	☐	☐	
<code>headers.inner_icmpv6.type</code>	☐	☐	
<code>headers.inner_icmpv6.code</code>	☐	☐	
<code>headers.inner_icmpv6.checksum</code>	☐	☐	
<code>headers.inner_icmpv6.payload_1</code>	☐	☐	
<code>headers.inner_icmpv6.payload_2</code>	☐	☐	
<code>headers.tcp.src_port</code> Alias with <code>headers.udp.src_port</code>	☐	✓	Can be set through <code>nv_set_l4_src_port</code> extern
<code>headers.tcp.dst_port</code> Alias with <code>headers.udp.dst_port</code>	☐	✓	Can be set through <code>nv_set_l4_dst_port</code> extern
<code>headers.tcp.seq_no</code>	✓	✓	
<code>headers.tcp.ack_no</code>	✓	✓	
<code>headers.tcp.data_offset</code>	☐	☐	
<code>headers.tcp.res</code>	☐	☐	
<code>headers.tcp.nonce_sum</code>	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
headers.tcp.ecn	✓	✓	
headers.tcp.flags	✓	✓	
headers.tcp.window	☐	☐	
headers.tcp.checksum	☐	☐	
headers.tcp.urgent_ptr	☐	☐	
headers.inner_tcp.src_port Alias with headers.inner_udp.src_port	☐	✓	
headers.inner_tcp.dst_port Alias with headers.inner_udp.dst_port	☐	✓	
headers.inner_tcp.seq_no	☐	✓	
headers.inner_tcp.ack_no	☐	✓	
headers.inner_tcp.data_of fset	☐	☐	
headers.inner_tcp.res	☐	☐	
headers.inner_tcp.nonce_s um	☐	☐	
headers.inner_tcp.ecn	☐	✓	
headers.inner_tcp.flags	☐	✓	
headers.inner_tcp.window	☐	☐	
headers.inner_tcp.checksu m	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
<code>headers.inner_tcp.urgent_ptr</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.udp.length</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.udp.checksum</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.inner_udp.length</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.inner_udp.checksum</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.checksum_present</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.reserved1</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.key_present</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.sequence_present</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.reserved2</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.reserved3</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.version</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.gre.protocol</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.nvgre_vsid.vsid</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.nvgre_vsid.flow_id</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.esp.security_parameters.index</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.esp.sequence_number</code>	<input type="checkbox"/>	<input type="checkbox"/>	
<code>headers.esp.next_header</code>	<input type="checkbox"/>	<input type="checkbox"/>	Value set by hardware after decryption
<code>headers.psp.next_header</code>	<input type="checkbox"/>	<input type="checkbox"/>	Value set by hardware after

Fixed Header Fields	Assignab le	Copyabl e	Notes
			decryption
headers.psp.hdr_ext_len	☐	☐	
headers.psp.crypt_offset	☐	☐	
headers.psp.needs_samplin g	☐	☐	
headers.psp.drop	☐	☐	
headers.psp.version	☐	☐	
headers.psp.has_virtualiz ation_key	☐	☐	
headers.psp.one_1	☐	☐	
headers.psp.security_para meters_index	☐	☐	
headers.psp.initializatio n_vector	☐	☐	
headers.psp.virtualizatio n_key_high	☐	☐	
headers.psp.virtualizatio n_key_low	☐	☐	
headers.vxlan.reserved1	☐	☐	
headers.vxlan.vni_valid	☐	☐	
headers.vxlan.reserved2	☐	☐	
headers.vxlan.reserved3	☐	☐	
headers.vxlan.vni	✓	✓	
headers.vxlan.reserved4	☐	☐	
headers.vxlan_gpe.reserve d1	☐	☐	

Fixed Header Fields	Assignab le	Copyabl e	Notes
headers.vxlan_gpe.vni_val id	☐	☐	
headers.vxlan_gpe.reserve d2	☐	☐	
headers.vxlan_gpe.reserve d3	☐	☐	
headers.vxlan_gpe.next_pr oto	☐	☐	
headers.vxlan_gpe.vni	✓	✓	
headers.vxlan_gpe.reserve d4	☐	☐	
headers.geneve.ver	☐	☐	
headers.geneve.opt_len	☐	☐	
headers.geneve.o	☐	☐	
headers.geneve.c	☐	☐	
headers.geneve.reserved1	☐	☐	
headers.geneve.protocol_t ype	☐	☐	
headers.geneve.vni	✓	✓	
headers.geneve.reserved2	☐	☐	

All the fields of BlueField standard metadata are read only. The following table outlines the current support for using a standard metadata field as an R-value in an expression.

Standard Metadata Fields	Copyab le	Notes
ingress_port	✓	
eth_to_fcs_packet_ len	☐	

Standard Metadata Fields	Copyable	Notes
is_l2_ok	<input type="checkbox"/>	
l2_type	<input type="checkbox"/>	
last_l2_ether_type	<input checked="" type="checkbox"/>	Last extracted value of etherType within ethernet header or VLAN tags
vlan_type	<input type="checkbox"/>	
is_l3_ok	<input type="checkbox"/>	
l3_type	<input type="checkbox"/>	
is_ip_fragmented	<input type="checkbox"/>	
is_ipv4_checksum_ok	<input type="checkbox"/>	
is_l4_ok	<input type="checkbox"/>	
l4_type	<input type="checkbox"/>	
l4_type_ext	<input type="checkbox"/>	
is_l4_checksum_ok	<input type="checkbox"/>	
l4_src_port	<input type="checkbox"/>	
l4_dst_port	<input type="checkbox"/>	
encap_type	<input type="checkbox"/>	ROCE not currently supported
ipsec_layer	<input type="checkbox"/>	
ipsec_syndrome	<input type="checkbox"/>	Valid only after hardware encrypt/decrypt
psp_syndrome	<input type="checkbox"/>	Valid only after hardware encrypt/decrypt
is_inner_l2_ok	<input type="checkbox"/>	
inner_l2_type	<input type="checkbox"/>	
inner_last_l2_ether_type	<input checked="" type="checkbox"/>	Last extracted value of etherType within inner ethernet header or VLAN tags
inner_vlan_type	<input type="checkbox"/>	

Standard Metadata Fields	Copyable	Notes
is_inner_l3_ok	☐	
inner_l3_type	☐	
is_inner_ipv4_checksum_ok	☐	
is_inner_l4_ok	☐	
inner_l4_type	☐	
inner_l4_type_ext	☐	
random_value	☐	
ut_clock	☐	
fr_clock	☐	
source_qp	☐	

Operators

The P4-16 language specification lists a wide variety of operations that the language accepts for the supported data types (see [Section 8](#)). The table below lists the operators that are officially supported by the NVIDIA P4 compiler:

Operator	Compile-time value	P4Runtime value	Runtime value	Spec section
Bool && Bool	✓	✓	✓	8.5
Bool Bool	✓	✓	✓	8.5
Bool == Bool	✓	✓ <u>1</u>	✓ <u>1</u>	8.5
Bool != Bool	✓	✓ <u>1</u>	✓ <u>1</u>	8.5
Bit<W> == Bit<W>	✓	✓ <u>1</u>	✓ <u>1</u>	8.6
Bit<W> != Bit<W>	✓	✓ <u>1</u>	✓ <u>1</u>	8.6

Operator	Compile-time value	P4Runtime value	Runtime value	Spec section
Bit<W> << integer	✓ <u>2</u>	✓ <u>2</u>	✓ <u>2</u>	8.6
Bit<W> >> integer	✓ <u>2</u>	✓ <u>2</u>	✓ <u>2</u>	8.6
Bit<W>[H:L]	✓ <u>3</u>	✓ <u>3</u>	✓ <u>3</u>	8.6
All explicit casts between supported types	✓	✓	✓	8.11.1
All implicit casts between supported types	✓	✓	✓	8.11.2
Bit<W>..Bit<W>	✓ <u>4</u>	✓ <u>4</u>	☐	8.15.4
Assignment to user struct fields	✓	✓	✓	8.16
Assignment to packet-in struct fields	✓	✓	✓	8.16
All operations on header fields	✓	✓	✓ <u>5</u>	8.17
Method calls	✓	✓	☐	8.20
Function calls with positional args	✓	✓ <u>6</u>	✓ <u>6</u>	8.20
Extern constructor invocations	✓	☐	☐	8.21
Parser constructor invocations	✓	☐	☐	8.21
Control constructor invocations	✓	☐	☐	8.21
Package constructor invocations	✓	☐	☐	8.21

1. LHS or RHS must be compile-time constant [_____](#)
2. RHS must be compile-time constant. See *spec 8.9.2* [_____](#)
3. H and L are subject to the restrictions described in the *spec 8.6*. Assigning to slices (slices as L-values) is not supported. Additionally, slices as R-values are only supported as P4 table keys. [_____](#)
4. Only valid in P4 Table entries [_____](#)
5. Limited to those fields that can be a copy source [_____](#)
6. Limited on a per extern function basis [_____](#)

Variables

Variables are supported in accordance with the following spec items:

- Constants (*spec 11.1*)
 - "Compile-time known values" are evaluated on a best-effort basis. It is possible that a compile-time known value may not be recognized by the compiler as such.
- Variables (*spec 11.2*)
- Instantiations (*spec 11.3*)
 - Instantiations with abstract methods (*spec 11.3.1*) are allowed in BlueField Target Architecture
 - Named arguments are not supported

Variables may be declared in any of the locations described in (*spec 11.2*) and follow the scope rules described there.

The compiler will emit errors for uninitialized values. In some cases where a struct is partially initialized, only a warning may be produced. In some cases there may be no error emitted when an uninitialized struct field is accessed. The accessed field will then contain an undefined value.

Control Apply Block

The following statements are supported in a control's apply block:

- `table.apply()` calls
- `if` statement
- `switch` statement
- extern function and method calls
- assignment statements with the supported operators
- the empty statement
- `return` statements

The `exit` statement is not supported.

All supported expressions are allowed within these statements, where applicable.

Actions

Actions support the same statements as controls except for the following:

- `table.apply()` calls
- Conditional statements - `if` and `switch`

Actions support the same expressions as controls except for the following:

- Boolean logical operators - `&&`, `||`, ternary operator
- Comparisons (`==`, `!=`, etc.)

Notice
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no

representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document. NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk. NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs. No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices. THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product. **Trademarks** NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2025, NVIDIA. PDF Generated on 04/24/2025