



# libNVVM API v4.0

## API Reference Manual

# Table of Contents

Chapter 1. Introduction.....	1
Chapter 2. Thread Safety.....	2
Chapter 3. Modules.....	3
3.1. Error Handling.....	3
nvmResult.....	3
nvmGetErrorString.....	3
3.2. General Information Query.....	4
nvmIRVersion.....	4
nvmVersion.....	4
3.3. Compilation.....	4
nvmProgram.....	5
nvmAddModuleToProgram.....	5
nvmCompileProgram.....	6
nvmCreateProgram.....	8
nvmDestroyProgram.....	8
nvmGetCompiledResult.....	9
nvmGetCompiledResultSize.....	9
nvmGetProgramLog.....	10
nvmGetProgramLogSize.....	10
nvmLazyAddModuleToProgram.....	11
nvmVerifyProgram.....	12

---

# Chapter 1. Introduction

libNVM API v4.0 provides an interface for generating PTX code from both binary and text NVVM IR inputs. Compatible input can be generated by tools and libraries that produce LLVM 7.0 IR and bitcode. Support for reading the text NVVM IR representation is deprecated and may be removed in a later release.

---

## Chapter 2. Thread Safety

Usage of libNVM, via the libNVM API, is intended to be thread-safe; however, users should expect serial execution performance.

---

# Chapter 3. Modules

Here is a list of all modules:

- ▶ [Error Handling](#)
- ▶ [General Information Query](#)
- ▶ [Compilation](#)

## 3.1. Error Handling

### enum nvvmResult

NVVM API call result code.

#### Values

```
NVVM_SUCCESS = 0  
NVVM_ERROR_OUT_OF_MEMORY = 1  
NVVM_ERROR_PROGRAM_CREATION_FAILURE = 2  
NVVM_ERROR_IR_VERSION_MISMATCH = 3  
NVVM_ERROR_INVALID_INPUT = 4  
NVVM_ERROR_INVALID_PROGRAM = 5  
NVVM_ERROR_INVALID_IR = 6  
NVVM_ERROR_INVALID_OPTION = 7  
NVVM_ERROR_NO_MODULE_IN_PROGRAM = 8  
NVVM_ERROR_COMPILATION = 9
```

### const char \*nvvmGetErrorString (nvvmResult result)

Get the message string for the given nvvmResult code.

#### Parameters

##### result

NVVM API result code.

## Returns

Message string for the given [nvvmResult](#) code.

## 3.2. General Information Query

### `nvvmResult nvvmIRVersion (int *majorIR, int *minorIR, int *majorDbg, int *minorDbg)`

Get the NVVM IR version.

#### Parameters

**majorIR**

NVVM IR major version number.

**minorIR**

NVVM IR minor version number.

**majorDbg**

NVVM IR debug metadata major version number.

**minorDbg**

NVVM IR debug metadata minor version number.

#### Returns

- ▶ `NVVM_SUCCESS`

### `nvvmResult nvvmVersion (int *major, int *minor)`

Get the NVVM version.

#### Parameters

**major**

NVVM major version number.

**minor**

NVVM minor version number.

#### Returns

- ▶ `NVVM_SUCCESS`

## 3.3. Compilation

```
typedef _nvvmProgram *nvvmProgram
```

NVVM Program.

An opaque handle for a program.

```
nvvmResult nvvmAddModuleToProgram
(nvvmProgram prog, const char *buffer, size_t size,
const char *name)
```

Add a module level NVVM IR to a program.

### Parameters

**prog**

NVVM program.

**buffer**

NVVM IR module in the bitcode or text representation.

**size**

Size of the NVVM IR module.

**name**

Name of the NVVM IR module. If NULL, "<unnamed>" is used as the name.

### Returns

- ▶ NVVM\_SUCCESS
- ▶ NVVM\_ERROR\_OUT\_OF\_MEMORY
- ▶ NVVM\_ERROR\_INVALID\_INPUT
- ▶ NVVM\_ERROR\_INVALID\_PROGRAM

### Description

The `buffer` should contain an NVVM IR module. The module should have NVVM IR either in the LLVM 7.0.1 bitcode representation or in the LLVM 7.0.1 text representation. Support for reading the text representation of NVVM IR is deprecated and may be removed in a later version.

## `nvvmResult nvvmCompileProgram (nvvmProgram prog, int numOptions, const char **options)`

Compile the NVVM program.

### Parameters

#### **prog**

NVVM program.

#### **numOptions**

Number of compiler options passed.

#### **options**

Compiler options in the form of C string array.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_OUT_OF_MEMORY`
- ▶ `NVVM_ERROR_IR_VERSION_MISMATCH`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`
- ▶ `NVVM_ERROR_INVALID_OPTION`
- ▶ `NVVM_ERROR_NO_MODULE_IN_PROGRAM`
- ▶ `NVVM_ERROR_COMPILATION`

### Description

The NVVM IR modules in the program will be linked at the IR level. The linked IR program is compiled to PTX.

The target datalayout in the linked IR program is used to determine the address size (32bit vs 64bit).

The valid compiler options are:

- ▶ `-g` (enable generation of full debugging information). Full debug support is only valid with `'-opt=0'`. Debug support requires the input module to utilize NVVM IR Debug Metadata. Line number (line info) only generation is also enabled via NVVM IR Debug Metadata, there is no specific libNVVM API flag for that case.
- ▶ `-opt=`
  - ▶ `0` (disable optimizations)
  - ▶ `3` (default, enable optimizations)

- ▶ -arch=
  - ▶ compute\_50
  - ▶ compute\_52 (default)
  - ▶ compute\_53
  - ▶ compute\_60
  - ▶ compute\_61
  - ▶ compute\_62
  - ▶ compute\_70
  - ▶ compute\_72
  - ▶ compute\_75
  - ▶ compute\_80
  - ▶ compute\_87
  - ▶ compute\_89
  - ▶ compute\_90
- ▶ -ftz=
  - ▶ 0 (default, preserve denormal values, when performing single-precision floating-point operations)
  - ▶ 1 (flush denormal values to zero, when performing single-precision floating-point operations)
- ▶ -prec-sqrt=
  - ▶ 0 (use a faster approximation for single-precision floating-point square root)
  - ▶ 1 (default, use IEEE round-to-nearest mode for single-precision floating-point square root)
- ▶ -prec-div=
  - ▶ 0 (use a faster approximation for single-precision floating-point division and reciprocals)
  - ▶ 1 (default, use IEEE round-to-nearest mode for single-precision floating-point division and reciprocals)
- ▶ -fma=
  - ▶ 0 (disable FMA contraction)
  - ▶ 1 (default, enable FMA contraction)

## `nvvmResult nvvmCreateProgram (nvvmProgram *prog)`

Create a program, and set the value of its handle to `*prog`.

### Parameters

**prog**

NVVM program.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_OUT_OF_MEMORY`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

**See also:**

[`nvvmDestroyProgram\(\)`](#)

## `nvvmResult nvvmDestroyProgram (nvvmProgram *prog)`

Destroy a program.

### Parameters

**prog**

NVVM program.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

**See also:**

[`nvvmCreateProgram\(\)`](#)

## `nvvmResult nvvmGetCompiledResult (nvvmProgram prog, char *buffer)`

Get the compiled result.

### Parameters

**prog**

NVVM program.

**buffer**

Compiled result.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

The result is stored in the memory pointed to by `buffer`.

## `nvvmResult nvvmGetCompiledResultSize (nvvmProgram prog, size_t *bufferSizeRet)`

Get the size of the compiled result.

### Parameters

**prog**

NVVM program.

**bufferSizeRet**

Size of the compiled result (including the trailing NULL).

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

## `nvvmResult nvvmGetProgramLog (nvvmProgram prog, char *buffer)`

Get the Compiler/Verifier Message.

### Parameters

**prog**

NVVM program.

**buffer**

Compilation/Verification log.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

The NULL terminated message string is stored in the memory pointed to by `buffer` when the return value is `NVVM_SUCCESS`.

## `nvvmResult nvvmGetProgramLogSize (nvvmProgram prog, size_t *bufferSizeRet)`

Get the Size of Compiler/Verifier Message.

### Parameters

**prog**

NVVM program.

**bufferSizeRet**

Size of the compilation/verification log (including the trailing NULL).

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

The size of the message string (including the trailing NULL) is stored into `bufferSizeRet` when the return value is `NVVM_SUCCESS`.

## `nvvmResult nvvmLazyAddModuleToProgram` (`nvvmProgram prog`, `const char *buffer`, `size_t size`, `const char *name`)

Add a module level NVVM IR to a program.

### Parameters

#### **prog**

NVVM program.

#### **buffer**

NVVM IR module in the bitcode representation.

#### **size**

Size of the NVVM IR module.

#### **name**

Name of the NVVM IR module. If NULL, "<unnamed>" is used as the name.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_OUT_OF_MEMORY`
- ▶ `NVVM_ERROR_INVALID_INPUT`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`

### Description

The `buffer` should contain an NVVM IR module. The module should have NVVM IR in the LLVM 7.0.1 bitcode representation.

A module added using this API is lazily loaded - the only symbols loaded are those that are required by module(s) loaded using `nvvmAddModuleToProgram`. It is an error for a program to have all modules loaded using this API. Compiler may also optimize entities in this module by making them internal to the linked NVVM IR module, making them eligible for other optimizations. Due to these optimizations, this API to load a module is more efficient and should be used where possible.

## `nvvmResult nvvmVerifyProgram (nvvmProgram prog, int numOptions, const char **options)`

Verify the NVVM program.

### Parameters

**prog**

NVVM program.

**numOptions**

Number of compiler options passed.

**options**

Compiler options in the form of C string array.

### Returns

- ▶ `NVVM_SUCCESS`
- ▶ `NVVM_ERROR_OUT_OF_MEMORY`
- ▶ `NVVM_ERROR_IR_VERSION_MISMATCH`
- ▶ `NVVM_ERROR_INVALID_PROGRAM`
- ▶ `NVVM_ERROR_INVALID_IR`
- ▶ `NVVM_ERROR_INVALID_OPTION`
- ▶ `NVVM_ERROR_NO_MODULE_IN_PROGRAM`

### Description

The valid compiler options are:

Same as for [nvvmCompileProgram\(\)](#).

**See also:**

[nvvmCompileProgram\(\)](#)

## 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.

## OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

## Trademarks

NVIDIA and the NVIDIA logo are trademarks 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

© 2007-2022 NVIDIA Corporation & affiliates. All rights reserved.