# clCompileProgram

Compiles a program’s source for all the devices or a specific device(s) in the OpenCL context associated with *program*.

```
cl_int
clCompileProgram (   cl_program program,
                     cl_uint num_devices,
                     const cl_device_id *device_list,
                     const char *options,
                     cl_uint num_input_headers,
                     const cl_program *input_headers,
                     const char **header_include_names,
                     void (CL_CALLBACK *pfn_notify)( cl_program
                     program, void *user_data),
                     void *user_data)
```

## Parameters

*program*

The program object that is the compilation target.

*device_list*

A pointer to a list of devices associated with *program*. If *device_list* is a NULL value, the compile is performed for all devices associated with program. If *device_list* is a non-NULL value, the compile is performed for devices specified in this list.

*num_devices*

The number of devices listed in *device_list*.

*options*

A pointer to a null-terminated string of characters that describes the compilation options to be used for building the program executable. Certain options are ignored when program is created with IL. The list of supported options is as described below.

*num_input_headers*

Specifies the number of programs that describe headers in the array referenced by *input_headers*.

*input_headers*

An array of program embedded headers created with clCreateProgramWithSource.

*header_include_names*

An array that has a one to one correspondence with *input_headers*. Each entry in *header_include_names* specifies the include name used by source in *program* that comes from an embedded header. The corresponding entry in *input_headers* identifies the program object which contains the header source to be used. The embedded headers are first searched before the headers in the list of directories specified by the –I compile option (as described in section 5.8.4.1). If multiple entries in *header_include_names* refer to the same header name, the first one encountered will be used.

*pfn_notify*

A function pointer to a notification routine. The notification routine is a callback function that an application can register and which will be called when the program executable has been built (successfully or unsuccessfully). If *pfn_notify* is not NULL, **clCompileProgram** does not need to wait for the compiler to complete and can return immediately once the compilation can begin. The compilation can begin if the context, program whose sources are being compiled, list of devices, input headers, programs that describe input headers and compiler options specified are all valid and appropriate host and device resources needed to perform the compile are available. If *pfn_notify* is NULL, **clCompileProgram** does not return until the compiler has completed. This callback function may be called asynchronously by the OpenCL implementation. It is the application's responsibility to ensure that the callback function is thread-safe.

*user_data*

Passed as an argument when *pfn_notify* is called. *user_data* can be NULL.

## Notes

Compiles a program's source for all the devices or a specific device(s) in the OpenCL context associated with *program*. The pre-processor runs before the program sources are compiled. The compiled binary is built for all devices associated with *program* or the list of devices specified. The compiled binary can be queried using clGetProgramInfo(*program*, `CL_PROGRAM_BINARIES`, ...)

and can be specified to clCreateProgramWithBinary to create a new program object.

## Compiler Options

The compiler options are categorized as pre-processor options, options for math intrinsics, options that control optimization and miscellaneous options. This specification defines a standard set of options that must be supported by the OpenCL C compiler when building program executables online or offline. These may be extended by a set of vendor- or platform specific options.

### Preprocessor Options

These options control the OpenCL C preprocessor which is run on each program source before actual compilation.

-D options are processed in the order they are given in the *options* argument to **clBuildProgram** or **clCompileProgram**.

*-D name*

Predefine *name* as a macro, with definition 1.

*-D name=definition*

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a '#define' directive. In particular, the definition will be truncated by embedded newline characters.

*-I dir*

Add the directory *dir* to the list of directories to be searched for header files.

### Math Intrinsics Options

These options control compiler behavior regarding floating-point arithmetic. These options trade off between speed and correctness.

*-cl-single-precision-constant*

Treat double precision floating-point constant as single precision constant.

*-cl-denorms-are-zero*

This option controls how single precision and double precision denormalized numbers are handled. If specified as a build option, the single precision denormalized numbers may be flushed to zero; double precision denormalized numbers may also be flushed to zero if the optional extension for double precsion is supported. This is intended to be a performance hint and the OpenCL compiler can choose not to flush denorms to zero if the device supports single precision (or double precision) denormalized numbers.

This option is ignored for single precision numbers if the device does not support single precision denormalized numbers i.e. `CL_FP_DENORM` bit is not set in `CL_DEVICE_SINGLE_FP_CONFIG`.

This option is ignored for double precision numbers if the device does not support double precision or if it does support double precison but not double precision denormalized numbers i.e. `CL_FP_DENORM` bit is not set in `CL_DEVICE_DOUBLE_FP_CONFIG`.

This flag only applies for scalar and vector single precision floating-point variables and computations on these floating-point variables inside a program. It does not apply to reading from or writing to image objects.

*-cl-fp32-correctly-rounded-divide-sqrt*

The `-cl-fp32-correctly-rounded-divide-sqrt` build option to **clBuildProgram** or **clCompileProgram** allows an application to specify that single precision floating-point divide (x/y and 1/x) and sqrt used in the program source are correctly rounded. If this build option is not specified, the minimum numerical accuracy of single precision floating-point divide and sqrt are as defined in section 7.4 of the OpenCL specification.

This build option can only be specified if the `CL_FP_CORRECTLY_ROUNDED_DIVIDE_SQRT` is set in `CL_DEVICE_SINGLE_FP_CONFIG` (as defined in in the table of allowed values for *param_name* for clGetDeviceInfo) for devices that the program is being build. **clBuildProgram** or **clCompileProgram** will fail to compile the program for a device if the `-cl-fp32-correctly-rounded-divide-sqrt` option is specified and `CL_FP_CORRECTLY_ROUNDED_DIVIDE_SQRT` is not set for the device.

## Optimization Options

These options control various sorts of optimizations. Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

*-cl-opt-disable*

This option disables all optimizations. The default is optimizations are enabled.

The following options control compiler behavior regarding floating-point arithmetic. These options trade off between performance and correctness and must be specifically enabled. These options are not turned on by default since it can result in incorrect output for programs which depend on an exact implementation of IEEE 754 rules/specifications for math functions.

*-cl-mad-enable*

Allow `a * b + c` to be replaced by a `mad`. The `mad` computes `a * b + c` with reduced accuracy. For example, some OpenCL devices implement `mad` as truncate the result of `a * b` before adding it to `c`.

*-cl-no-signed-zeros*

Allow optimizations for floating-point arithmetic that ignore the signedness of zero. IEEE 754 arithmetic specifies the distinct behavior of `+0.0` and `-0.0` values, which then prohibits simplification of expressions such as `x+0.0` or `0.0*x` (even with `-clfinite-math` only). This option implies that the sign of a zero result isn't significant.

*-cl-unsafe-math-optimizations*

Allow optimizations for floating-point arithmetic that (a) assume that arguments and results are valid, (b) may violate IEEE 754 standard and (c) may violate the OpenCL numerical compliance requirements as defined in section 7.4 for single precision and double precision floating-point, and edge case behavior in section 7.5. This option includes the `-cl-no-signed-zeros` and `-cl-mad-enable` options.

*-cl-finite-math-only*

Allow optimizations for floating-point arithmetic that assume that arguments and results are not NaNs or ±∞. This option may violate the OpenCL numerical compliance requirements defined in section 7.4 for single precision and double precision floating point, and edge case behavior in section 7.5.

*-cl-fast-relaxed-math*

Sets the optimization options `-cl-finite-math-only` and `-cl-unsafe-math-optimizations`. This allows optimizations for floating-point arithmetic that may violate the IEEE 754 standard and the OpenCL numerical compliance requirements defined in the specification in section 7.4 for single-precision and double precision floating-point, and edge case behavior in section 7.5. This option also relaxes the precision of commonly used math functions (refer to table 7.2 defined in section 7.4). This option causes the preprocessor macro `__FAST_RELAXED_MATH__` to be defined in the OpenCL program.

*-cl-uniform-work-group-size*

This requires that the global work-size be a multiple of the work-group size specified to clEnqueueNDRangeKernel. Allow optimizations that are made possible by this restriction.

**Options to Request or Suppress Warnings**

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error. The following language independent options do not enable specific warnings but control the kinds of diagnostics produced by the OpenCL compiler.

*-w*

Inhibit all warning messages.

*-Werror*

Make all warnings into errors.

**Options Controlling the OpenCL C Version**

The following option controls the version of OpenCL C that the compiler accepts.

*-cl-std=*

Determine the OpenCL C language version to use. A value for this option must be provided. Valid values are:

`CL1.1` - Support all OpenCL C programs that use the OpenCL C language features defined in section 6 of the OpenCL 1.1 specification.

`CL1.2` - Support all OpenCL C programs that use the OpenCL C language features defined in section 6 of the OpenCL 1.2 specification.

`CL2.0` - Support all OpenCL C programs that use the OpenCL C language features defined in section 6 of the OpenCL 2.0 specification.

Calls to **clBuildProgram** or **clCompileProgram** with the `-cl-std=CL1.1` option will fail to compile the program for any devices with `CL_DEVICE_OPENCL_C_VERSION` = OpenCL C 1.0.

Calls to **clBuildProgram** or **clCompileProgram** with the `-cl-std=CL1.2` option will fail to compile the program for any devices with `CL_DEVICE_OPENCL_C_VERSION` = OpenCL C 1.0 or OpenCL C 1.1.

Calls to **clBuildProgram** or **clCompileProgram** with the `-cl-std=CL2.0` option will fail to compile the program for any devices with `CL_DEVICE_OPENCL_C_VERSION` = OpenCL C 1.0, OpenCL C 1.1, or OpenCL C 1.2.

If the `–cl-std` build option is not specified, the highest OpenCL C 1.x language version supported by each device is used when compiling the program for each device. Applications are required to specify the `–cl-std=CL2.0` option if they want to compile or build their programs with OpenCL C 2.0.

### Options enabled by the cl_khr_spir extension

*-x spir*

Indicates that the binary is in SPIR format.

*-spir-std*

Specifies the version of the SPIR specification that describes the format and meaning of the binary. For example, if the binary is as described in SPIR version 1.2, then `-spir-std=1.2` must be specified. Failing to specify these compile options may result in implementation defined behavior.

### Options for Querying Kernel Argument Information

*-cl-kernel-arg-info*

This option allows the compiler to store information about the arguments of a kernel(s) in the program executable. The argument information stored includes the argument name, its type, the address and access qualifiers used. Refer to description of clGetKernelArgInfo on how to query this information.

### Options for debugging your program

*-g*

This option can currently be used to generate additional errors for the built-in functions that allow you to enqueue commands on a device (refer to section 6.13.17).

### Linker Options

This specification defines a standard set of linker options that must be supported by the OpenCL C compiler when linking compiled programs online or offline. These linker options are categorized as library linking options and program linking options. These may be extended by a set of vendor- or platform-specific options.

### Library Linking Options

The following options can be specified when creating a library of compiled binaries.

*-create-library*

Create a library of compiled binaries specified in *input_programs* argument to clLinkProgram.

*-enable-link-options*

Allows the linker to modify the library behavior based on one or more link options (described in Program Linking Options, below) when this library is linked with a program executable. This option must be specified with the `-create-library` option.

### Program Linking Options

The following options can be specified when linking a program executable.

- `-cl-denorms-are-zero`
- `-cl-no-signed-zeroes`
- `-cl-unsafe-math-optimizations`
- `-cl-finite-math-only`
- `-cl-fast-relaxed-mat`

The linker may apply these options to all compiled program objects specified to clLinkProgram. The linker may apply these options only to libraries which were created with the `-enable-link-option`.

### Separate Compilation and Linking of Programs

OpenCL programs are compiled and linked to support the following:

- Separate compilation and link stages. Program sources can be compiled to generate a compiled binary object and linked in a separate stage with other compiled program objects to the program exectuable.

- Embedded headers. In OpenCL 1.0 and 1.1, the `-I` build option could be used to specify the list of directories to be searched for headers files that are included by a program source(s). OpenCL 1.2 extends this by allowing the header sources to come from program objects instead of just header files.

- Libraries. The linker can be used to link compiled objects and libraries into a program executable or to create a library of compiled binaries.

## Errors

**clCompileProgram** returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_PROGRAM if *program* is not a valid program object.

- CL_INVALID_VALUE if *device_list* is NULL and *num_devices* is greater than zero, or if *device_list* is not NULL and *num_devices* is zero.

- CL_INVALID_VALUE if *num_input_headers* is zero and *header_include_names* or *input_headers* are not NULL or if *num_input_headers* is not zero and *header_include_names* or *input_headers* are NULL.

- CL_INVALID_VALUE if *pfn_notify* is NULL but *user_data* is not NULL.

- CL_INVALID_DEVICE if OpenCL devices listed in *device_list* are not in the list of devices associated with *program*.

- CL_INVALID_COMPILER_OPTIONS if the compiler options specified by *options* are invalid.

- CL_INVALID_OPERATION if the compilation or build of a program executable for any of the devices listed in *device_list* by a previous call to **clCompileProgram** or **clBuildProgram** for *program* has not completed.

- CL_COMPILER_NOT_AVAILABLE if a compiler is not available i.e. `CL_DEVICE_COMPILER_AVAILABLE` specified in in the table of allowed values for *param_name* for clGetDeviceInfo is set to `CL_FALSE`.

- CL_COMPILE_PROGRAM_FAILURE if there is a failure to compile the program source. This error will be returned if **clCompileProgram** does not return until the compile has completed.

- CL_INVALID_OPERATION if there are kernel objects attached to *program*.
- CL_INVALID_OPERATION if *program* has no source or IL available, i.e. it has not been created with clCreateProgramWithSource or clCreateProgramWithIL.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

## Example

For example, consider the following program source:

```
#include <foo.h>
#include <mydir/myinc.h>

__kernel void
image_filter (int n, int m,
            __constant float *filter_weights,
            __read_only image2d_t src_image,
            __write_only image2d_t dst_image)
{
...
}
```

This kernel includes two headers foo.h and mydir/myinc.h. The following describes how these headers can be passed as embedded headers in program objects:

```
cl_program foo_pg = clCreateProgramWithSource(context,
                           1, &foo_header_src, NULL, &err);
cl_program myinc_pg = clCreateProgramWithSource(context,
                           1, &myinc_header_src, NULL, &err);

// let’s assume the program source described above is given
// by program_A and is loaded via clCreateProgramWithSource

cl_program input_headers[2] = { foo_pg, myinc_pg };
char * input_header_names[2] = { “foo.h”, “mydir/myinc.h” };
```

```
clCompileProgram(program_A,
                 0, NULL, // num_devices & device_list
                 NULL,    // compile_options
                 2,       // num_input_headers
                 input_headers,
                 input_header_names,
                 NULL, NULL); // pfn_notify & user_data
```

## Specification

OpenCL Specification

## Also see

clGetDeviceInfo, clLinkProgram, clBuildProgram, clCreateProgramWithBinary, clCreateProgramWithSource, clCreateProgramWithIL