<https://code.google.com/archive/p/javacl/wikis/FAQ.wiki>

# javacl - FAQ.wiki

## What is JavaCL / ScalaCL / OpenCL4Java ?

JavaCL is an API that wraps the [OpenCL](http://www.khronos.org/opencl/) library to make it available to the Java platform ([OpenCL](http://www.khronos.org/opencl/) is the first open, royalty-free standard for cross-platform, parallel programming of modern processors and GPUs).

**With JavaCL, Java programs can execute tasks directly on graphic cards and benefit from their massive parallel horsepower**.

JavaCL comprises the following parts : \* a low-level API that matches 1 to 1 the C OpenCL API : OpenCL4Java. **Its direct use is deprecated** \* a nice Object-Oriented API that retains all the power of the OpenCL API without most of the C head-scratching : JavaCL Core \* demos \* basic utilities (parallel reduction, experimental matrix implementation for [UJMP](http://www.ujmp.org/) \* an experimental Scala DSL (Domain-Specific Language) : [ScalaCL](http://code.google.com/p/scalacl/)

## What does it look like ?

CLContext context = JavaCL.createBestContext(); CLProgram program = context.createProgram(myKernelSource).build(); CLBuffer<Integer> input = context.createIntBuffer(Usage.Input, inputBuffer, true); CLBuffer<Float> output = context.createFloatBuffer(Usage.Output, resultsBuffer, false); CLKernel kernel = program.createKernel( "myKernel", new float[] { u, v }, input, output ); ...

[*See the complete example...*](https://github.com/ochafik/nativelibs4java/tree/master/libraries/OpenCL/Core/src/test/java/com/nativelibs4java/opencl/OpenCL4JavaBasicTest.java)

## Why choose JavaCL over C+OpenCL ?

For quite a few reasons: \* Write and compile once, run everywhere: one-jar for all platforms, and JavaCL takes care of some well-known platform discrepancies / bugs \* Programmer-friendly API with hassle-free memory management \* Tons of helper code: \* image loading (and format conversion) \* OpenGL context sharing \* choice of "best" devices (with configurable criteria) \* linear algebra, reduction and random number generation utils \* Automatic and transparent caching of program binaries \* GUI to experiment with image transform kernels \* Very competitive performance (see [this thread](https://groups.google.com/forum/#!msg/nativelibs4java/PxzVFpVclP0/YuTwy1er5XEJ) for some figures) \* Powerful kernel code management: \* kernels are stored as plain old Java resources and can include each other straight from their JAR! \* JavaCL Generator regenerates typed wrappers at each build to prevent misuse of kernel arguments \* Books and Tutorials are available to get you started \* [groups.google.com/group/nativelibs4java JavaCL] has passionate support from its author(s) and its community.

## How do I get started ?

Please see the "Quick Start" section on the project's homepage.

## How fast is JavaCL ?

Pretty fast!

The call overhead is usually negligible when compared to the execution time of kernels or the time to transfer buffer data between main memory and device memory.

However, recent snapshots have been heavily optimized to reduce thread contention, GC activity and lower per-call overhead by a big factor: more details in [this thread](https://groups.google.com/forum/?fromgroups#!topic/nativelibs4java/PxzVFpVclP0).

## Is JavaCL thread-safe ?

Yes !

OpenCL 1.1 drivers are guaranteed to be thread-safe, and JavaCL dynamically adds synchronization to its low-level bindings to guarantee thread-safety even with OpenCL 1.0.

More specifically, if any OpenCL 1.0 platform is detected, JavaCL will force synchronization on low-level calls, so it will be faster if it only sees OpenCL 1.1 platforms.

## What does JavaCL offer in terms of memory management ?

JavaCL maps each OpenCL entity to a Java object. OpenCL entities are released whenever their corresponding Java object is garbage-collected, or when their release() method is called explicitly (which is strongly advised: please read this wiki page for more information).

## My old 1.0-beta code broke with 1.0-RC and later !

Please read MigratingFromJNAToBridJ.

## JavaCL Crashes on Linux (including demos)

Please read TroubleShootingJavaCLOnLinux for a workaround.

## I get a weird InvalidCommandQueue exception, why ?

Command queues can become invalid after some errors that can go unnoticed by default. To see all errors, please set the environment variable CL\_LOG\_ERRORS to stdout : \* On Windows : set CL\_LOG\_ERRORS=stdout \* On Unix / Linux / Mac OS X : export CL\_LOG\_ERRORS=stdout

See [this thread on macresearch.org](http://www.macresearch.org/opencl_episode4#comment-15801) or [that thread on khronos.org](http://www.khronos.org/message_boards/viewtopic.php?f=28&t=2061) for more details.

## I get a weird InvalidValue exception, why ?

If you are on MacOS X and have installed a utility to turn off your discrete GPU in order to optimize your battery life, you might want to make sure your GPU is indeed enabled when you run JavaCL : if not, you'll still see the GPU from OpenCL, but any attempt to use it will throw InvalidValue exceptions.

## My kernel returns weird double / float values when run on ATI Radeon GPUs

Radeon GPUs are big-endian, whereas most other GPUs and CPUs are little-endian.

Some precautions need to be taken to ensure a little-endian host (such as x86) can exchange data with a big-endian device, otherwise you'll see some insanely huge values popping out of nowhere.

Please read ReadFromAndWriteToBuffersAndImages for more information.

## Which version of OpenCL does JavaCL support ?

Since Apple hasn't bothered to go beyond OpenCL 1.0 yet, JavaCL mainly targets 1.0.

JavaCL does however support most features of 1.1, that nicely throw NoSuchOperationException errors when not available :-)

And support of 1.2 is planned, any help is welcome.

## Which platforms does JavaCL support ?

Short answer is "all of them". Please read details here...

Beware though that **on Linux you'll have to alter your environment settings to avoid crashes**. Please read details here...

## How do I build JavaCL / ScalaCL / OpenCL4Java ?

Please read the Build page.

## How do I run JavaCL Demos ?

You can launch the Particles and HardwareReport JavaCL Demos from the Java Web Start links from the front page.

If you've built JavaCL from the sources, you can also launch the demos directly in command line :

* Install JOGL properly as described on the Build page
* Go to libraries/OpenCL/Demos/target
* Launch your demo (read "What demos are available ?" on this page and pick the right demo class) : java -cp javacl-demos-1.0-SNAPSHOT-shaded.jar com.nativelibs4java.opencl.demos.hardware.HardwareReport

*(in Maveneese, "shaded" means "with all dependencies included" so there's only one JAR to put in the classpath, JOGL issues left apart)*

## What demos are available ?

JavaCL's most notable demos are : \* Particles Demo : shows a particles system animated in real time \* Interactive Image Transform Demo : lets you edit image transformation kernels and run them on the fly (with examples provided) \* Hardware Report : displays information about the available OpenCL implementations detected on your setup, with details on each device.

For more details, please visit SamplesAndDemos.

## How do I write and call OpenCL kernels from Java ?

Please read CallingOpenCLKernelsFromJava and look at the demos source code.

## Does JavaCL support OpenCL type XXX for kernel arguments ?

JavaCL supports all OpenCL types, please see UnderstandOpenCLKernelArguments for type mapping details.

## Can I include headers / source files from OpenCL kernels ?

Sure ! You can even include files that are in the Java classpath.

Please read IncludesInKernels for more details.

## How do I use buffers / images in JavaCL ?

Please read ReadFromAndWriteToBuffersAndImages

## What's up with double-precision floats ? Are they optional / supported ?

Not all OpenCL devices support double-precision floats (a.k.a doubles), but JavaCL will give you the most out of what is supported by the hardware : you can create and use double buffers, pass double parameters, and query whether an individual device has double support by calling [CLDevice.isDoubleSupported()](http://nativelibs4java.sourceforge.net/javacl/api/stable/com/nativelibs4java/opencl/CLDevice.html" \l "isDoubleSupported()).

Please also note that there are two extensions for double support (they will yield different results for some built-in functions) : \* [cl\_khr\_fp64](http://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/cl_khr_fp64.html) \* [cl\_amd\_fp64](http://www.khronos.org/registry/cl/extensions/amd/cl_amd_fp64.txt)

[CLDevice.isDoubleSupported()](http://nativelibs4java.sourceforge.net/javacl/api/stable/com/nativelibs4java/opencl/CLDevice.html#isDoubleSupported()) will return true if either of these extensions is supported, but you may query them separately with [isDoubleSupportedKHR()](http://nativelibs4java.sourceforge.net/javacl/api/stable/com/nativelibs4java/opencl/CLDevice.html" \l "isDoubleSupportedKHR()) and [isDoubleSupportedAMD()](http://nativelibs4java.sourceforge.net/javacl/api/stable/com/nativelibs4java/opencl/CLDevice.html" \l "isDoubleSupportedAMD()).

Also, you'll need to enable one of these extensions explicitly in your OpenCL source code with something like : ```

# pragma OPENCL EXTENSION cl\_khr\_fp64 : enable

```

You can safely pick cl\_khr\_fp64, for **JavaCL will automatically convert the #pragma directive to the AMD variant** if the KHR extension is not present and the AMD extension is (you can turn off this behaviour by setting the property javacl.adjustDoubleExtension to false or the environment variable JAVACL\_ADJUST\_DOUBLE\_EXTENSION to 0).

## My question is not here... How can I get some help ?

JavaCL has a [user group / mailing-list](http://groups.google.com/group/nativelibs4java) : you can search in its archive to find similar past questions or post a new message, we'll try to help the best we can.

## How can I contribute ?

Here are some ways to contribute to JavaCL : \* Join the [user group / mailing-list](http://groups.google.com/group/nativelibs4java) and help newbies \* Download the source code and make it evolve / help fix bugs (for a list of pending bugs, please go to the Issues page \* Make remarks / suggestions / create content for the project's documentation (including this page) \* Create new demos \* Contribute hardware (weird platforms, weird graphic cards) to the project \* Contribute money to JavaCL's author :

*http://sourceforge.net/donate/index.php?group\_id=266856'>http://images.sourceforge.net/images/project-support.jpg' alt='Support This Project' border='0' width='88' height='32' /> (Olivier Chafik is also the author of [JNAerator](http://jnaerator.googlecode.com/))*