

---

## 1 How to Build a Sample

Ensure that the AMD APP SDK Samples package has been installed before proceeding.

The following sections describe the steps to build and execute a sample that's included in the APP SDK v2.8.1. The samples may be OpenCL, OpenCV-CL, C++ AMP, Bolt, or Aparapi samples.

To run pre-built Bolt and C++AMP samples, Microsoft Visual Studio 2012 must be installed on the host operating system.

C++ AMP samples can be built and executed only with Microsoft Visual Studio 2012.

Bolt OpenCL samples can be built and executed with Microsoft Visual Studio versions 2010 and 2012. Bolt C++ AMP samples can be built with Microsoft Visual Studio 2012.

### 1.1 OpenCL On Windows

#### Building With Visual Studio Solution Files –

The samples installation contains a Microsoft Visual Studio 2008 solution file (`OpenCLSamples.sln`), a Microsoft Visual Studio 2010 solution file (`OpenCLSamplesVS10.sln`), and a Microsoft Visual Studio 2012 solution file (`OpenCLSamplesVS12.sln`). These are located at `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\`. The solution file contains the entire sample project. To build a sample:

1. Open the `OpenCLSamples.sln` file with Microsoft Visual Studio 2008 Professional Edition or open the or `OpenCLSamplesVS10.sln` with Microsoft Visual Studio 2010 Professional Edition or `OpenCLSamplesVS12.sln` with Microsoft Visual Studio 2012 Professional Edition.
2. Select **Build > Build Solution** to build all solutions.
3. Select the project file in the Solutions Explorer.

#### Building With Visual Studio Solution Files Using the Intel Compiler (icl) –

The samples installation contains a Microsoft Visual Studio 2008 solution file (`OpenCLSamples.sln`), a Microsoft Visual Studio 2010 solution file (`OpenCLSamplesVS10.sln`), and a Microsoft Visual Studio 2012 solution file (`OpenCLSamplesVS12.sln`). These are located at `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\`. The solution file contains the entire sample project. To build a sample:

1. Open the `OpenCLSamples.sln` file with Microsoft Visual Studio 2008 Professional Edition, or open the `OpenCLSamplesVS10.sln` with Microsoft Visual Studio 2010 Professional Edition, or `OpenCLSamplesVS12.sln` with Microsoft Visual Studio 2012 Professional Edition.

2. Right-click on a project file, and select *Properties*.
3. Under Configuration Properties | General, change the Platform Toolset to Intel C++ Compiler, and Click OK.
4. Right-click on the project file, and select *Build* to build the sample.

#### Building With VS Compiler Using Makefiles -

1. Open the Visual Studio command prompt. (Ensure `make.exe` is in the PATH. This executable can be from cygwin or MinGW. Open the 32-bit Visual Studio command prompt for 32-bit builds; open the 64-bit Visual Studio command prompt for 64-bit builds.)
2. Go to the `AMDAPPSDKSAMPLESROOT\samples\opengl\` directory to build all samples. To build samples individually, go to the `AMDAPPSDKSAMPLESROOT\samples\opengl\cl\app` directory.
3. On a 64-bit machine, use `make` to build the debug binaries; to create a 32-bit build on a 64-bit machine, use `make bitness=32`. By default, `bitness` is set to native bits supported by the architecture and OS.
4. To create release version of the binaries, use `make bitness=<32/64> release=1`

#### Building with DirectX -

To build DirectX samples in the package:

1. Ensure that the Microsoft DirectX SDK (June 2010) is installed.
2. Open a Visual Studio command prompt.
3. Run `C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Utilities\bin\dx_setenv.cmd`. A successful execution of `dx_setenv.cmd` outputs the following message:

```
Dx x64 target environment is now enabled.
```

```
Dx x64 host environment is now enabled.
```

This allows the DirectX samples in the package to be built.

#### Building With MinGW + GCC Using Makefiles -

1. Open the command window (Start > Run, then type `cmd.exe`).
2. Add MinGW bin and Msys bin to PATH:  
`PAT=C:\MinGW\bin;C:\MinGW\msys\1.0\bin;%PATH%`
3. Go to the `AMDAPPSDKSAMPLESROOT\samples` directory.
4. `make bitness=32`.

#### Building With MinGW-x64 + GCC Using Makefiles -

1. Open the command window (Start > Run, then type `cmd.exe`).
2. Add MinGW64 bins to PATH: `PAT=C:\MinGW64\bin;C:\MinGW\msys\1.0\bin;%PATH%`
3. Go to the `AMDAPPSDKSAMPLESROOT\samples` directory.
4. `make bitness=64`.

**Note:** To use Msys under MinGW64-x64:

- a. Open the `fstab` file (available at `msys/1.0/etc/`)
- b. Modify according to your MinGW64-w64 path. For example, modify  
`C:\MinGW\ /mingw` to `C:\MinGW64\ /mingw`

## 1.2 C++ AMP On Windows

### Building With Visual Studio Solution Files –

The samples installation contains a Microsoft Visual Studio 2012 solution file (`OpenCLSamplesVS12.sln`). This is located at `$(AMDAPPSDKSAMPLESROOT)\samples\C++AMP\`.

**Note:** C++ AMP is supported only by Microsoft Visual Studio 2012. Also, C++ AMP samples do not work on Linux.

The solution file contains the entire sample project. To build a sample:

1. Open the `C++AmpSamplesVS12.sln` file with Microsoft Visual Studio 2012 Professional Edition.
2. Select *Build > Build Solution* to build all solutions.
3. Select the project file in the Solutions Explorer.
4. Right-click on the project file, and select *Build* to build a particular sample.

### Building With VS Compiler Using Makefiles –

1. Open the Visual Studio command prompt. (Ensure `make.exe` is in the PATH. This executable can be from cygwin or MinGW. Open the 32-bit Visual Studio command prompt for 32-bit builds; open the 64-bit Visual Studio command prompt for 64-bit builds.)
2. Go to the `AMDAPPSDKSAMPLESROOT\samples` directory to build all samples. To build samples individually, go to the `AMDAPPSDKSAMPLESROOT\samples\C++AMP\examples` directory.
3. On a 64-bit machine, use `make` to build the debug binaries; to create a 32-bit build on a 64-bit machine, use `make bitness=32`. By default, `bitness` is set to native bits supported by the architecture and OS.
4. To create a release version of the binaries, use `make bitness=<32/64> release=1`

## 1.3 Bolt On Windows

### Installing of Bolt 1.0 AND TBB Libraries –

1. Ensure that your system contains the following prerequisites for Bolt:
  - ☐ Windows 7/8
  - ☐ Microsoft Visual Studio version 2010 or higher (Microsoft Visual Studio 2012 for APP SDK Bolt C++ AMP samples)
  - ☐ CMake 2.8.10 (required only if one needs to build Bolt libraries from source)
  - ☐ TBB library (For Multi-core CPU path, BOLT is tested with 4.1 Update 3)

2. Download the TBB libraries from <http://threadingbuildingblocks.org/download> - 4.1 Update 3 and install the binaries.
  3. Set the environmental variable `TBB_ROOT` to the root directory of the installed TBB binaries.
  4. Append the `PATH` environment variable with the directory containing all the TBB .dll files. For example, for a 64-bit machine with VS 2012, this path will be:  
`%TBB_ROOT%\bin\intel64\vc11\.`
  5. Download and install the Bolt 1.0 prebuilt binaries from <http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/bolt-c-template-library/>.  
 Bolt can also be built from the source: <https://github.com/HSA-Libraries/Bolt> using CMake 2.8.10.
4. Set the `BOLTLIB_DIR` environmental variable to the root directory of the Bolt binaries.

#### Building With Visual Studio Solution Files –

The samples installation contains a Microsoft Visual Studio 2010 solution file (`OpenCLSamplesVS10.sln`) and Microsoft Visual Studio 2012 solution file (`OpenCLSamplesVS12.sln`). This file is located at `$(AMDAPPSDKSAMPLESROOT)\samples\Bolt\.`

**Note:** Bolt samples can be built only by Microsoft Visual Studio 2010 and Microsoft Visual Studio 2012. Also, Bolt samples do not work on Linux.

The solution file contains the entire sample project. To build a sample:

1. Open the `BoltSamplesVS10.sln` or `BoltSamplesVS12.sln` file with the appropriate Microsoft Visual Studio Professional Edition.
2. Select *Build > Build Solution* to build all solutions.
3. Select the project file in the Solutions Explorer.
4. Right-click on the project file, and select *Build* to build a particular sample.

#### Building With VS Compiler Using Makefiles –

1. Open the Visual Studio cross tools command prompt. (Ensure `make.exe` is in the `PATH`. This executable can be from cygwin or MinGW. Open the 32-bit Visual Studio command prompt for 32-bit builds; open the 64-bit Visual Studio command prompt for 64-bit builds.
2. Go to the `AMDAPPSDKSAMPLESROOT\samples\bolt` directory to build all samples. To build samples individually, go to the `AMDAPPSDKSAMPLESROOT\samples\bolt\examples` directory.
3. Use `make` to build the debug binaries.
4. To create a release version of the binaries, use `make bitness=<32/64> release=1`.

## 1.4 OpenCL On Linux

#### Building With GCC –

This samples installation contains `makefiles` to build samples. To build a sample:

1. Ensure the library path is set to find `libOpenCL.so`. If it is not set, follow the instructions in the *AMD APP SDK Installation Notes*, section 3.2.

2. Go to the `samples` directory if you want to build all the samples, or go to a particular sample directory, and type `make`. If no options are specified, `make` builds for the native platform in debug configuration. To select release configuration, add the option `release = 1`. To force a 32-bit build on a 64-bit platform, add the option `bitness=32`.
3. Executables are generated in the `bin` directory at the samples installation root, as well as inside the `bin` directory of the samples: `AMDAPPSDKSAMPLESROOT/samples/opencv/bin`.

#### Building With the Intel Compiler –

1. Use the option `intel=1`; all other options are the same as above.

Add a new section

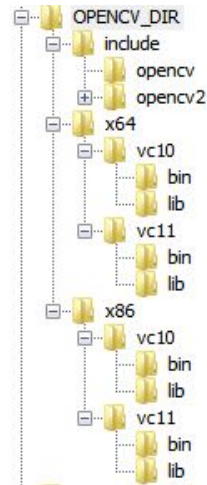
## 1.5 OpenCV on Windows and Linux

APP SDK v2.8.1 OpenCV samples require OpenCV 2.4.4 to be installed as a prerequisite. The prebuilts of OpenCV 2.4.4 do not have the OpenCV-CL libraries in them. To compile and run the OpenCV samples in APP SDK v2.8.1, you must build from the source files of OpenCV 2.4.4, enabling the OpenCL flag in CMake.

Before building and running OpenCV-CL samples, perform the following actions:

1. Build the OpenCV library from the source files.  
The OpenCV sources for the 2.4.4 build are available at <https://github.com/Itseez/opencv/tree/2.4.4>. To have OpenCL support for OpenCV, the `opencv_ocl` library must be built. During configuring with CMake, select the `WITH_OPENCCL` option and provide the path of the OpenCL library (`libOpenCL.so` in Linux and `OpenCL.lib` in Windows). The following links from [opencv.org](http://opencv.org) are useful:
  - Linux: [http://docs.opencv.org/doc/tutorials/introduction/linux\\_install/linux\\_install.html](http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html)
  - Windows: [http://docs.opencv.org/doc/tutorials/introduction/windows\\_install/windows\\_install.html](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html)
2. Set the correct directory structure.  
On Windows, the directory structure of the compiled OpenCV binaries created in the preceding step must be restructured as per the OpenCV prebuilt directory structure. This step ensures that, when future OpenCV releases include the OpenCV-CL components in the prebuilts, developers do not need to change the path of the OpenCV library references in their projects.  
Restructure the OpenCV binaries created from sources as per the prebuilt directory structure for Windows as shown in the following figure. Place the created `include` directly under the root OpenCV directory and copy the `bin` and `lib` files (both debug and release versions) into the following locations:
  - `x86/vc10`, if your binaries and libraries are built with Microsoft Visual Studio 10 and the target is x86.
  - `x86/vc11`, if your binaries and libraries are built with Microsoft Visual Studio 12 and the target is x86.
  - `x64/vc10` if your binaries and libraries are built with Microsoft Visual Studio 10 and the target is x64.

- x64/vc11 if your binaries and libraries are built with Microsoft Visual Studio 12 and the target is x64.



Alternatively, you may skip restructuring the directory as per the OpenCV pre-built, in which case, the Visual Studio property sheets of the OpenCV-CL projects must be updated to point to the correct paths.

For Linux, retain the default directory structure created from sources.

### 3. Set the environment Variables

- Create and set the environmental variable, `OPENCV_DIR`, to the root directory containing the OpenCV include and lib files created in the preceding step.
- Create and set the environmental variable, `OCVCL_VER`, to the OpenCV version used for APP SDK 2.8.1 release, that is, set `OCVCL_VER = 244`.
- For Windows, set `PATH` to the directory containing all the OpenCV .dll files. For example, for a 64-bit machine with VS 2012, this path will be `%OPENCV_DIR%\x64\vc11\bin`. If you have built OpenCV with `AMD BLAS` and `FFT`, then set those paths also
- For Linux, export `LD_LIBRARY_PATH` to the directory containing all the OpenCV shared object files.

For answers to frequently asked questions on OpenCV, see the AMD APP SDK FAQ.

## 2 How to Run the Application

This section describes how to run the application that was just built.

### 2.1 On Windows

There are three ways to run the application: by using Microsoft Visual Studio 2008 Professional Edition, using Microsoft Visual Studio 2010 Edition, using Microsoft Visual Studio 2012 Edition, or by using the command line. Note that Microsoft Visual Studio 2008 and Microsoft Visual Studio 2010 cannot be used for all samples, as indicated in the earlier sections. The following

description is applicable for OpenCL samples only and is to be used as a general guide for the other class of samples.

### Using Visual Studio

1. Open `OpenCLSamples.sln` with Microsoft Visual Studio 2008 Professional Edition, or `OpenCLSamplesVS10.sln` with Microsoft Visual Studio 2010 Professional Edition, or `OpenCLSamplesVS12.sln` with Microsoft Visual Studio 2012 Professional Edition, and build it.
2. Select the desired project file in the Solutions Explorer.
3. Right-click on it, and select *Set as StartUp Project*. To run the application, press Ctrl+F5. To run the application in debug mode, simply press F5.

### Using the Command Line

1. Open a command prompt.
2. Go to the `$(AMDAPPSDKSAMPLESROOT)/samples/ocl/bin.`
3. Go into the appropriate architecture directory (`x86` or `x86_64`).
4. Run the samples by typing the name of their executables. See the individual sample documents for their respective command line arguments.

## 2.2 On Linux

1. Ensure the path is set to include the location of `libOpenCL.so`.
2. Open a terminal window.
3. Go to the `$(AMDAPPSDKSAMPLESROOT)/samples/ocl/bin/` directory.
4. Go into the appropriate architecture directory (`x86` or `x86_64`).
5. Run the samples by typing the name of their executables. You may have to prepend the executable name with `./`. See the individual sample documents for their respective command line arguments.

## 2.3 Sample Code

The simplest OpenCL sample in the SDK is the HelloWorld sample. It is for developers that are new to OpenCL programming. See the HelloWorld sample documentation (in `$(AMDAPPSDKSAMPLESROOT)/samples/ocl/cl/app/HelloWorld/docs`), which explains the workflow for setting up a basic AMD APP application using OpenCL. The HelloWorld sample is included for instructional purposes.

For basic debugging techniques, see the BasicDebug sample.

There are two kinds of OpenCL samples in the AMD APP SDK. One is written using native OpenCL calls (in `$(AMDAPPSDKSAMPLESROOT)/samples/ocl/cl`); the other is written using the AMD C++ bindings to OpenCL (in `$(AMDAPPSDKSAMPLESROOT)/samples/ocl/cpp_cl`).

Most of the OpenCL samples make use of a utility library called the AMD APP SDK Utility Library (SDKUtil). This library provides commonly used routines, such as parsing command line options, loading and writing bitmap images, printing formatted output, comparing results, and reading files.

This is an object-oriented library that is structured as a set of classes. The SDKUtil `cpp` files and headers are available in the `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\SDKUtil` folder.

### 3 Aparapi

The samples for Aparapi are different from the other samples because all the samples are coded in Java. These samples can be executed on Windows and Linux; they do not require a particular version of Microsoft Visual Studio.

To build and run the Aparapi examples requires the following environment variables to be configured:

- Set `JAVA_HOME` to the directory containing JRE/JDK, version 1.7 or above.
- Set `ANT_HOME` to the directory containing ANT, version 1.8 or above.
- Set `LIBAPARAPI` to the directory where [aparapi-2012-11-14.zip](#) (or above) is unzipped.
- Ensure that `PATH` is set such that the `java` and `javac` executables are used from JDK version 1.7 or above, and the `ant` executable is used from `ANT_HOME`.
- In Linux, set `LD_LIBRARY_PATH` to the directory where [aparapi-2012-11-14.zip](#) (or above) is unzipped.

To build the example, first `cd` into  
`AMDAPPSDKSAMPLESROOT\samples\aparapi\AparapiUtil` folder.

If building from the command line: Type `ant` to create `AparapiUtil.jar`.

If building on Eclipse, open the `AparapiUtil` project and build.

**Note:** Before importing projects into Eclipse, do the following:

- On Linux: In the `$(AMDAPPSDKSAMPLESROOT)/samples/aparapi`, run  
`sh ./setenv_aparapi.sh`.
- On Windows: In the `$(AMDAPPSDKSAMPLESROOT)/samples/aparapi`, run  
`setenv_aparapi.bat`.

This updates the `.classpath` for all the examples, which allows Aparapi projects to be successfully imported and built using Eclipse.

Then, change the directory to `AMDAPPSDKSAMPLESROOT\samples\aparapi\<example>`

If building from the command line: Type `ant` to create `<example>.jar`.

If building on Eclipse, open the `<example>` project, and build.

To run the sample on Windows or Cygwin, run the appropriate `<example>.bat` from the command line. Right-clicking on the `<example>` project in Eclipse and running it has the same effect. `<example>.bat -h` lists the options exposed by the sample.

To run the sample on a Linux terminal, run the appropriate `<example>.sh`.

The `makefile` also can be used to build the sample.



## 4 Important Notes

- Unless specifically recommended otherwise, developers must use the latest graphics drivers for their platform. These drivers can be downloaded from <http://support.amd.com/us/Pages/AMDSupportHub.aspx>.

For current recommendations, click [here](#).

- For SDK 2.8.1, the following values are returned when querying strings from OpenCL:

CL\_PLATFORM\_VERSION: OpenCL 1.2 AMD-APP (*build.revision*)

CL\_PLATFORM\_NAME: AMD Accelerated Parallel Processing

CL\_PLATFORM\_VENDOR: Advanced Micro Devices, Inc.

- Check the Platform Vendor string, not the Platform Name, to determine AMD hardware. For example code that shows how to check and use the CL\_PLATFORM\_VENDOR string, see the AMD APP v2 OpenCL samples.

## 5 Supported Devices

We are continually qualifying devices. For an up-to-date list of supported devices, please visit the [APP SDK System Requirement and Driver Compatibility](#) page.

---

### Contact

Advanced Micro Devices, Inc.  
One AMD Place  
P.O. Box 3453  
Sunnyvale, CA, 94088-3453  
Phone: +1.408.749.4000

### For AMD Accelerated Parallel Processing:

URL: [developer.amd.com/appsdk](http://developer.amd.com/appsdk)  
Developing: [developer.amd.com/](http://developer.amd.com/)  
Forum: [developer.amd.com/openclforum](http://developer.amd.com/openclforum)



The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

### Copyright and Trademarks

© 2013 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ATI, the ATI logo, Radeon, FireStream, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. Other names are for informational purposes only and may be trademarks of their respective owners.