



API Debugger Tutorial

1 INTRODUCTION

OpenCL™ API Debugger is a Microsoft Visual Studio* plug-in that enables debugging of OpenCL applications by allowing developers to monitor and understand their OpenCL environment. API Debugger is a new feature of the Compute Code Builder found in the Intel® Integrated Native Developer Experience (Intel® INDE) and the Intel® SDK for OpenCL™ Applications 2014 Beta.

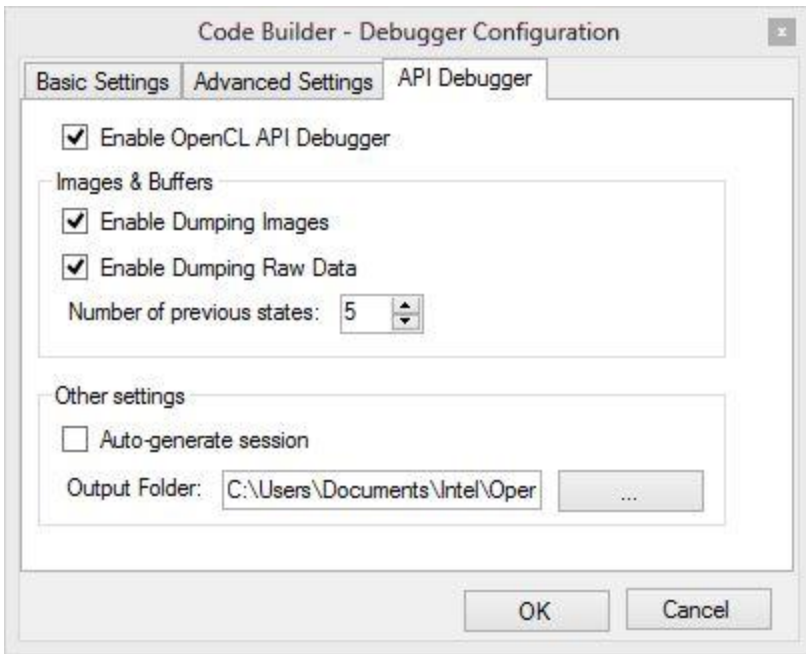
The API Debugger lets developers view the list of all OpenCL API calls that their applications call with an ability to view information about the function parameters, return values, and execution times. The plug-in also allows for viewing various OpenCL objects that exist in memory while the application is executing. You can read more about API Debugger and its features in the [Intel® SDK for OpenCL™ Applications 2014 Beta – User's Guide](#).

In this tutorial we will show one important usage of API Debugger and that is to debug and find the root cause of an application failure in the presence of only the kernel source code and the application binary (no host side source code). Note that a very nice feature of the API Debugger is the ability to show the kernel source code even if it is embedded in the application being debugged. In the latter case, it is possible to identify the source of the kernel bug but not fix it in the absence of full source code for the application.

For demonstration purposes we will introduce a bug in the sobel filter kernel from the [JumpStart tutorial](#) and show you how to identify the bug without looking at the host source code.

2 ENABLE API DEBUGGER IN VISUAL STUDIO*

The first step is to make sure the API Debugger is enabled in Visual Studio. To do this, click **Tools**→**Code Builder – Options** in Visual Studio or press **Ctrl+1** to open the Debugger Configuration dialog as shown below.



Make sure you have at least the **Enable OpenCL API Debugger** option checked and click OK. Once the plug-in is enabled you can start debugging your application.

3 LAUNCHING THE APPLICATION

As mentioned previously, we introduced a bug in the sobel application by changing the kernel argument types for width and height as shown below.

```
kernel void
sobel(__global uchar4* src, __global uchar4* dst, ushort width, ushort height)
{
```

When we try to run the application with this new kernel, the application fails at the point the kernel arguments are set. Assuming no proper error handling, this is very difficult to detect without full source code. Now let's see how this issue can be debugged.

Since we have only the application binary and the OpenCL kernel, we need to somehow launch the application to be debugged from Visual Studio. Start the Visual Studio command prompt from the Start menu or alternately add %VSINSTALLDIR%\Common7\IDE\ (where %VSINSTALLDIR% refers to the directory where Visual Studio is installed) to the PATH environment variable and start a command prompt. Launch the application from the command prompt like so:

```
devenv <application>
```

where <application> is the full or relative path of the application we are trying to debug, say sobel.exe.

At this point, Visual Studio should launch with the application to be debugged. Make sure the API Debugger is enabled as shown in the previous section. Start debugging by pressing **F5**. The application will launch but will fail without executing the OpenCL kernel.

4 PROBLEMS VIEW AND TRACE VIEW

Open the Problems View by going to **Tools→Code Builder→OpenCL Debugger→Problems View**. The **Problems View** window will list all the warnings and errors resulting from running the OpenCL application. We are particularly interested in the errors (but it's good practice to check the warnings too). You will see two errors as a result of calling `clSetKernelArg()` as shown below.

Problems view	
Save As...	2 Errors 2 Warnings
#	Description
1	Kernel [1] (sobel)'s argument number 2 is not initialized yet
2	Kernel [1] (sobel)'s argument number 3 is not initialized yet
3	clSetKernelArg API call #15 failed with CL_INVALID_ARG_SIZE. Right-click here to show the trace.
4	clSetKernelArg API call #16 failed with CL_INVALID_ARG_SIZE. Right-click here to show the trace.

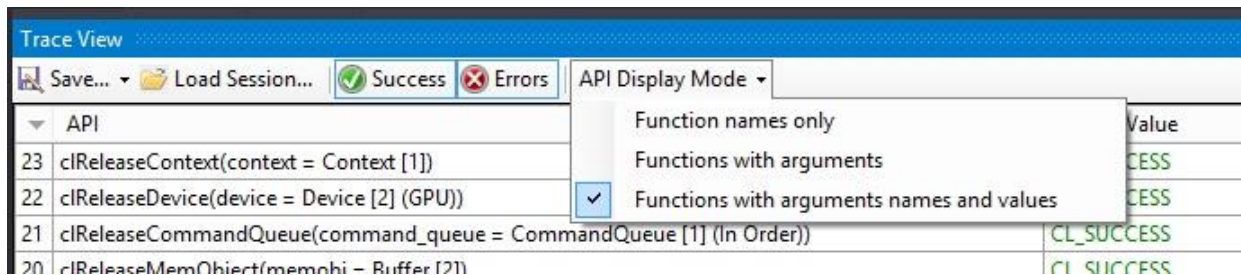
The above view only tells us half the story—that the call failed with the `CL_INVALID_ARG_SIZE`. Now right-click one of the errors and select **Show in Trace View**.

Problems view	
Save As...	2 Errors 2 Warnings
#	Description
1	Kernel [1] (sobel)'s argument number 2 is not initialized yet
2	Kernel [1] (sobel)'s argument number 3 is not initialized yet
3	clSetKernelArg API call #15 failed with CL_INVALID_ARG_SIZE. Right-click here to show the trace.
4	clSetKernelArg API call #16 failed with CL_INVALID_ARG_SIZE. Right-click here to show the trace.

This will open the **Trace View** window with the OpenCL API trace.

Trace View		
Save...	Load Session...	Success Errors API Display Mode ▾
API	Return Value	Time
23 clReleaseContext(Context [1])	CL_SUCCESS	14:02:49:676
22 clReleaseDevice(Device [2] (GPU))	CL_SUCCESS	14:02:49:660
21 clReleaseCommandQueue(CommandQueue [1] (In Order))	CL_SUCCESS	14:02:49:660
20 clReleaseMemObject(Buffer [2])	CL_SUCCESS	14:02:49:660
19 clReleaseMemObject(Buffer [1])	CL_SUCCESS	14:02:49:660
18 clReleaseProgram(Program [1])	CL_SUCCESS	14:02:49:660
17 clReleaseKernel(Kernel [1] (sobel))	CL_SUCCESS	14:02:49:660
16 clSetKernelArg(Kernel [1] (sobel), 0x3, 0x4, 0x114f874)	CL_INVALID_ARG_SIZE	14:02:49:660
15 clSetKernelArg(Kernel [1] (sobel), 0x2, 0x4, 0x114f870)	CL_INVALID_ARG_SIZE	14:02:49:660
14 clSetKernelArg(Kernel [1] (sobel), 0x1, 0x4, 0x114fb40)	CL_SUCCESS	14:02:49:660
13 clSetKernelArg(Kernel [1] (sobel), 0, 0x4, 0x114fb38)	CL_SUCCESS	14:02:49:660
12 clCreateKernel(Program [1], sobel, 0x114fb64)	CL_SUCCESS	14:02:49:660
11 clBuildProgram(Program [1], 0x1, 0x114fb28, , NULL, NULL)	CL_SUCCESS	14:02:49:660
10 clCreateProgramWithSource(Context [1], 0x1, 0x114f858, 0x114fb4c, 0x114fb...	CL_SUCCESS	14:02:49:192
9 clCreateBuffer(Context [1], CL_MEM_WRITE_ONLY CL_MEM_USE_HOST_P...	CL_SUCCESS	14:02:49:192
8 clCreateBuffer(Context [1], CL_MEM_READ_ONLY CL_MEM_USE_HOST_PT...	CL_SUCCESS	14:02:49:192
7 clGetDeviceInfo(Device [2] (GPU), CL_DEVICE_MEM_BASE_ADDR_ALIGN, 0x4...	CL_SUCCESS	14:02:48:332
6 clCreateCommandQueue(Context [1], Device [2] (GPU), CL_QUEUE_PROFILI...	CL_SUCCESS	14:02:48:332
5 clGetContextInfo(Context [1], CL_CONTEXT_DEVICES, 0x4, 0x114fb28, NULL)	CL_SUCCESS	14:02:48:332
4 clCreateContextFromType(0x114fb18, CL_DEVICE_TYPE_GPU, NULL, NULL, 0...	CL_SUCCESS	14:02:48:332
3 clGetDeviceIDs(Platform [1] (Intel(R) OpenCL), CL_DEVICE_TYPE_GPU, 0, NU...	CL_SUCCESS	14:02:48:332
2 clGetPlatformIDs(0x1, 0x13693c8, NULL)	CL_SUCCESS	14:02:48:317
1 clGetPlatformIDs(0, NULL, 0x114f380)	CL_SUCCESS	14:02:48:317

In the Trace View window select **Functions with arguments names and values** from the list of **API Display Mode** as shown below.



This verbose display mode gives additional API details including the number of arguments and their sizes.

API	Return Value	Time
23 clReleaseContext(context = Context [1])	CL_SUCCESS	14:02:49.676
22 clReleaseDevice(device = Device [2] (GPU))	CL_SUCCESS	14:02:49.660
21 clReleaseCommandQueue(command_queue = CommandQueue [1] (In Order))	CL_SUCCESS	14:02:49.660
20 clReleaseMemObject(memobj = Buffer [2])	CL_SUCCESS	14:02:49.660
19 clReleaseMemObject(memobj = Buffer [1])	CL_SUCCESS	14:02:49.660
18 clReleaseProgram(program = Program [1])	CL_SUCCESS	14:02:49.660
17 clReleaseKernel(kernel = Kernel [1] (sobel))	CL_SUCCESS	14:02:49.660
16 clSetKernelArg(kernel = Kernel [1] (sobel), arg_index = 0x3, arg_size = 0x4, arg_value = 0x114f874)	CL_INVALID_ARG_SIZE	14:02:49.660
15 clSetKernelArg(kernel = Kernel [1] (sobel), arg_index = 0x2, arg_size = 0x4, arg_value = 0x114f870)	CL_INVALID_ARG_SIZE	14:02:49.660
14 clSetKernelArg(kernel = Kernel [1] (sobel), arg_index = 0x1, arg_size = 0x4, arg_value = 0x114fb40)	CL_SUCCESS	14:02:49.660
13 clSetKernelArg(kernel = Kernel [1] (sobel), arg_index = 0x0, arg_size = 0x4, arg_value = 0x114fb38)	CL_SUCCESS	14:02:49.660
12 clCreateKernel(program = Program [1], kernel_name = sobel, errcode_ret = 0x114f864)	CL_SUCCESS	14:02:49.660
11 clBuildProgram(program = Program [1], num_devices = 0x1, device_list = 0x114fb28, options = , pfn_notify = NULL, user_data = NULL)	CL_SUCCESS	14:02:49.660
10 clCreateProgramWithSource(context = Context [1], count = 0x1, strings = 0x114f858, lengths = 0x114f84c, errcode_ret = 0x114f864)	CL_SUCCESS	14:02:49.660
9 clCreateBuffer(context = Context [1], flags = CL_MEM_WRITE_ONLY CL_MEM_USE_HOST_PTR, size = 0x50000, host_ptr = 0x3ff940, errcode_ret = 0x114f854)	CL_SUCCESS	14:02:49.192
8 clCreateBuffer(context = Context [1], flags = CL_MEM_READ_ONLY CL_MEM_USE_HOST_PTR, size = 0x50000, host_ptr = 0x3ff940, errcode_ret = 0x114f854)	CL_SUCCESS	14:02:49.192

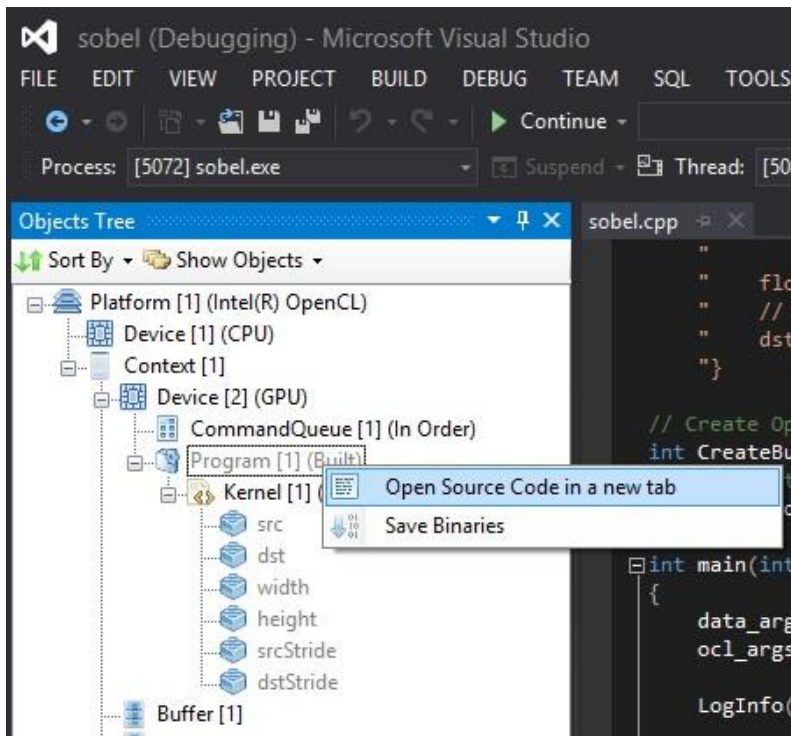
Looking at the offending calls and the return value of `clSetKernelArg()`, it is obvious there is some problem with setting kernel arguments #2 and #3. Let's look at the kernel signature again.

```
kernel void
sobel(__global uchar4* src, __global uchar4* dst, ushort width, ushort height)
{
```

You can see from the **Trace View** details that the host code set the kernel arguments whose size is 4 (from `arg_size = 0x4` for the two offending calls), whereas the OpenCL kernel is accepting `ushort` whose size is 2. Now we know that either the host code or the kernel needs to be fixed so that both types match.

4.1 VIEWING EMBEDDED KERNEL SOURCE CODE

As mentioned previously, API Debugger can show kernel source code even if it is embedded in the host binary. As before, make sure API Debugger is enabled and launch your application from Visual Studio. After breaking into the debugger open the Objects Tree View (Tools→Code Builder – OpenCL Debugger→Objects Tree View) and look for the built program node (something like “Program [1] (Built)”). Then right-click on the program node and choose “Open Source Code in a new tab” to view the kernel source code.



Note that the above binary was built without debugging information. So it is possible to inspect the kernel source code even when the binary is built in release mode.

5 SUMMARY

This tutorial shows how **API Debugger** can help determine the root cause of certain errors in OpenCL applications, particularly when source code is not available. API Debugger has much more functionality and nicely extends the debugging capabilities of Visual Studio by providing capabilities for debugging OpenCL host applications from the IDE.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Copyright © 2014 Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc and are used by permission by Khronos.