

# Chapter 1

## simpleCl.h API

### 1.1 data types

---

```
*/
typedef struct {
    cl_platform_id platform;
    cl_context context;
    cl_device_id device;
    cl_command_queue queue;
    int nComputeUnits;
    unsigned long int maxPointerSize;
    int deviceType; /* deviceType 0 = GPU | deviceType 1 = CPU |
                    deviceType 2 =
                        Accelerator | deviceType 3 = other */
    int devNum;
} sclHard;
typedef sclHard *ptsclHard;
typedef struct {
    cl_program program;
    cl_kernel kernel;
    char kernelName[98];
} sclSoft;
//
```

---

### 1.2 USER FUNCTIONS

#### 1.2.1 Device memory allocation read and write

---

```
*/

cl_mem sclMalloc(sclHard hardware, cl_int mode, size_t size);
```

```

cl_mem sclMallocWrite(sclHard hardware, cl_int mode, size_t size,
                      void *hostPointer);
void sclWrite(sclHard hardware, size_t size, cl_mem buffer, void
             *hostPointer);
void sclRead(sclHard hardware, size_t size, cl_mem buffer, void
            *hostPointer);

/*

```

---

## 1.2.2 initialization of sclSoft structs

---

```

*/

void sclGetCLSoftware(const char *kernel_file, const char *kernel_name,
                     const sclHard hardware, sclSoft *Software);

/*

```

---

## 1.2.3 Release and retain OpenCL objects

---

```

void sclReleaseClSoft(sclSoft soft);
void sclReleaseClHard(sclHard hard);
void sclRetainClHard(sclHard hardware);
void sclReleaseAllHardware(sclHard *hardList, cl_int found);
void sclRetainAllHardware(sclHard *hardList, cl_int found);
void sclReleaseMemObject(cl_mem object);

/*

```

---

## 1.2.4 Debug functions

---

```

void sclPrintErrorFlags(cl_int flag);
void sclPrintHardwareStatus(sclHard hardware);
void sclPrintDeviceNamePlatforms(sclHard *hardList, cl_int found);

/*

```

---

## 1.2.5 Device execution

---

```

cl_event sclLaunchKernel(sclHard hardware, sclSoft software,
                        size_t *global_work_size, size_t
                        *local_work_size);
cl_event sclEnqueueKernel(sclHard hardware, sclSoft software,
                        size_t *global_work_size, size_t
                        *local_work_size);
cl_event sclSetArgsLaunchKernel(sclHard hardware, sclSoft software,
                                size_t *global_work_size,
                                size_t *local_work_size,
                                const char *sizesValues, ...);
cl_event sclSetArgsEnqueueKernel(sclHard hardware, sclSoft software,
                                size_t *global_work_size,
                                size_t *local_work_size,
                                const char *sizesValues, ...);
cl_event sclManageArgsLaunchKernel(sclHard hardware, sclSoft software,
                                    size_t *global_work_size,
                                    size_t *local_work_size,
                                    const char *sizesValues, ...);
/*

```

---

### 1.2.6 Event queries

---

```

*/

cl_ulong sclGetEventTime(sclHard hardware, cl_event event);

/*

```

---

### 1.2.7 Queue management

---

```

*/

cl_int sclFinish(sclHard hardware);

/*

```

---

### 1.2.8 Kernel argument setting

---

```

*/

void sclSetKernelArg(sclSoft software, int argnum, size_t typeSize,
                    void *argument);

```

---

```
void sclSetKernelArgs(sclSoft software, const char *sizesValues, ...);
void _sclVSetKernelArgs(sclSoft software, const char *sizesValues,
                        va_list argList);
```

```
/*
```

---

## 1.2.9 Hardware init and selection

---

```
*/
```

```
void sclGetHardwareByType(const cl_device_type device_type, const int
                        iDevice,
                        int *found, sclHard *hardware);
void sclGetGPUHardware(const int nDevice, int *found, sclHard
                        *GPUHardware);
void sclGetCPUHardware(const int nDevice, int *found, sclHard
                        *CPUHardware);
void sclGetAcceleratorHardware(const int iDevice, int *found, sclHard
                        *AcceleratorHardware);
//void sclGetAllHardware(int *found, sclHard *hardwareList);
void sclGetFastestDevice(const sclHard *hardList, const cl_int found,
                        sclHard
                        *fastest);
```

```
/*
```

---

## 1.3 INTERNAL FUNCITONS

### 1.3.1 debug

---

```
*/
```

```
void _sclWriteArgOnAFile(int argnum, void *arg, size_t size, const char
                        *diff);
```

```
/*
```

---

### 1.3.2 cl software management

---

```
*/
```

```
void _sclBuildProgram(cl_program program, cl_device_id devices,
                        const char *pName);
```

```
cl_kernel _sclCreateKernel(sclSoft software);
cl_program _sclCreateProgram(char *program_source, cl_context context);
char *_sclLoadProgramSource(const char *filename);
```

```
/*
```

---

### 1.3.3 hardware management

---

```
*/
```

```
int _sclGetMaxComputeUnits(cl_device_id device);
unsigned long int _sclGetMaxMemAllocSize(cl_device_id device);
int _sclGetDeviceType(cl_device_id device);
void _sclSmartCreateContexts(sclHard *hardList, cl_int found);
void _sclCreateQueues(sclHard *hardList, cl_int found);
```

```
/*
```

---

## Chapter 2

# simpleCl.c API

---

```
//void sclReleaseAllHardware(sclHard *hardList, cl_int found) {  
//  //  
//  //
```

---

```
//void sclGetAllHardware(int *found, sclHard *hardwareList) {  
//  //
```

---

```
void sclGetCLSoftware(const char *kernel_file, const char *kernel_name,  
                     const sclHard hardware, sclSoft *Software) {  
    //
```

---

```
cl_event sclManageArgsLaunchKernel(sclHard hardware, sclSoft software,  
                                   size_t *global_work_size,  
                                   size_t *local_work_size,  
                                   const char *sizesValues, ...) {  
    //
```

---

### 2.0.1 kernel argument assignement sizeTypes string's letter meanings

---

```
for (p = sizesValues; *p != '\0'; p++) {  
    if (*p == '%') {  
        switch (*++p) {  
            case 'a': /* Single value non pointer argument: byte length, array  
                       pointer  
                       */  
                //  
            //
```

---

---

```
case 'v': /* Buffer or image object void* argument: array pointer
*/
//
```

---

```
case 'N': /* Local memory object using NULL argument: byte length
*/
//
```

---

```
case 'w': /* output write_only cl_mem buffer: byte length, array
pointer*/
//
```

---

```
case 'r': /* input read_only cl_mem buffer: byte size, array
pointer */
//
```

---

```
case 'R': /* output read_write buffer: byte length, array pointer*/
//
```

---

```
case 'g': /* output read_write variable: bytesize */
//
```

---