An event is an object that communicates the status of OpenCL commands.
From the OpenCL Notes:
11. Enqueue the Kernel Object for Execution

```c
size_t globalWorkSize[3] = { NUM_ELEMENT, 1, 1 };
size_t localWorkSize[3] = { LOCAL_SIZE, 1, 1 };
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );
```

Creating an Event

```c
cl_event waitKernelC;
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelC );
```
cl_event waitKernelA, waitKernelB.

... cl_event dependencies[2];

dependencies[0] = waitKernelA;
dependencies[1] = waitKernelB;

status = clEnqueueNDRangeKernel( cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, NULL);

Creating an Execution Graph Structure

cl_event waitKernelC.

...

cl_event dependencies[2];

dependencies[0] = waitKernelA;
dependencies[1] = waitKernelB;

status = clEnqueueNDRangeKernel( cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, &waitKernelC);
cl_event waitKernelA, waitKernel B.

... 

status = clEnqueueNDRangeKernel( cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 1, &waitKernelA, NULL );

Placing a Barrier in the Command Queue

status = clEnqueueBarrier( cmdQueue );

Note: this cannot throw its own event

This does not complete until all commands enqueued before it have completed.
Placing an Event Marker in the Command Queue

```c
cl_event waitMarker;
status = clEnqueueMarker( cmdQueue, &waitMarker );
```

Note: this can throw its own event

This does not complete until all commands enqueued before it have completed.

This is just like a barrier, but it can throw an event to be waited for.

---

Waiting for Events Without Enqueuing Another Command

```c
status = clWaitForEvents( 2, dependencies );
```

This blocks until the specified events are thrown, so use it carefully!
I Like Synchronizing Things This Way

```c
void Wait( cl_command_queue queue )
{
    cl_event wait;
    cl_int status;

    status = clEnqueueMarker( queue, &wait );
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clEnqueueMarker failed\n" );

    status = clWaitForEvents( 1, &wait );  // blocks until everything is done!
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clWaitForEvents failed\n" );
}
```

Call this before starting the timer, before ending the timer, and before retrieving data from an array computed in an OpenCL program.

---

Getting Event Statuses Without Blocking

<table>
<thead>
<tr>
<th>Specification</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_EVENT_COMMAND_QUEUE</td>
<td>cl_int</td>
</tr>
<tr>
<td>CL_EVENT_CONTEXT</td>
<td>cl_int</td>
</tr>
<tr>
<td>CL_EVENT_COMMAND_TYPE</td>
<td>cl_int</td>
</tr>
<tr>
<td>CL_EVENT_COMMAND_EXECUTION_STATUS</td>
<td>cl_int</td>
</tr>
</tbody>
</table>

```c
cl_int eventStatus;

status = clGetEventInfo( waitKernelC, CL_EVENT_COMMAND_EXECUTION_STATUS, sizeof(cl_int), &eventStatus, NULL );
```

Specify one of these

- CL_QUEUE
- CL_SUBMITTED
- CL_RUNNING
- CL_COMPLETE

Note that this a nice way to check on event statuses without blocking. Thus, you could put this in a loop and go get some other work done in between calls.