



# CUPTI

DA-05679-001 \_v6.5 | August 2014

## User's Guide



CUPTI contains a number of enhancements as part of the CUDA Toolkit 6.5 release.

- ▶ Instruction classification is done for source-correlated Instruction Execution activity `CUpti_ActivityInstructionExecution`. See `CUpti_ActivityInstructionClass` for instruction classes.
- ▶ Two new device attributes are added to the activity `CUpti_DeviceAttribute`:
  - ▶ `CUPTI_DEVICE_ATTR_FLOP_SP_PER_CYCLE` gives peak single precision flop per cycle for the GPU.
  - ▶ `CUPTI_DEVICE_ATTR_FLOP_DP_PER_CYCLE` gives peak double precision flop per cycle for the GPU.
- ▶ Two new metric properties are added:
  - ▶ `CUPTI_METRIC_PROPERTY_FLOP_SP_PER_CYCLE` gives peak single precision flop per cycle for the GPU.
  - ▶ `CUPTI_METRIC_PROPERTY_FLOP_DP_PER_CYCLE` gives peak double precision flop per cycle for the GPU.
- ▶ Activity record `CUpti_ActivityGlobalAccess` for source level global access information has been deprecated and replaced by new activity record `CUpti_ActivityGlobalAccess2`. New record additionally gives information needed to map SASS assembly instructions to CUDA C source code. And it also provides ideal L2 transactions count based on the access pattern.
- ▶ Activity record `CUpti_ActivityBranch` for source level branch information has been deprecated and replaced by new activity record `CUpti_ActivityBranch2`. New record additionally gives information needed to map SASS assembly instructions to CUDA C source code.
- ▶ Sample `sass_source_map` is added to demonstrate the mapping of SASS assembly instructions to CUDA C source code.
- ▶ Default event collection mode is changed to Kernel (`CUPTI_EVENT_COLLECTION_MODE_KERNEL`) from Continuous (`CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS`). Also Continuous mode is now supported only on Tesla devices.
- ▶ Profiling results might be inconsistent when auto boost is enabled. Profiler tries to disable auto boost by default, it might fail to do so in some conditions, but profiling will continue. A new API `cuptiGetAutoBoostState` is added to query the auto boost state of the device. This API returns error `CUPTI_ERROR_NOT_SUPPORTED` on devices that don't support auto boost. Note that auto boost is supported only on certain Tesla devices from the Kepler+ family.

# TABLE OF CONTENTS

<b>Chapter 1. Usage</b> .....	<b>1</b>
1.1. CUPTI Compatibility and Requirements.....	1
1.2. CUPTI Initialization.....	1
1.3. CUPTI Activity API.....	1
1.3.1. Context Activity Record.....	2
1.3.2. Legacy Activity Records.....	3
1.4. CUPTI Callback API.....	3
1.4.1. Driver and Runtime API Callbacks.....	4
1.4.2. Resource Callbacks.....	5
1.4.3. Synchronization Callbacks.....	6
1.4.4. NVIDIA Tools Extension Callbacks.....	6
1.5. CUPTI Event API.....	7
1.5.1. Collecting Kernel Execution Events.....	9
1.5.2. Sampling Events.....	10
1.6. CUPTI Metric API.....	10
1.7. Samples.....	36
<b>Chapter 2. Modules</b> .....	<b>37</b>
2.1. CUPTI Version.....	37
cuptiGetVersion.....	37
CUPTI_API_VERSION.....	38
2.2. CUPTI Result Codes.....	38
CUptiResult.....	38
cuptiGetResultString.....	40
2.3. CUPTI Activity API.....	40
CUpti_Activity.....	41
CUpti_ActivityAPI.....	41
CUpti_ActivityAutoBoostState.....	41
CUpti_ActivityBranch.....	41
CUpti_ActivityBranch2.....	41
CUpti_ActivityCdpKernel.....	41
CUpti_ActivityContext.....	41
CUpti_ActivityDevice.....	41
CUpti_ActivityDeviceAttribute.....	41
CUpti_ActivityEnvironment.....	41
CUpti_ActivityEvent.....	41
CUpti_ActivityEventInstance.....	41
CUpti_ActivityFunction.....	41
CUpti_ActivityGlobalAccess.....	42
CUpti_ActivityGlobalAccess2.....	42
CUpti_ActivityInstructionExecution.....	42

CUpti_ActivityKernel.....	42
CUpti_ActivityKernel2.....	42
CUpti_ActivityMarker.....	42
CUpti_ActivityMarkerData.....	42
CUpti_ActivityMemcpy.....	42
CUpti_ActivityMemcpy2.....	42
CUpti_ActivityMemset.....	42
CUpti_ActivityMetric.....	42
CUpti_ActivityMetricInstance.....	42
CUpti_ActivityModule.....	43
CUpti_ActivityName.....	43
CUpti_ActivityObjectKindId.....	43
CUpti_ActivityOverhead.....	43
CUpti_ActivityPreemption.....	43
CUpti_ActivitySharedAccess.....	43
CUpti_ActivitySourceLocator.....	43
CUpti_ActivityUnifiedMemoryCounter.....	43
CUpti_ActivityUnifiedMemoryCounterConfig.....	43
CUpti_ActivityAttribute.....	43
CUpti_ActivityComputeApiKind.....	44
CUpti_ActivityEnvironmentKind.....	44
CUpti_ActivityFlag.....	45
CUpti_ActivityInstructionClass.....	46
CUpti_ActivityKind.....	48
CUpti_ActivityMemcpyKind.....	51
CUpti_ActivityMemoryKind.....	52
CUpti_ActivityObjectKind.....	52
CUpti_ActivityOverheadKind.....	53
CUpti_ActivityPreemptionKind.....	53
CUpti_ActivityUnifiedMemoryCounterKind.....	53
CUpti_ActivityUnifiedMemoryCounterScope.....	54
CUpti_EnvironmentClocksThrottleReason.....	54
CUpti_BuffersCallbackCompleteFunc.....	55
CUpti_BuffersCallbackRequestFunc.....	55
cuptiActivityConfigureUnifiedMemoryCounter.....	55
cuptiActivityDisable.....	56
cuptiActivityDisableContext.....	57
cuptiActivityEnable.....	57
cuptiActivityEnableContext.....	58
cuptiActivityFlush.....	59
cuptiActivityFlushAll.....	60
cuptiActivityGetAttribute.....	60
cuptiActivityGetNextRecord.....	61

cuptiActivityGetNumDroppedRecords.....	62
cuptiActivityRegisterCallbacks.....	63
cuptiActivitySetAttribute.....	64
cuptiGetAutoBoostState.....	64
cuptiGetContextId.....	65
cuptiGetDeviceId.....	66
cuptiGetStreamId.....	66
cuptiGetTimestamp.....	67
CUPTI_AUTO_BOOST_INVALID_CLIENT_PID.....	67
CUPTI_CORRELATION_ID_UNKNOWN.....	68
CUPTI_GRID_ID_UNKNOWN.....	68
CUPTI_SOURCE_LOCATOR_ID_UNKNOWN.....	68
CUPTI_TIMESTAMP_UNKNOWN.....	68
2.4. CUPTI Callback API.....	68
CUpti_CallbackData.....	68
CUpti_ModuleResourceData.....	68
CUpti_NvtxData.....	68
CUpti_ResourceData.....	68
CUpti_SynchronizeData.....	68
CUpti_ApiCallbackSite.....	68
CUpti_CallbackDomain.....	69
CUpti_CallbackIdResource.....	69
CUpti_CallbackIdSync.....	70
CUpti_CallbackFunc.....	70
CUpti_CallbackId.....	71
CUpti_DomainTable.....	71
CUpti_SubscriberHandle.....	71
cuptiEnableAllDomains.....	71
cuptiEnableCallback.....	72
cuptiEnableDomain.....	73
cuptiGetCallbackName.....	74
cuptiGetCallbackState.....	74
cuptiSubscribe.....	75
cuptiSupportedDomains.....	76
cuptiUnsubscribe.....	77
2.5. CUPTI Event API.....	77
CUpti_EventGroupSet.....	78
CUpti_EventGroupSets.....	78
CUpti_DeviceAttribute.....	78
CUpti_DeviceAttributeDeviceClass.....	79
CUpti_EventAttribute.....	79
CUpti_EventCategory.....	79
CUpti_EventCollectionMethod.....	80

CUpti_EventCollectionMode.....	80
CUpti_EventDomainAttribute.....	81
CUpti_EventGroupAttribute.....	81
CUpti_ReadEventFlags.....	82
CUpti_EventDomainID.....	82
CUpti_EventGroup.....	82
CUpti_EventID.....	82
cuptiDeviceEnumEventDomains.....	83
cuptiDeviceGetAttribute.....	84
cuptiDeviceGetEventDomainAttribute.....	84
cuptiDeviceGetNumEventDomains.....	86
cuptiDeviceGetTimestamp.....	86
cuptiDisableKernelReplayMode.....	87
cuptiEnableKernelReplayMode.....	87
cuptiEnumEventDomains.....	88
cuptiEventDomainEnumEvents.....	89
cuptiEventDomainGetAttribute.....	89
cuptiEventDomainGetNumEvents.....	91
cuptiEventGetAttribute.....	91
cuptiEventGetIdFromName.....	92
cuptiEventGroupAddEvent.....	93
cuptiEventGroupCreate.....	94
cuptiEventGroupDestroy.....	95
cuptiEventGroupDisable.....	96
cuptiEventGroupEnable.....	96
cuptiEventGroupGetAttribute.....	97
cuptiEventGroupReadAllEvents.....	98
cuptiEventGroupReadEvent.....	100
cuptiEventGroupRemoveAllEvents.....	101
cuptiEventGroupRemoveEvent.....	102
cuptiEventGroupResetAllEvents.....	103
cuptiEventGroupSetAttribute.....	103
cuptiEventGroupSetDisable.....	104
cuptiEventGroupSetEnable.....	105
cuptiEventGroupSetsCreate.....	106
cuptiEventGroupSetsDestroy.....	107
cuptiGetNumEventDomains.....	107
cuptiSetEventCollectionMode.....	108
CUPTI_EVENT_OVERFLOW.....	109
2.6. CUPTI Metric API.....	109
CUpti_MetricValue.....	109
CUpti_MetricAttribute.....	109
CUpti_MetricCategory.....	109

CUpti_MetricEvaluationMode.....	110
CUpti_MetricPropertyDeviceClass.....	110
CUpti_MetricPropertyID.....	111
CUpti_MetricValueKind.....	111
CUpti_MetricValueUtilizationLevel.....	112
CUpti_MetricID.....	112
cuptiDeviceEnumMetrics.....	112
cuptiDeviceGetNumMetrics.....	113
cuptiEnumMetrics.....	114
cuptiGetNumMetrics.....	114
cuptiMetricCreateEventGroupSets.....	115
cuptiMetricEnumEvents.....	116
cuptiMetricEnumProperties.....	116
cuptiMetricGetAttribute.....	117
cuptiMetricGetIdFromName.....	118
cuptiMetricGetNumEvents.....	119
cuptiMetricGetNumProperties.....	119
cuptiMetricGetRequiredEventGroupSets.....	120
cuptiMetricGetValue.....	121
cuptiMetricGetValue2.....	122
<b>Chapter 3. Data Structures.....</b>	<b>125</b>
CUpti_Activity.....	127
kind.....	127
CUpti_ActivityAPI.....	128
cbid.....	128
correlationId.....	128
end.....	128
kind.....	128
processId.....	128
returnValue.....	128
start.....	128
threadId.....	128
CUpti_ActivityAutoBoostState.....	129
enabled.....	129
pid.....	129
CUpti_ActivityBranch.....	129
correlationId.....	129
diverged.....	129
executed.....	129
kind.....	129
pcOffset.....	130
sourceLocatorId.....	130
threadsExecuted.....	130

CUpti_ActivityBranch2.....	130
correlationId.....	130
diverged.....	130
executed.....	130
functionId.....	130
kind.....	130
pad.....	130
pcOffset.....	131
sourceLocatorId.....	131
threadsExecuted.....	131
CUpti_ActivityCdpKernel.....	131
blockX.....	131
blockY.....	131
blockZ.....	131
completed.....	131
contextId.....	131
correlationId.....	131
deviceId.....	132
dynamicSharedMemory.....	132
end.....	132
executed.....	132
gridId.....	132
gridX.....	132
gridY.....	132
gridZ.....	132
kind.....	132
localMemoryPerThread.....	132
localMemoryTotal.....	133
name.....	133
parentBlockX.....	133
parentBlockY.....	133
parentBlockZ.....	133
parentGridId.....	133
queued.....	133
registersPerThread.....	133
requested.....	133
sharedMemoryConfig.....	133
start.....	134
staticSharedMemory.....	134
streamId.....	134
submitted.....	134
CUpti_ActivityContext.....	134
computeApiKind.....	134



contextId.....	134
deviceId.....	134
kind.....	134
nullStreamId.....	135
CUpti_ActivityDevice.....	135
computeCapabilityMajor.....	135
computeCapabilityMinor.....	135
constantMemorySize.....	135
coreClockRate.....	135
flags.....	135
globalMemoryBandwidth.....	135
globalMemorySize.....	135
id.....	136
kind.....	136
l2CacheSize.....	136
maxBlockDimX.....	136
maxBlockDimY.....	136
maxBlockDimZ.....	136
maxBlocksPerMultiprocessor.....	136
maxGridDimX.....	136
maxGridDimY.....	136
maxGridDimZ.....	136
maxIPC.....	136
maxRegistersPerBlock.....	137
maxSharedMemoryPerBlock.....	137
maxThreadsPerBlock.....	137
maxWarpsPerMultiprocessor.....	137
name.....	137
numMemcpyEngines.....	137
numMultiprocessors.....	137
numThreadsPerWarp.....	137
CUpti_ActivityDeviceAttribute.....	137
attribute.....	138
deviceId.....	138
flags.....	138
kind.....	138
value.....	138
CUpti_ActivityEnvironment.....	138
clocksThrottleReasons.....	139
cooling.....	139
deviceId.....	139
environmentKind.....	139
fanSpeed.....	139

gpuTemperature.....	139
kind.....	139
memoryClock.....	139
pcieLinkGen.....	139
pcieLinkWidth.....	139
power.....	140
power.....	140
powerLimit.....	140
smClock.....	140
speed.....	140
temperature.....	140
timestamp.....	140
CUpti_ActivityEvent.....	140
correlationId.....	141
domain.....	141
id.....	141
kind.....	141
value.....	141
CUpti_ActivityEventInstance.....	141
correlationId.....	141
domain.....	141
id.....	142
instance.....	142
kind.....	142
pad.....	142
value.....	142
CUpti_ActivityFunction.....	142
contextId.....	142
functionIndex.....	142
id.....	142
kind.....	142
moduleId.....	143
name.....	143
CUpti_ActivityGlobalAccess.....	143
correlationId.....	143
executed.....	143
flags.....	143
kind.....	143
l2_transactions.....	143
pcOffset.....	143
sourceLocatorId.....	143
threadsExecuted.....	144
CUpti_ActivityGlobalAccess2.....	144

correlationId.....	144
executed.....	144
flags.....	144
functionId.....	144
kind.....	144
l2_transactions.....	144
pad.....	144
pcOffset.....	144
sourceLocatorId.....	145
theoreticalL2Transactions.....	145
threadsExecuted.....	145
CUpti_ActivityInstructionExecution.....	145
correlationId.....	145
executed.....	145
flags.....	145
functionId.....	146
kind.....	146
notPredOffThreadsExecuted.....	146
pad.....	146
pcOffset.....	146
sourceLocatorId.....	146
threadsExecuted.....	146
CUpti_ActivityKernel.....	146
blockX.....	147
blockY.....	147
blockZ.....	147
cacheConfigExecuted.....	147
cacheConfigRequested.....	147
contextId.....	147
correlationId.....	147
deviceId.....	147
dynamicSharedMemory.....	147
end.....	147
gridX.....	148
gridY.....	148
gridZ.....	148
kind.....	148
localMemoryPerThread.....	148
localMemoryTotal.....	148
name.....	148
pad.....	148
registersPerThread.....	148
reserved0.....	148

runtimeCorrelationId.....	149
start.....	149
staticSharedMemory.....	149
streamId.....	149
CUpti_ActivityKernel2.....	149
blockX.....	149
blockY.....	149
blockZ.....	149
completed.....	149
contextId.....	150
correlationId.....	150
deviceId.....	150
dynamicSharedMemory.....	150
end.....	150
executed.....	150
gridId.....	150
gridX.....	150
gridY.....	150
gridZ.....	150
kind.....	151
localMemoryPerThread.....	151
localMemoryTotal.....	151
name.....	151
registersPerThread.....	151
requested.....	151
reserved0.....	151
sharedMemoryConfig.....	151
start.....	151
staticSharedMemory.....	151
streamId.....	152
CUpti_ActivityMarker.....	152
flags.....	152
id.....	152
kind.....	152
name.....	152
objectId.....	152
objectKind.....	152
timestamp.....	153
CUpti_ActivityMarkerData.....	153
category.....	153
color.....	153
flags.....	153
id.....	153

kind.....	153
payload.....	153
payloadKind.....	153
CUpti_ActivityMemcpy.....	154
bytes.....	154
contextId.....	154
copyKind.....	154
correlationId.....	154
deviceId.....	154
dstKind.....	154
end.....	155
flags.....	155
kind.....	155
reserved0.....	155
runtimeCorrelationId.....	155
srcKind.....	155
start.....	155
streamId.....	155
CUpti_ActivityMemcpy2.....	156
bytes.....	156
contextId.....	156
copyKind.....	156
correlationId.....	156
deviceId.....	156
dstContextId.....	156
dstDeviceId.....	156
dstKind.....	156
end.....	157
flags.....	157
kind.....	157
pad.....	157
reserved0.....	157
srcContextId.....	157
srcDeviceId.....	157
srcKind.....	157
start.....	158
streamId.....	158
CUpti_ActivityMemset.....	158
bytes.....	158
contextId.....	158
correlationId.....	158
deviceId.....	158
end.....	158

kind.....	158
reserved0.....	159
runtimeCorrelationId.....	159
start.....	159
streamId.....	159
value.....	159
CUpti_ActivityMetric.....	159
correlationId.....	159
flags.....	159
id.....	160
kind.....	160
pad.....	160
value.....	160
CUpti_ActivityMetricInstance.....	160
correlationId.....	160
flags.....	160
id.....	161
instance.....	161
kind.....	161
pad.....	161
value.....	161
CUpti_ActivityModule.....	161
contextId.....	161
cubin.....	161
cubinSize.....	161
id.....	161
kind.....	162
pad.....	162
CUpti_ActivityName.....	162
kind.....	162
name.....	162
objectId.....	162
objectKind.....	162
CUpti_ActivityObjectKindId.....	162
dcs.....	163
pt.....	163
CUpti_ActivityOverhead.....	163
end.....	163
kind.....	163
objectId.....	163
objectKind.....	163
overheadKind.....	164
start.....	164

CUpti_ActivityPreemption.....	164
blockX.....	164
blockY.....	164
blockZ.....	164
gridId.....	164
kind.....	164
pad.....	164
preemptionKind.....	165
timestamp.....	165
CUpti_ActivitySharedAccess.....	165
correlationId.....	165
executed.....	165
flags.....	165
functionId.....	165
kind.....	165
pad.....	165
pcOffset.....	166
sharedTransactions.....	166
sourceLocatorId.....	166
theoreticalSharedTransactions.....	166
threadsExecuted.....	166
CUpti_ActivitySourceLocator.....	166
fileName.....	166
id.....	166
kind.....	166
lineNumber.....	167
CUpti_ActivityUnifiedMemoryCounter.....	167
counterKind.....	167
deviceId.....	167
kind.....	167
pad.....	167
processId.....	167
scope.....	168
timestamp.....	168
value.....	168
CUpti_ActivityUnifiedMemoryCounterConfig.....	168
deviceId.....	168
enable.....	168
kind.....	168
scope.....	169
CUpti_CallbackData.....	169
callbackSite.....	169
context.....	169

contextUid.....	169
correlationData.....	169
correlationId.....	169
functionName.....	170
functionParams.....	170
functionReturnValue.....	170
symbolName.....	170
CUpti_EventGroupSet.....	170
eventGroups.....	170
numEventGroups.....	170
CUpti_EventGroupSets.....	171
numSets.....	171
sets.....	171
CUpti_MetricValue.....	171
CUpti_ModuleResourceData.....	171
cubinSize.....	171
moduleId.....	171
pCubin.....	172
CUpti_NvtxData.....	172
functionName.....	172
functionParams.....	172
CUpti_ResourceData.....	172
context.....	172
resourceDescriptor.....	173
stream.....	173
CUpti_SynchronizeData.....	173
context.....	173
stream.....	173
<b>Chapter 4. Data Fields.....</b>	<b>174</b>



## LIST OF TABLES

Table 1	Capability 1.x Metrics .....	12
Table 2	Capability 2.x Metrics .....	12
Table 3	Capability 3.x Metrics .....	20
Table 4	Capability 5.x Metrics .....	28



# Chapter 1.

## USAGE

The *CUDA Profiling Tools Interface* (CUPTI) enables the creation of profiling and tracing tools that target CUDA applications. CUPTI provides four APIs: *the Activity API*, the *Callback API*, the *Event API*, and the *Metric API*. Using these APIs, you can develop profiling tools that give insight into the CPU and GPU behavior of CUDA applications. CUPTI is delivered as a dynamic library on all platforms supported by CUDA.

### 1.1. CUPTI Compatibility and Requirements

New versions of the CUDA driver are backwards compatible with older versions of CUPTI. For example, a developer using a profiling tool based on CUPTI 4.1 can update to a more recently released CUDA driver. However, new versions of CUPTI are not backwards compatible with older versions of the CUDA driver. For example, a developer using a profiling tool based on CUPTI 4.1 must have a version of the CUDA driver released with CUDA Toolkit 4.1 (or later) installed as well. CUPTI calls will fail with `CUPTI_ERROR_NOT_INITIALIZED` if the CUDA driver version is not compatible with the CUPTI version.

### 1.2. CUPTI Initialization

CUPTI initialization occurs lazily the first time you invoke any CUPTI function. For the Activity, Event, Metric, and Callback APIs there are no requirements on when this initialization must occur (i.e. you can invoke the first CUPTI function at any point). See the CUPTI Activity API section for more information on CUPTI initialization requirements for the activity API.

### 1.3. CUPTI Activity API

The CUPTI Activity API allows you to asynchronously collect a trace of an application's CPU and GPU CUDA activity. The following terminology is used by the activity API.

## Activity Record

CPU and GPU activity is reported in C data structures called activity records. There is a different C structure type for each activity kind (e.g. `CUpti_ActivityMemcpy`). Records are generically referred to using the `CUpti_Activity` type. This type contains only a kind field that indicates the kind of the activity record. Using this kind, the object can be cast from the generic `CUpti_Activity` type to the specific type representing the activity. See the `printActivity` function in the [activity\\_trace\\_async](#) sample for an example.

## Activity Buffer

An activity buffer is used to transfer one or more activity records from CUPTI to the client. CUPTI fills activity buffers with activity records as the corresponding activities occur on the CPU and GPU. The CUPTI client is responsible for providing empty activity buffers as necessary to ensure that no records are dropped.

An *asynchronous* buffering API is implemented by `cuptiActivityRegisterCallbacks` and `cuptiActivityFlushAll`.

It is not required that the activity API be initialized before CUDA, but if the activity API is not initialized before CUDA some activity records may not be collected. You can force initialization of the activity API by enabling one or more activity kinds using `cuptiActivityEnable` or `cuptiActivityEnableContext`, as shown in the `initTrace` function of the [activity\\_trace\\_async](#) sample. Some activity kinds cannot be directly enabled, see the API documentation for `CUpti_ActivityKind` for details. Functions `cuptiActivityEnable` and `cuptiActivityEnableContext` will return `CUPTI_ERROR_NOT_COMPATIBLE` if the requested activity kind cannot be enabled.

The activity buffer API uses callbacks to request and return buffers of activity records. To use the asynchronous buffering API you must first register two callbacks using `cuptiActivityRegisterCallbacks`. One of these callbacks will be invoked whenever CUPTI needs an empty activity buffer. The other callback is used to deliver a buffer containing one or more activity records to the client. To minimize profiling overhead the client should return as quickly as possible from these callbacks. Function `cuptiActivityFlushAll` can be used to force CUPTI to deliver any activity buffers that contain completed activity records. Functions `cuptiActivityGetAttribute` and `cuptiActivitySetAttribute` can be used to read and write attributes that control how the buffering API behaves. See the API documentation for more information.

The [activity\\_trace\\_async](#) sample shows how to use the activity buffer API to collect a trace of CPU and GPU activity for a simple application.

### 1.3.1. Context Activity Record

In 6.0 the context activity record, `CUpti_ActivityContext`, was changed in a manner that introduced a new field into the structure. This new field was introduced in a way that preserves backward compatibility with any persisted versions of this structure.

The 32-bit `computeApiKind` field was replaced with two 16 bit fields, `computeApiKind` and `defaultStreamId`. Because all valid `computeApiKind` values fit within 16 bits, and because all supported CUDA platforms are little-endian, persisted context record data read with the new structure will have the correct value for `computeApiKind` and have a value of zero for `defaultStreamId`. The CUPTI client is responsible for versioning the persisted context data to recognize when the `defaultStreamId` field is valid.

### 1.3.2. Legacy Activity Records

In CUPTI 5.5 the `CUpti_ActivityKernel2` structure replaced `CUpti_ActivityKernel` as the activity record used for the `CUPTI_ACTIVITY_KIND_KERNEL` and `CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL` activity kinds. The `CUpti_ActivityKernel` definition is retained in CUPTI to enable newer versions of CUPTI to work with persisted activity record data.

The CUPTI client is responsible for versioning the persisted activity record data to recognize when the persisted data is stored using `CUpti_ActivityKernel` or `CUpti_ActivityKernel2`.

## 1.4. CUPTI Callback API

The CUPTI Callback API allows you to register a callback into your own code. Your callback will be invoked when the application being profiled calls a CUDA runtime or driver function, or when certain events occur in the CUDA driver. The following terminology is used by the callback API.

### Callback Domain

Callbacks are grouped into domains to make it easier to associate your callback functions with groups of related CUDA functions or events. There are currently four callback domains, as defined by `CUpti_CallbackDomain`: a domain for CUDA runtime functions, a domain for CUDA driver functions, a domain for CUDA resource tracking, and a domain for CUDA synchronization notification.

### Callback ID

Each callback is given a unique ID within the corresponding callback domain so that you can identify it within your callback function. The CUDA driver API IDs are defined in `cupti_driver_cbid.h` and the CUDA runtime API IDs are defined in `cupti_runtime_cbid.h`. Both of these headers are included for you when you include `cupti.h`. The CUDA resource callback IDs are defined by `CUpti_CallbackIdResource` and the CUDA synchronization callback IDs are defined by `CUpti_CallbackIdSync`.

### Callback Function

Your callback function must be of type `CUpti_CallbackFunc`. This function type has two arguments that specify the callback domain and ID so that you know why

the callback is occurring. The type also has a `cbdata` argument that is used to pass data specific to the callback.

### Subscriber

A subscriber is used to associate each of your callback functions with one or more CUDA API functions. There can be at most one subscriber initialized with `cuptiSubscribe()` at any time. Before initializing a new subscriber, the existing subscriber must be finalized with `cuptiUnsubscribe()`.

Each callback domain is described in detail below. Unless explicitly stated, it is not supported to call any CUDA runtime or driver API from within a callback function. Doing so may cause the application to hang.

## 1.4.1. Driver and Runtime API Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_DRIVER_API` or `CUPTI_CB_DOMAIN_RUNTIME_API` domains, you can associate a callback function with one or more CUDA API functions. When those CUDA functions are invoked in the application, your callback function is invoked as well. For these domains, the `cbdata` argument to your callback function will be of the type `CUpti_CallbackData`.

It is legal to call `cudaThreadSynchronize()`, `cudaDeviceSynchronize()`, `cudaStreamSynchronize()`, `cuCtxSynchronize()`, and `cuStreamSynchronize()` from within a driver or runtime API callback function.

The following code shows a typical sequence used to associate a callback function with one or more CUDA API functions. To simplify the presentation error checking code has been removed.

```
CUpti_SubscriberHandle subscriber;
MyDataStruct *my_data = ...;
...
cuptiSubscribe(&subscriber,
              (CUpti_CallbackFunc)my_callback , my_data);
cuptiEnableDomain(1, subscriber,
                 CUPTI_CB_DOMAIN_RUNTIME_API);
```

First, `cuptiSubscribe` is used to initialize a subscriber with the `my_callback` callback function. Next, `cuptiEnableDomain` is used to associate that callback with all the CUDA runtime API functions. Using this code sequence will cause `my_callback` to be called twice each time any of the CUDA runtime API functions are invoked, once on entry to the CUDA function and once just before exit from the CUDA function. CUPTI callback API functions `cuptiEnableCallback` and `cuptiEnableAllDomains` can also be used to associate CUDA API functions with a callback (see reference below for more information).

The following code shows a typical callback function.

```
void CUPTIAPI
my_callback(void *userdata, CUpti_CallbackDomain domain,
            CUpti_CallbackId cbid, const void *cbdata)
{
    const CUpti_CallbackData *cbInfo = (CUpti_CallbackData *)cbdata;
    MyDataStruct *my_data = (MyDataStruct *)userdata;

    if ((domain == CUPTI_CB_DOMAIN_RUNTIME_API) &&
        (cbid == CUPTI_RUNTIME_TRACE_CBID_cudaMemcpy_v3020)) {
        if (cbInfo->callbackSite == CUPTI_API_ENTER) {
            cudaMemcpy_v3020_params *funcParams =
                (cudaMemcpy_v3020_params *) (cbInfo->
                    functionParams);

            size_t count = funcParams->count;
            enum cudaMemcpyKind kind = funcParams->kind;
            ...
        }
    }
    ...
}
```

In your callback function, you use the `CUpti_CallbackDomain` and `CUpti_CallbackID` parameters to determine which CUDA API function invocation is causing this callback. In the example above, we are checking for the CUDA runtime `cudaMemcpy` function. The `cbdata` parameter holds a structure of useful information that can be used within the callback. In this case we use the `callbackSite` member of the structure to detect that the callback is occurring on entry to `cudaMemcpy`, and we use the `functionParams` member to access the parameters that were passed to `cudaMemcpy`. To access the parameters we first cast `functionParams` to a structure type corresponding to the `cudaMemcpy` function. These parameter structures are contained in `generated_cuda_runtime_api_meta.h`, `generated_cuda_meta.h`, and a number of other files. When possible these files are included for you by `cupti.h`.

The **callback\_event** and **callback\_timestamp** samples described on the [samples page](#) both show how to use the callback API for the driver and runtime API domains.

## 1.4.2. Resource Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_RESOURCE` domain, you can associate a callback function with some CUDA resource creation and destruction events. For example, when a CUDA context is created, your callback function will be invoked with a callback ID equal to `CUPTI_CBID_RESOURCE_CONTEXT_CREATED`. For this domain, the `cbdata` argument to your callback function will be of the type `CUpti_ResourceData`.

Note that, APIs `cuptiActivityFlush` and `cuptiActivityFlushAll` will result in deadlock when called from stream destroy starting callback identified using callback ID `CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING`.

### 1.4.3. Synchronization Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_SYNCHRONIZE` domain, you can associate a callback function with CUDA context and stream synchronizations. For example, when a CUDA context is synchronized, your callback function will be invoked with a callback ID equal to `CUPTI_CBID_SYNCHRONIZE_CONTEXT_SYNCHRONIZED`. For this domain, the `cbdata` argument to your callback function will be of the type `CUpti_SynchronizeData`.

### 1.4.4. NVIDIA Tools Extension Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_NVTX` domain, you can associate a callback function with NVIDIA Tools Extension (NVTX) API functions. When an NVTX function is invoked in the application, your callback function is invoked as well. For these domains, the `cbdata` argument to your callback function will be of the type `CUpti_NvtxData`.

The NVTX library has its own convention for discovering the profiling library that will provide the implementation of the NVTX callbacks. To receive callbacks you must set the NVTX environment variables appropriately so that when the application calls an NVTX function, your profiling library receive the callbacks. The following code sequence shows a typical initialization sequence to enable NVTX callbacks and activity records.

```
/* Set env so CUPTI-based profiling library loads on first nvtx call. */
char *inj32_path = "/path/to/32-bit/version/of/cupti/based/profiling/library";
char *inj64_path = "/path/to/64-bit/version/of/cupti/based/profiling/library";
setenv("NVTX_INJECTION32_PATH", inj32_path, 1);
setenv("NVTX_INJECTION64_PATH", inj64_path, 1);
```

The following code shows a typical sequence used to associate a callback function with one or more NVTX functions. To simplify the presentation error checking code has been removed.

```
CUpti_SubscriberHandle subscriber;
MyDataStruct *my_data = ...;
...
cuptiSubscribe(&subscriber,
              (CUpti_CallbackFunc)my_callback , my_data);
cuptiEnableDomain(1, subscriber,
                 CUPTI_CB_DOMAIN_NVTX);
```

First, `cuptiSubscribe` is used to initialize a subscriber with the `my_callback` callback function. Next, `cuptiEnableDomain` is used to associate that callback with all the NVTX functions. Using this code sequence will cause `my_callback` to be called once each time any of the NVTX functions are invoked. CUPTI callback API functions `cuptiEnableCallback` and `cuptiEnableAllDomains` can also be used to associate NVTX API functions with a callback (see reference below for more information).



The following code shows a typical callback function.

```
void CUPTI_API
my_callback(void *userdata, CUpti_CallbackDomain domain,
            CUpti_CallbackId cbid, const void *cbdata)
{
    const CUpti_NvtxData *nvtxInfo = (CUpti_NvtxData *)cbdata;
    MyDataStruct *my_data = (MyDataStruct *)userdata;

    if ((domain == CUPTI_CB_DOMAIN_NVTX) &&
        (cbid == NVTX_CBID_CORE_NameOsThreadA)) {
        nvtxNameOsThreadA_params *params = (nvtxNameOsThreadA_params *)nvtxInfo->
            functionParams;
        ...
    }
    ...
}
```

In your callback function, you use the `CUpti_CallbackDomain` and `CUpti_CallbackID` parameters to determine which NVTX API function invocation is causing this callback. In the example above, we are checking for the `nvtxNameOsThreadA` function. The `cbdata` parameter holds a structure of useful information that can be used within the callback. In this case, we use the `functionParams` member to access the parameters that were passed to `nvtxNameOsThreadA`. To access the parameters we first cast `functionParams` to a structure type corresponding to the `nvtxNameOsThreadA` function. These parameter structures are contained in `generated_nvtx_meta.h`.

## 1.5. CUPTI Event API

The CUPTI Event API allows you to query, configure, start, stop, and read the event counters on a CUDA-enabled device. The following terminology is used by the event API.

### Event

An event is a countable activity, action, or occurrence on a device.

### Event ID

Each event is assigned a unique identifier. A named event will represent the same activity, action, or occurrence on all device types. But the named event may have different IDs on different device families. Use `cuptiEventGetIdFromName` to get the ID for a named event on a particular device.

### Event Category

Each event is placed in one of the categories defined by `CUpti_EventCategory`. The category indicates the general type of activity, action, or occurrence measured by the event.

### Event Domain

A device exposes one or more event domains. Each event domain represents a group of related events available on that device. A device may have multiple instances of a domain, indicating that the device can simultaneously record multiple instances of each event within that domain.

## Event Group

An event group is a collection of events that are managed together. The number and type of events that can be added to an event group are subject to device-specific limits. At any given time, a device may be configured to count events from a limited number of event groups. All events in an event group must belong to the same event domain.

## Event Group Set

An event group set is a collection of event groups that can be enabled at the same time. Event group sets are created by `cuptiEventGroupSetsCreate` and `cuptiMetricCreateEventGroupSets`.

You can determine the events available on a device using the `cuptiDeviceEnumEventDomains` and `cuptiEventDomainEnumEvents` functions. The `cupti_query` sample described on the [samples page](#) shows how to use these functions. You can also enumerate all the CUPTI events available on any device using the `cuptiEnumEventDomains` function.

Configuring and reading event counts requires the following steps. First, select your event collection mode. If you want to count events that occur during the execution of a kernel, use `cuptiSetEventCollectionMode` to set mode `CUPTI_EVENT_COLLECTION_MODE_KERNEL`. If you want to continuously sample the event counts, use mode `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS`. Next determine the names of the events that you want to count, and then use the `cuptiEventGroupCreate`, `cuptiEventGetIdFromName`, and `cuptiEventGroupAddEvent` functions to create and initialize an event group with those events. If you are unable to add all the events to a single event group then you will need to create multiple event groups. Alternatively, you can use the `cuptiEventGroupSetsCreate` function to automatically create the event group(s) required for a set of events.

To begin counting a set of events, enable the event group or groups that contain those events by using the `cuptiEventGroupEnable` function. If your events are contained in multiple event groups you may be unable to enable all of the event groups at the same time, due to device limitations. In this case, you can gather the events across multiple executions of the application or you can enable kernel replay. If you enable kernel replay using `cuptiEnableKernelReplayMode` you will be able to enable any number of event groups and all the contained events will be collect.

Use the `cuptiEventGroupReadEvent` and/or `cuptiEventGroupReadAllEvents` functions to read the event values. When you are done collecting events, use the `cuptiEventGroupDisable` function to stop counting of the events contained in an event group. The `callback_event` sample described on the [samples page](#) shows how to use these functions to create, enable, and disable event groups, and how to read event counts.

## 1.5.1. Collecting Kernel Execution Events

A common use of the event API is to count a set of events during the execution of a kernel (as demonstrated by the `callback_event` sample). The following code shows a typical callback used for this purpose. Assume that the callback was enabled only for a kernel launch using the CUDA runtime (i.e. by `cuptiEnableCallback(1, subscriber, CUPTI_CB_DOMAIN_RUNTIME_API, CUPTI_RUNTIME_TRACE_CBID_cudaLaunch_v3020)`). To simplify the presentation error checking code has been removed.

```
static void CUPTIAPI
getEventValueCallback(void *userdata,
                     CUpti_CallbackDomain domain,
                     CUpti_CallbackId cbid,
                     const void *cbdata)
{
    const CUpti_CallbackData *cbData =
        (CUpti_CallbackData *)cbdata;

    if (cbData->callbackSite == CUPTI_API_ENTER) {
        cudaDeviceSynchronize();
        cuptiSetEventCollectionMode(cbInfo->context,
                                   CUPTI_EVENT_COLLECTION_MODE_KERNEL);
        cuptiEventGroupEnable(eventGroup);
    }

    if (cbData->callbackSite == CUPTI_API_EXIT) {
        cudaDeviceSynchronize();
        cuptiEventGroupReadEvent(eventGroup,
                                 CUPTI_EVENT_READ_FLAG_NONE,
                                 eventId,
                                 &bytesRead, &eventVal);

        cuptiEventGroupDisable(eventGroup);
    }
}
```

Two synchronization points are used to ensure that events are counted only for the execution of the kernel. If the application contains other threads that launch kernels, then additional thread-level synchronization must also be introduced to ensure that those threads do not launch kernels while the callback is collecting events. When the `cudaLaunch` API is entered (that is, before the kernel is actually launched on the device), `cudaDeviceSynchronize` is used to wait until the GPU is idle. The event collection mode is set to `CUPTI_EVENT_COLLECTION_MODE_KERNEL` so that the event counters are automatically started and stopped just before and after the kernel executes. Then event collection is enabled with `cuptiEventGroupEnable`.

When the `cudaLaunch` API is exited (that is, after the kernel is queued for execution on the GPU) another `cudaDeviceSynchronize` is used to cause the CPU thread to wait for the kernel to finish execution. Finally, the event counts are read with `cuptiEventGroupReadEvent`.

## 1.5.2. Sampling Events

The event API can also be used to sample event values while a kernel or kernels are executing (as demonstrated by the **event\_sampling** sample). The sample shows one possible way to perform the sampling. The event collection mode is set to `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS` so that the event counters run continuously. Two threads are used in **event\_sampling**: one thread schedules the kernels and memcpys that perform the computation, while another thread wakes periodically to sample an event counter. In this sample there is no correlation of the event samples with what is happening on the GPU. To get some coarse correlation, you can use `cuptiDeviceGetTimestamp` to collect the GPU timestamp at the time of the sample and also at other interesting points in your application.

## 1.6. CUPTI Metric API

The CUPTI Metric API allows you to collect application metrics calculated from one or more event values. The following terminology is used by the metric API.

### **Metric**

An characteristic of an application that is calculated from one or more event values.

### **Metric ID**

Each metric is assigned a unique identifier. A named metric will represent the same characteristic on all device types. But the named metric may have different IDs on different device families. Use `cuptiMetricGetIdFromName` to get the ID for a named metric on a particular device.

### **Metric Category**

Each metric is placed in one of the categories defined by `CUpti_MetricCategory`. The category indicates the general type of the characteristic measured by the metric.

### **Metric Property**

Each metric is calculated from input values. These input values can be events or properties of the device or system. The available properties are defined by `CUpti_MetricPropertyID`.

### **Metric Value**

Each metric has a value that represents one of the kinds defined by `CUpti_MetricValueKind`. For each value kind, there is a corresponding member of the `CUpti_MetricValue` union that is used to hold the metric's value.

The tables included in this section list the metrics available for each device, as determined by the device's compute capability. You can also determine the metrics available on a device using the `cuptiDeviceEnumMetrics` function. The **cupti\_query** sample described on the [samples page](#) shows how to use this function. You can also enumerate all the CUPTI metrics available on any device using the `cuptiEnumMetrics` function.

CUPTI provides two functions for calculating a metric value. `cuptiMetricGetValue2` can be used to calculate a metric value when the device is not available. All required event values and metric properties must be provided by the caller. `cuptiMetricGetValue` can be used to calculate a metric value when the device is available (as a `CUdevice` object). All required event values must be provided by the caller but CUPTI will determine the appropriate property values from the `CUdevice` object.

Configuring and calculating metric values requires the following steps. First, determine the name of the metric that you want to collect, and then use the `cuptiMetricGetIdFromName` to get the metric ID. Use `cuptiMetricEnumEvents` to get the events required to calculate the metric and follow instructions in the CUPTI Event API section to create the event groups for those events. When creating event groups in this manner it is important to use the result of `cuptiMetricGetRequiredEventGroupSets` to properly group together events that must be collected in the same pass to ensure proper metric calculation.

Alternatively, you can use the `cuptiMetricCreateEventGroupSets` function to automatically create the event group(s) required for metric's events. When using this function events will be grouped as required to most accurately calculate the metric, as a result it is not necessary to use `cuptiMetricGetRequiredEventGroupSets`.

If you are using `cuptiMetricGetValue2` the you must also collect the required metric property values using `cuptiMetricEnumProperties`.

Collect event counts as described in the CUPTI Event API section, and then use either `cuptiMetricGetValue` or `cuptiMetricGetValue2` to calculate the metric value from the collected event and property values. The **callback\_metric** sample described on the [samples page](#) shows how to use the functions to calculate event values and calculate a metric using `cuptiMetricGetValue`. Note that, as shown in the example, you should collect event counts from all domain instances and normalize the counts to get the most accurate metric values. It is necessary to normalize the event counts because the number of event counter instances varies by device and by the event being counted.

For example, a device might have 8 multiprocessors but only have event counters for 4 of the multiprocessors, and might have 3 memory units and only have events counters for one memory unit. When calculating a metric that requires a multiprocessor event and a memory unit event, the 4 multiprocessor counters should be summed and multiplied by 2 to normalize the event count across the entire device. Similarly, the one memory unit counter should be multiplied by 3 to normalize the event count across the entire device. The normalized values can then be passed to `cuptiMetricGetValue` or `cuptiMetricGetValue2` to calculate the metric value.

As described, the normalization assumes the kernel executes a sufficient number of blocks to completely load the device. If the kernel has only a small number of blocks, normalizing across the entire device may skew the result.

### Metric Reference - Compute Capability 1.x

Devices with compute capability less than 2.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 1 Capability 1.x Metrics

Metric Name	Description	Scope
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Single-context
gld_efficiency	Ratio of requested global memory load transactions to actual global memory load transactions expressed as percentage	Single-context
gst_efficiency	Ratio of requested global memory store transactions to actual global memory store transactions expressed as percentage	Single-context
gld_requested_throughput	Requested global memory load throughput	Single-context
gst_requested_throughput	Requested global memory store throughput	Single-context

### Metric Reference - Compute Capability 2.x

Devices with compute capability between 2.0, inclusive, and 3.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 2 Capability 2.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context
alu_fu_utilization	The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions on a scale of 0 to 10	Multi-context

Metric Name	Description	Scope
atomic_replay_overhead	Average number of replays due to atomic and reduction bank conflicts for each instruction executed	Multi-context
atomic_throughput	Global memory atomic and reduction throughput	Multi-context
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context
cf_issued	Number of issued control-flow instructions	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context



Metric Name	Description	Scope
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_cache_replay_overhead	Average number of replays due to global memory cache misses for each instruction executed	Single-context
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context

Metric Name	Description	Scope
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
ipc_instance	Instructions executed per cycle for a single multiprocessor	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l1_cache_global_hit_rate	Hit rate in L1 cache for global loads	Single-context
l1_cache_local_hit_rate	Hit rate in L1 cache for local loads and stores	Single-context
l1_shared_utilization	The utilization level of the L1/shared memory relative to peak utilization on a scale of 0 to 10	Single-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Single-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_l1_read_hit_rate	Hit rate at L2 cache for all read requests from L1 cache	Single-context
l2_l1_read_throughput	Memory read throughput seen at L2 cache for read requests from L1 cache	Single-context
l2_l1_read_transactions	Memory read transactions seen at L2 cache for all read requests from L1 cache	Single-context

Metric Name	Description	Scope
l2_l1_write_throughput	Memory write throughput seen at L2 cache for write requests from L1 cache	Single-context
l2_l1_write_transactions	Memory write transactions seen at L2 cache for all write requests from L1 cache	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_texture_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_texture_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Single-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context

Metric Name	Description	Scope
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage	Single-context
local_replay_overhead	Average number of replays due to local memory accesses for each instruction executed	Single-context
local_store_throughput	Local memory store throughput	Single-context
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_replay_overhead	Average number of replays due to shared memory conflicts for each instruction executed	Single-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU	Single-context
sm_efficiency_instance	The percentage of time at least one warp is active on a specific multiprocessor	Single-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being	Multi-context

Metric Name	Description	Scope
	available or fully utilized, or because too many requests of a given type are outstanding	
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a __syncthreads() call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
systemem_read_throughput	System memory read throughput	Single-context
systemem_read_transactions	System memory read transactions	Single-context
systemem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10	Single-context
systemem_write_throughput	System memory write throughput	Single-context
systemem_write_transactions	System memory write transactions	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp	Multi-context

Metric Name	Description	Scope
	supported on a multiprocessor expressed as percentage	

### Metric Reference - Compute Capability 3.x

Devices with compute capability between 3.0, inclusive, and 4.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 3 Capability 3.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context
alu_fu_utilization	The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions on a scale of 0 to 10	Multi-context
atomic_replay_overhead	Average number of replays due to atomic and reduction bank conflicts for each instruction executed	Multi-context
atomic_throughput	Global memory atomic and reduction throughput	Multi-context
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage. This is available for compute capability 3.0.	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context

Metric Name	Description	Scope
cf_issued	Number of issued control-flow instructions	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions expressed as percentage	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_cache_replay_overhead	Average number of replays due to global memory cache misses for each instruction executed	Multi-context
global_replay_overhead	Average number of replays due to global memory cache misses	Multi-context



Metric Name	Description	Scope
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context

Metric Name	Description	Scope
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
ipc_instance	Instructions executed per cycle for a single multiprocessor	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l1_cache_global_hit_rate	Hit rate in L1 cache for global loads	Single-context
l1_cache_local_hit_rate	Hit rate in L1 cache for local loads and stores	Single-context
l1_shared_utilization	The utilization level of the L1/shared memory relative to peak utilization on a scale of 0 to 10. This is available for compute capability 3.0 and 3.5.	Single-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Single-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_l1_read_hit_rate	Hit rate at L2 cache for all read requests from L1 cache. This is available for compute capability 3.0 and 3.5.	Single-context
l2_l1_read_throughput	Memory read throughput seen at L2 cache for read requests from L1 cache. This is available for compute capability 3.0 and 3.5.	Single-context
l2_l1_read_transactions	Memory read transactions seen at L2 cache for all read requests from L1 cache	Single-context
l2_l1_write_throughput	Memory write throughput seen at L2 cache for write requests from L1 cache	Single-context
l2_l1_write_transactions	Memory write transactions seen at L2 cache for all write requests from L1 cache	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context

Metric Name	Description	Scope
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_texture_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_texture_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Single-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage. This is available for compute capability 3.0 and 3.5.	Single-context
local_replay_overhead	Average number of replays due to local memory accesses for each instruction executed	Multi-context
local_store_throughput	Local memory store throughput	Single-context

Metric Name	Description	Scope
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
nc_cache_global_hit_rate	Hit rate in non coherent cache for global loads	Single-context
nc_gld_efficiency	Ratio of requested non coherent global memory load throughput to required non coherent global memory load throughput expressed as percentage	Single-context
nc_gld_requested_throughput	Requested throughput for global memory loaded via non-coherent cache	Multi-context
nc_gld_throughput	Non coherent global memory load throughput	Single-context
nc_l2_read_throughput	Memory read throughput for non coherent global read requests seen at L2 cache	Single-context
nc_l2_read_transactions	Memory read transactions seen at L2 cache for non coherent global read requests	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_replay_overhead	Average number of replays due to shared memory conflicts for each instruction executed	Multi-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context

Metric Name	Description	Scope
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU	Single-context
sm_efficiency_instance	The percentage of time at least one warp is active on a specific multiprocessor	Single-context
stall_compute	Percentage of stalls occurring because a compute operation cannot be performed due to the required resources not being available	Multi-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being available or fully utilized, or because too many requests of a given type are outstanding	Multi-context
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_imc	Percentage of stalls occurring because of immediate constant cache miss	Multi-context
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a <code>__syncthreads()</code> call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
systemem_read_throughput	System memory read throughput. This is available for compute capability 3.0 and 3.5.	Single-context
systemem_read_transactions	System memory read transactions. This is available for compute capability 3.0 and 3.5.	Single-context
systemem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10. This is available for compute capability 3.0 and 3.5.	Single-context

Metric Name	Description	Scope
systemem_write_throughput	System memory write throughput. This is available for compute capability 3.0 and 3.5.	Single-context
systemem_write_transactions	System memory write transactions. This is available for compute capability 3.0 and 3.5.	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context
warp_nonpred_execution_efficiency	Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context

### Metric Reference - Compute Capability 5.x

Devices with compute capability greater than or equal to 5.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 4 Capability 5.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context

Metric Name	Description	Scope
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context
cf_issued	Number of issued control-flow instructions	Multi-context
double_precision_fu_utilization	The utilization level of the multiprocessor function units that execute double-precision floating-point instructions and integer instructions on a scale of 0 to 10	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context



Metric Name	Description	Scope
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_hit_rate	Hit rate for global loads	Single-context
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context

Metric Name	Description	Scope
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Multi-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_tex_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Single-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_tex_write_hit_rate	Hit Rate at L2 cache for all write requests from texture cache	Single-context
l2_tex_write_throughput	Memory write throughput seen at L2 cache for write requests from the texture cache	Single-context

Metric Name	Description	Scope
l2_tex_write_transactions	Memory write transactions seen at L2 cache for write requests from the texture cache	Single-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_hit_rate	Hit rate for local loads and stores	Single-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage	Single-context
local_store_throughput	Local memory store throughput	Single-context
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context

Metric Name	Description	Scope
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context
shared_utilization	The utilization level of the shared memory relative to peak utilization on a scale of 0 to 10	Single-context
single_precision_fu_utilization	The utilization level of the multiprocessor function units that execute single-precision floating-point instructions and integer instructions on a scale of 0 to 10	Multi-context
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor	Single-context
special_fu_utilization	The utilization level of the multiprocessor function units that execute sin, cos, ex2, popc, flo, and similar instructions on a scale of 0 to 10	Multi-context
stall_compute	Percentage of stalls occurring because a compute operation cannot be performed due to the required resources not being available	Multi-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being available or fully utilized, or because too many requests of a given type are outstanding	Multi-context
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_imc	Percentage of stalls occurring because of immediate constant cache miss	Multi-context

Metric Name	Description	Scope
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a __syncthreads() call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
sysmem_read_throughput	System memory read throughput	Single-context
sysmem_read_transactions	System memory read transactions	Single-context
sysmem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10	Single-context
sysmem_write_throughput	System memory write throughput	Single-context
sysmem_write_transactions	System memory write transactions	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context
warp_nonpred_execution_efficiency	Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor	Multi-context

## 1.7. Samples

The CUPTI installation includes several samples that demonstrate the use of the CUPTI APIs. The samples are:

### **activity\_trace\_async**

This sample shows how to collect a trace of CPU and GPU activity using the new asynchronous activity buffer APIs.

### **callback\_event**

This sample shows how to use both the callback and event APIs to record the events that occur during the execution of a simple kernel. The sample shows the required ordering for synchronization, and for event group enabling, disabling and reading.

### **callback\_metric**

This sample shows how to use both the callback and metric APIs to record the metric's events during the execution of a simple kernel, and then use those events to calculate the metric value.

### **callback\_timestamp**

This sample shows how to use the callback API to record a trace of API start and stop times.

### **cupti\_query**

This sample shows how to query CUDA-enabled devices for their event domains, events, and metrics.

### **event\_sampling**

This sample shows how to use the event API to sample events using a separate host thread.

### **sass\_source\_map**

This sample shows how to generate CUpti\_ActivityInstructionExecution records and how to map SASS assembly instructions to CUDA C source.

### **unified\_memory**

This sample shows how to collect various counters like page faults and page transfers for unified memory.

# Chapter 2.

## MODULES

Here is a list of all modules:

- ▶ CUPTI Version
- ▶ CUPTI Result Codes
- ▶ CUPTI Activity API
- ▶ CUPTI Callback API
- ▶ CUPTI Event API
- ▶ CUPTI Metric API

### 2.1. CUPTI Version

Function and macro to determine the CUPTI version.

**CUptiResult** `cuptiGetVersion (uint32_t *version)`

Get the CUPTI API version.

#### Parameters

##### **version**

Returns the version

#### Returns

- ▶ `CUPTI_SUCCESS`  
on success
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`  
if `version` is `NULL`

**Description**

Return the API version in `*version`.

**See also:**

[CUPTI\\_API\\_VERSION](#)

**#define CUPTI\_API\_VERSION 6**

The API version for this implementation of CUPTI.

The API version for this implementation of CUPTI. This define along with [cuptiGetVersion](#) can be used to dynamically detect if the version of CUPTI compiled against matches the version of the loaded CUPTI library.

v1 : CUDAToolsSDK 4.0 v2 : CUDAToolsSDK 4.1 v3 : CUDA Toolkit 5.0 v4 : CUDA Toolkit 5.5 v5 : CUDA Toolkit 6.0 v6 : CUDA Toolkit 6.5

## 2.2. CUPTI Result Codes

Error and result codes returned by CUPTI functions.

**enum CUptiResult**

CUPTI result codes.

Error and result codes returned by CUPTI functions.

**Values**

**CUPTI\_SUCCESS = 0**

No error.

**CUPTI\_ERROR\_INVALID\_PARAMETER = 1**

One or more of the parameters is invalid.

**CUPTI\_ERROR\_INVALID\_DEVICE = 2**

The device does not correspond to a valid CUDA device.

**CUPTI\_ERROR\_INVALID\_CONTEXT = 3**

The context is NULL or not valid.

**CUPTI\_ERROR\_INVALID\_EVENT\_DOMAIN\_ID = 4**

The event domain id is invalid.

**CUPTI\_ERROR\_INVALID\_EVENT\_ID = 5**

The event id is invalid.

**CUPTI\_ERROR\_INVALID\_EVENT\_NAME = 6**

The event name is invalid.

**CUPTI\_ERROR\_INVALID\_OPERATION = 7**

The current operation cannot be performed due to dependency on other factors.



**CUPTI\_ERROR\_OUT\_OF\_MEMORY = 8**

Unable to allocate enough memory to perform the requested operation.

**CUPTI\_ERROR\_HARDWARE = 9**

An error occurred on the performance monitoring hardware.

**CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT = 10**

The output buffer size is not sufficient to return all requested data.

**CUPTI\_ERROR\_API\_NOT\_IMPLEMENTED = 11**

API is not implemented.

**CUPTI\_ERROR\_MAX\_LIMIT\_REACHED = 12**

The maximum limit is reached.

**CUPTI\_ERROR\_NOT\_READY = 13**

The object is not yet ready to perform the requested operation.

**CUPTI\_ERROR\_NOT\_COMPATIBLE = 14**

The current operation is not compatible with the current state of the object

**CUPTI\_ERROR\_NOT\_INITIALIZED = 15**

CUPTI is unable to initialize its connection to the CUDA driver.

**CUPTI\_ERROR\_INVALID\_METRIC\_ID = 16**

The metric id is invalid.

**CUPTI\_ERROR\_INVALID\_METRIC\_NAME = 17**

The metric name is invalid.

**CUPTI\_ERROR\_QUEUE\_EMPTY = 18**

The queue is empty.

**CUPTI\_ERROR\_INVALID\_HANDLE = 19**

Invalid handle (internal?).

**CUPTI\_ERROR\_INVALID\_STREAM = 20**

Invalid stream.

**CUPTI\_ERROR\_INVALID\_KIND = 21**

Invalid kind.

**CUPTI\_ERROR\_INVALID\_EVENT\_VALUE = 22**

Invalid event value.

**CUPTI\_ERROR\_DISABLED = 23**

CUPTI is disabled due to conflicts with other enabled profilers

**CUPTI\_ERROR\_INVALID\_MODULE = 24**

Invalid module.

**CUPTI\_ERROR\_INVALID\_METRIC\_VALUE = 25**

Invalid metric value.

**CUPTI\_ERROR\_HARDWARE\_BUSY = 26**

The performance monitoring hardware is in use by other client.

**CUPTI\_ERROR\_NOT\_SUPPORTED = 27**

The attempted operation is not supported on the current system or device.

**CUPTI\_ERROR\_UNKNOWN = 999**

An unknown internal error has occurred.

**CUPTI\_ERROR\_FORCE\_INT = 0x7fffffff**

## CUptiResult cuptiGetString (CUptiResult result, const char \*\*str)

Get the descriptive string for a CUptiResult.

### Parameters

#### result

The result to get the string for

#### str

Returns the string

### Returns

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `str` is NULL or `result` is not a valid CUptiResult

### Description

Return the descriptive string for a CUptiResult in `*str`.



**Thread-safety:** this function is thread safe.

## 2.3. CUPTI Activity API

Functions, types, and enums that implement the CUPTI Activity API.

## **struct CUpti\_Activity**

The base activity record.

## **struct CUpti\_ActivityAPI**

The activity record for a driver or runtime API invocation.

## **struct CUpti\_ActivityAutoBoostState**

Device auto boost state structure.

## **struct CUpti\_ActivityBranch**

The activity record for source level result branch. (deprecated).

## **struct CUpti\_ActivityBranch2**

The activity record for source level result branch.

## **struct CUpti\_ActivityCdpKernel**

The activity record for CDP (CUDA Dynamic Parallelism) kernel.

## **struct CUpti\_ActivityContext**

The activity record for a context.

## **struct CUpti\_ActivityDevice**

The activity record for a device.

## **struct CUpti\_ActivityDeviceAttribute**

The activity record for a device attribute.

## **struct CUpti\_ActivityEnvironment**

The activity record for CUPTI environmental data.

## **struct CUpti\_ActivityEvent**

The activity record for a CUPTI event.

## **struct CUpti\_ActivityEventInstance**

The activity record for a CUPTI event with instance information.

## **struct CUpti\_ActivityFunction**

The activity record for global/device functions.

## struct CUpti\_ActivityGlobalAccess

The activity record for source-level global access. (deprecated).

## struct CUpti\_ActivityGlobalAccess2

The activity record for source-level global access.

## struct CUpti\_ActivityInstructionExecution

The activity record for source-level sass/source line-by-line correlation.

## struct CUpti\_ActivityKernel

The activity record for kernel. (deprecated).

## struct CUpti\_ActivityKernel2

The activity record for a kernel (CUDA 5.5 onwards).

## struct CUpti\_ActivityMarker

The activity record providing a marker which is an instantaneous point in time.

## struct CUpti\_ActivityMarkerData

The activity record providing detailed information for a marker.

## struct CUpti\_ActivityMemcpy

The activity record for memory copies.

## struct CUpti\_ActivityMemcpy2

The activity record for peer-to-peer memory copies.

## struct CUpti\_ActivityMemset

The activity record for memset.

## struct CUpti\_ActivityMetric

The activity record for a CUPTI metric.

## struct CUpti\_ActivityMetricInstance

The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (CUPTI\_ACTIVITY\_KIND\_METRIC\_INSTANCE). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect metric data may choose to use this type to

store the collected metric data. This activity record should be used when metric domain instance information needs to be associated with the metric.

## struct CUpti\_ActivityModule

The activity record for a CUDA module.

## struct CUpti\_ActivityName

The activity record providing a name.

## union CUpti\_ActivityObjectKindId

Identifiers for object kinds as specified by CUpti\_ActivityObjectKind.

## struct CUpti\_ActivityOverhead

The activity record for CUPTI and driver overheads.

## struct CUpti\_ActivityPreemption

The activity record for a preemption of a CDP kernel.

## struct CUpti\_ActivitySharedAccess

The activity record for source-level shared access.

## struct CUpti\_ActivitySourceLocator

The activity record for source locator.

## struct CUpti\_ActivityUnifiedMemoryCounter

The activity record for Unified Memory counters.

## struct CUpti\_ActivityUnifiedMemoryCounterConfig

Unified Memory counters configuration structure.

## enum CUpti\_ActivityAttribute

Activity attributes.

These attributes are used to control the behavior of the activity API.

### Values

#### CUPTI\_ACTIVITY\_ATTR\_DEVICE\_BUFFER\_SIZE = 0

The device memory size (in bytes) reserved for storing profiling data for non-CDP operations for each buffer on a context. The value is a `size_t`. Having larger buffer size means less flush operations but consumes more device memory. Having smaller buffer size increases the risk of dropping timestamps for kernel records if too many kernels are launched/replayed at one time. This value only applies to new buffer

allocations. Set this value before initializing CUDA or before creating a context to ensure it is considered for the following allocations. The default value is 4194304 (4MB). Note: The actual amount of device memory per buffer reserved by CUPTI might be larger.

**CUPTI\_ACTIVITY\_ATTR\_DEVICE\_BUFFER\_SIZE\_CDP = 1**

The device memory size (in bytes) reserved for storing profiling data for CDP operations for each buffer on a context. The value is a `size_t`. Having larger buffer size means less flush operations but consumes more device memory. This value only applies to new allocations. Set this value before initializing CUDA or before creating a context to ensure it is considered for the following allocations. The default value is 8388608 (8MB). Note: The actual amount of device memory per context reserved by CUPTI might be larger.

**CUPTI\_ACTIVITY\_ATTR\_DEVICE\_BUFFER\_POOL\_LIMIT = 2**

The maximum number of memory buffers per context. The value is a `size_t`. Buffers can be reused by the context. Increasing this value reduces the times CUPTI needs to flush the buffers. Setting this value will not modify the number of memory buffers currently stored. Set this value before initializing CUDA to ensure the limit is not exceeded. The default value is 4.

**CUPTI\_ACTIVITY\_ATTR\_DEVICE\_BUFFER\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityComputeApiKind

The kind of a compute API.

### Values

**CUPTI\_ACTIVITY\_COMPUTE\_API\_UNKNOWN = 0**

The compute API is not known.

**CUPTI\_ACTIVITY\_COMPUTE\_API\_CUDA = 1**

The compute APIs are for CUDA.

**CUPTI\_ACTIVITY\_COMPUTE\_API\_CUDA\_MPS = 2**

The compute APIs are for CUDA running in MPS (Multi-Process Service) environment.

**CUPTI\_ACTIVITY\_COMPUTE\_API\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityEnvironmentKind

The kind of environment data. Used to indicate what type of data is being reported by an environment activity record.

### Values

**CUPTI\_ACTIVITY\_ENVIRONMENT\_UNKNOWN = 0**

Unknown data.

**CUPTI\_ACTIVITY\_ENVIRONMENT\_SPEED = 1**

The environment data is related to speed.

**CUPTI\_ACTIVITY\_ENVIRONMENT\_TEMPERATURE = 2**

The environment data is related to temperature.

**CUPTI\_ACTIVITY\_ENVIRONMENT\_POWER = 3**

The environment data is related to power.

**CUPTI\_ACTIVITY\_ENVIRONMENT\_COOLING = 4**

The environment data is related to cooling.

**CUPTI\_ACTIVITY\_ENVIRONMENT\_COUNT**

**CUPTI\_ACTIVITY\_ENVIRONMENT\_KIND\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityFlag

Flags associated with activity records.

Activity record flags. Flags can be combined by bitwise OR to associated multiple flags with an activity record. Each flag is specific to a certain activity kind, as noted below.

### Values

**CUPTI\_ACTIVITY\_FLAG\_NONE = 0**

Indicates the activity record has no flags.

**CUPTI\_ACTIVITY\_FLAG\_DEVICE\_CONCURRENT\_KERNELS = 1<<0**

Indicates the activity represents a device that supports concurrent kernel execution.

Valid for CUPTI\_ACTIVITY\_KIND\_DEVICE.

**CUPTI\_ACTIVITY\_FLAG\_DEVICE\_ATTRIBUTE\_CUDEVICE = 1<<0**

Indicates if the activity represents a CUdevice\_attribute value or a CUpti\_DeviceAttribute value. Valid for

CUPTI\_ACTIVITY\_KIND\_DEVICE\_ATTRIBUTE.

**CUPTI\_ACTIVITY\_FLAG\_MEMCPY\_ASYNC = 1<<0**

Indicates the activity represents an asynchronous memcpy operation. Valid for

CUPTI\_ACTIVITY\_KIND\_MEMCPY.

**CUPTI\_ACTIVITY\_FLAG\_MARKER\_INSTANTANEOUS = 1<<0**

Indicates the activity represents an instantaneous marker. Valid for

CUPTI\_ACTIVITY\_KIND\_MARKER.

**CUPTI\_ACTIVITY\_FLAG\_MARKER\_START = 1<<1**

Indicates the activity represents a region start marker. Valid for

CUPTI\_ACTIVITY\_KIND\_MARKER.

**CUPTI\_ACTIVITY\_FLAG\_MARKER\_END = 1<<2**

Indicates the activity represents a region end marker. Valid for

CUPTI\_ACTIVITY\_KIND\_MARKER.

**CUPTI\_ACTIVITY\_FLAG\_MARKER\_COLOR\_NONE = 1<<0**

Indicates the activity represents a marker that does not specify a color. Valid for

CUPTI\_ACTIVITY\_KIND\_MARKER\_DATA.

**CUPTI\_ACTIVITY\_FLAG\_MARKER\_COLOR\_ARGB = 1<<1**

Indicates the activity represents a marker that specifies a color in alpha-red-green-blue format. Valid for `CUPTI_ACTIVITY_KIND_MARKER_DATA`.

**CUPTI\_ACTIVITY\_FLAG\_GLOBAL\_ACCESS\_KIND\_SIZE\_MASK = 0xFF<<0**

The number of bytes requested by each thread Valid for

`CUpti_ActivityGlobalAccess2`.

**CUPTI\_ACTIVITY\_FLAG\_GLOBAL\_ACCESS\_KIND\_LOAD = 1<<8**

If bit in this flag is set, the access was load, else it is a store access. Valid for

`CUpti_ActivityGlobalAccess2`.

**CUPTI\_ACTIVITY\_FLAG\_GLOBAL\_ACCESS\_KIND\_CACHED = 1<<9**

If this bit in flag is set, the load access was cached else it is uncached. Valid for

`CUpti_ActivityGlobalAccess2`.

**CUPTI\_ACTIVITY\_FLAG\_METRIC\_OVERFLOWED = 1<<0**

If this bit in flag is set, the metric value overflowed. Valid for `CUpti_ActivityMetric` and `CUpti_ActivityMetricInstance`.

**CUPTI\_ACTIVITY\_FLAG\_METRIC\_VALUE\_INVALID = 1<<1**

If this bit in flag is set, the metric value couldn't be calculated. This occurs when a value(s) required to calculate the metric is missing. Valid for `CUpti_ActivityMetric` and `CUpti_ActivityMetricInstance`.

**CUPTI\_ACTIVITY\_FLAG\_INSTRUCTION\_VALUE\_INVALID = 1<<0**

If this bit in flag is set, the source level metric value couldn't be calculated. This occurs when a value(s) required to calculate the source level metric cannot be evaluated.

Valid for `CUpti_ActivityInstructionExecution`.

**CUPTI\_ACTIVITY\_FLAG\_INSTRUCTION\_CLASS\_MASK = 0xFF<<1**

The mask for the instruction class, `CUpti_ActivityInstructionClass` Valid for

`CUpti_ActivityInstructionExecution`.

**CUPTI\_ACTIVITY\_FLAG\_FLUSH\_FORCED = 1<<0**

When calling `cuptiActivityFlushAll`, this flag can be set to force CUPTI to flush all records in the buffer, whether finished or not

**CUPTI\_ACTIVITY\_FLAG\_SHARED\_ACCESS\_KIND\_SIZE\_MASK = 0xFF<<0**

The number of bytes requested by each thread Valid for `CUpti_ActivitySharedAccess`.

**CUPTI\_ACTIVITY\_FLAG\_SHARED\_ACCESS\_KIND\_LOAD = 1<<8**

If bit in this flag is set, the access was load, else it is a store access. Valid for

`CUpti_ActivitySharedAccess`.

**CUPTI\_ACTIVITY\_FLAG\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityInstructionClass

SASS instruction classification.

The sass instruction are broadly divided into different class. Each enum represents a classification.

### Values

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_UNKNOWN = 0**



The instruction class is not known.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_FP\_32 = 1**

Represents a 32 bit floating point operation.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_FP\_64 = 2**

Represents a 64 bit floating point operation.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_INTEGER = 3**

Represents an integer operation.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_BIT\_CONVERSION = 4**

Represents a bit conversion operation.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_CONTROL\_FLOW = 5**

Represents a control flow instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_GLOBAL = 6**

Represents a global load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_SHARED = 7**

Represents a shared load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_LOCAL = 8**

Represents a local load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_GENERIC = 9**

Represents a generic load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_SURFACE = 10**

Represents a surface load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_CONSTANT = 11**

Represents a constant load instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_TEXTURE = 12**

Represents a texture load-store instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_GLOBAL\_ATOMIC = 13**

Represents a global atomic instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_SHARED\_ATOMIC = 14**

Represents a shared atomic instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_SURFACE\_ATOMIC = 15**

Represents a surface atomic instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_INTER\_THREAD\_COMMUNICATION = 16**

Represents a inter-thread communication instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_BARRIER = 17**

Represents a barrier instruction.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_MISCELLANEOUS = 18**

Represents some miscellaneous instructions which do not fit in the above classification.

**CUPTI\_ACTIVITY\_INSTRUCTION\_CLASS\_KIND\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityKind

The kinds of activity records.

Each activity record kind represents information about a GPU or an activity occurring on a CPU or GPU. Each kind is associated with a activity record structure that holds the information associated with the kind.

### See also:

[CUpti\\_Activity](#)

[CUpti\\_ActivityAPI](#)

[CUpti\\_ActivityContext](#)

[CUpti\\_ActivityDevice](#)

[CUpti\\_ActivityDeviceAttribute](#)

[CUpti\\_ActivityEvent](#)

[CUpti\\_ActivityEventInstance](#)

[CUpti\\_ActivityKernel](#)

[CUpti\\_ActivityKernel2](#)

[CUpti\\_ActivityCdpKernel](#)

[CUpti\\_ActivityPreemption](#)

[CUpti\\_ActivityMemcpy](#)

[CUpti\\_ActivityMemcpy2](#)

[CUpti\\_ActivityMemset](#)

[CUpti\\_ActivityMetric](#)

[CUpti\\_ActivityMetricInstance](#)

[CUpti\\_ActivityName](#)

[CUpti\\_ActivityMarker](#)

[CUpti\\_ActivityMarkerData](#)

[CUpti\\_ActivitySourceLocator](#)

[CUpti\\_ActivityGlobalAccess](#)

[CUpti\\_ActivityGlobalAccess2](#)

[CUpti\\_ActivityBranch](#)

[CUpti\\_ActivityBranch2](#)

[CUpti\\_ActivityOverhead](#)  
[CUpti\\_ActivityEnvironment](#)  
[CUpti\\_ActivityInstructionExecution](#)  
[CUpti\\_ActivityUnifiedMemoryCounter](#)  
[CUpti\\_ActivityFunction](#)  
[CUpti\\_ActivityModule](#)  
[CUpti\\_ActivitySharedAccess](#)

## Values

**CUPTI\_ACTIVITY\_KIND\_INVALID = 0**

The activity record is invalid.

**CUPTI\_ACTIVITY\_KIND\_MEMCPY = 1**

A host->host, host->device, or device->device memory copy. The corresponding activity record structure is [CUpti\\_ActivityMemcpy](#).

**CUPTI\_ACTIVITY\_KIND\_MEMSET = 2**

A memory set executing on the GPU. The corresponding activity record structure is [CUpti\\_ActivityMemset](#).

**CUPTI\_ACTIVITY\_KIND\_KERNEL = 3**

A kernel executing on the GPU. The corresponding activity record structure is [CUpti\\_ActivityKernel2](#).

**CUPTI\_ACTIVITY\_KIND\_DRIVER = 4**

A CUDA driver API function execution. The corresponding activity record structure is [CUpti\\_ActivityAPI](#).

**CUPTI\_ACTIVITY\_KIND\_RUNTIME = 5**

A CUDA runtime API function execution. The corresponding activity record structure is [CUpti\\_ActivityAPI](#).

**CUPTI\_ACTIVITY\_KIND\_EVENT = 6**

An event value. The corresponding activity record structure is [CUpti\\_ActivityEvent](#).

**CUPTI\_ACTIVITY\_KIND\_METRIC = 7**

A metric value. The corresponding activity record structure is [CUpti\\_ActivityMetric](#).

**CUPTI\_ACTIVITY\_KIND\_DEVICE = 8**

Information about a device. The corresponding activity record structure is [CUpti\\_ActivityDevice](#).

**CUPTI\_ACTIVITY\_KIND\_CONTEXT = 9**

Information about a context. The corresponding activity record structure is [CUpti\\_ActivityContext](#).

**CUPTI\_ACTIVITY\_KIND\_CONCURRENT\_KERNEL = 10**

A (potentially concurrent) kernel executing on the GPU. The corresponding activity record structure is [CUpti\\_ActivityKernel2](#).

**CUPTI\_ACTIVITY\_KIND\_NAME = 11**

- Thread, device, context, etc. name. The corresponding activity record structure is [CUpti\\_ActivityName](#).
- CUPTI\_ACTIVITY\_KIND\_MARKER = 12**  
Instantaneous, start, or end marker. The corresponding activity record structure is [CUpti\\_ActivityMarker](#).
- CUPTI\_ACTIVITY\_KIND\_MARKER\_DATA = 13**  
Extended, optional, data about a marker. The corresponding activity record structure is [CUpti\\_ActivityMarkerData](#).
- CUPTI\_ACTIVITY\_KIND\_SOURCE\_LOCATOR = 14**  
Source information about source level result. The corresponding activity record structure is [CUpti\\_ActivitySourceLocator](#).
- CUPTI\_ACTIVITY\_KIND\_GLOBAL\_ACCESS = 15**  
Results for source-level global access. The corresponding activity record structure is [CUpti\\_ActivityGlobalAccess2](#).
- CUPTI\_ACTIVITY\_KIND\_BRANCH = 16**  
Results for source-level branch. The corresponding activity record structure is [CUpti\\_ActivityBranch2](#).
- CUPTI\_ACTIVITY\_KIND\_OVERHEAD = 17**  
Overhead activity records. The corresponding activity record structure is [CUpti\\_ActivityOverhead](#).
- CUPTI\_ACTIVITY\_KIND\_CDP\_KERNEL = 18**  
A CDP (CUDA Dynamic Parallel) kernel executing on the GPU. The corresponding activity record structure is [CUpti\\_ActivityCdpKernel](#). This activity can not be directly enabled or disabled. It is enabled and disabled through concurrent kernel activity [CUPTI\\_ACTIVITY\\_KIND\\_CONCURRENT\\_KERNEL](#)
- CUPTI\_ACTIVITY\_KIND\_PREEMPTION = 19**  
Preemption activity record indicating a preemption of a CDP (CUDA Dynamic Parallel) kernel executing on the GPU. The corresponding activity record structure is [CUpti\\_ActivityPreemption](#).
- CUPTI\_ACTIVITY\_KIND\_ENVIRONMENT = 20**  
Environment activity records indicating power, clock, thermal, etc. levels of the GPU. The corresponding activity record structure is [CUpti\\_ActivityEnvironment](#).
- CUPTI\_ACTIVITY\_KIND\_EVENT\_INSTANCE = 21**  
An event value associated with a specific event domain instance. The corresponding activity record structure is [CUpti\\_ActivityEventInstance](#).
- CUPTI\_ACTIVITY\_KIND\_MEMCPY2 = 22**  
A peer to peer memory copy. The corresponding activity record structure is [CUpti\\_ActivityMemcpy2](#).
- CUPTI\_ACTIVITY\_KIND\_METRIC\_INSTANCE = 23**  
A metric value associated with a specific metric domain instance. The corresponding activity record structure is [CUpti\\_ActivityMetricInstance](#).
- CUPTI\_ACTIVITY\_KIND\_INSTRUCTION\_EXECUTION = 24**

SASS/Source line-by-line correlation record. The corresponding activity record structure is `CUpti_ActivityInstructionExecution`.

**CUPTI\_ACTIVITY\_KIND\_UNIFIED\_MEMORY\_COUNTER = 25**

Unified Memory counter record. The corresponding activity record structure is `CUpti_ActivityUnifiedMemoryCounter`.

**CUPTI\_ACTIVITY\_KIND\_FUNCTION = 26**

Device global/function record. The corresponding activity record structure is `CUpti_ActivityFunction`.

**CUPTI\_ACTIVITY\_KIND\_MODULE = 27**

CUDA Module record. The corresponding activity record structure is `CUpti_ActivityModule`.

**CUPTI\_ACTIVITY\_KIND\_DEVICE\_ATTRIBUTE = 28**

A device attribute value. The corresponding activity record structure is `CUpti_ActivityDeviceAttribute`.

**CUPTI\_ACTIVITY\_KIND\_SHARED\_ACCESS = 29**

Results for source-level shared access. The corresponding activity record structure is `CUpti_ActivitySharedAccess`.

**CUPTI\_ACTIVITY\_KIND\_FORCE\_INT = 0x7fffffff**

## enum `CUpti_ActivityMemcpyKind`

The kind of a memory copy, indicating the source and destination targets of the copy.

Each kind represents the source and destination targets of a memory copy. Targets are host, device, and array.

### Values

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_UNKNOWN = 0**

The memory copy kind is not known.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_HTOD = 1**

A host to device memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_DTOH = 2**

A device to host memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_HTOA = 3**

A host to device array memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_ATOH = 4**

A device array to host memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_ATOA = 5**

A device array to device array memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_ATOD = 6**

A device array to device memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_DTOA = 7**

A device to device array memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_DTOD = 8**

A device to device memory copy on the same device.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_HTOH = 9**

A host to host memory copy.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_PTOP = 10**

A peer to peer memory copy across different devices.

**CUPTI\_ACTIVITY\_MEMCPY\_KIND\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityMemoryKind

The kinds of memory accessed by a memory copy.

Each kind represents the type of the source or destination memory accessed by a memory copy.

### Values

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_UNKNOWN = 0**

The source or destination memory kind is unknown.

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_PAGEABLE = 1**

The source or destination memory is pageable.

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_PINNED = 2**

The source or destination memory is pinned.

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_DEVICE = 3**

The source or destination memory is on the device.

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_ARRAY = 4**

The source or destination memory is an array.

**CUPTI\_ACTIVITY\_MEMORY\_KIND\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ActivityObjectKind

The kinds of activity objects.

**See also:**

[CUpti\\_ActivityObjectKindId](#)

### Values

**CUPTI\_ACTIVITY\_OBJECT\_UNKNOWN = 0**

The object kind is not known.

**CUPTI\_ACTIVITY\_OBJECT\_PROCESS = 1**

A process.

**CUPTI\_ACTIVITY\_OBJECT\_THREAD = 2**

A thread.

**CUPTI\_ACTIVITY\_OBJECT\_DEVICE = 3**

A device.

**CUPTI\_ACTIVITY\_OBJECT\_CONTEXT = 4**

A context.

`CUPTI_ACTIVITY_OBJECT_STREAM = 5`

A stream.

`CUPTI_ACTIVITY_OBJECT_FORCE_INT = 0x7fffffff`

## enum CUpti\_ActivityOverheadKind

The kinds of activity overhead.

### Values

`CUPTI_ACTIVITY_OVERHEAD_UNKNOWN = 0`

The overhead kind is not known.

`CUPTI_ACTIVITY_OVERHEAD_DRIVER_COMPILER = 1`

Compiler(JIT) overhead.

`CUPTI_ACTIVITY_OVERHEAD_CUPTI_BUFFER_FLUSH = 1<<16`

Activity buffer flush overhead.

`CUPTI_ACTIVITY_OVERHEAD_CUPTI_INSTRUMENTATION = 2<<16`

CUPTI instrumentation overhead.

`CUPTI_ACTIVITY_OVERHEAD_CUPTI_RESOURCE = 3<<16`

CUPTI resource creation and destruction overhead.

`CUPTI_ACTIVITY_OVERHEAD_FORCE_INT = 0x7fffffff`

## enum CUpti\_ActivityPreemptionKind

The kind of a preemption activity.

### Values

`CUPTI_ACTIVITY_PREEMPTION_KIND_UNKNOWN = 0`

The preemption kind is not known.

`CUPTI_ACTIVITY_PREEMPTION_KIND_SAVE = 1`

Preemption to save CDP block.

`CUPTI_ACTIVITY_PREEMPTION_KIND_RESTORE = 2`

Preemption to restore CDP block.

`CUPTI_ACTIVITY_PREEMPTION_KIND_FORCE_INT = 0x7fffffff`

## enum CUpti\_ActivityUnifiedMemoryCounterKind

Kind of the Unified Memory counter.

Many activities are associated with Unified Memory mechanism; among them are transfer from host to device, device to host, page fault at host side.

### Values

`CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_UNKNOWN = 0`

The unified memory counter kind is not known.

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_KIND\_BYTES\_TRANSFER\_HTOD**  
= 1

Number of bytes transfered from host to device

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_KIND\_BYTES\_TRANSFER\_DTOH**  
= 2

Number of bytes transferred from device to host

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_KIND\_CPU\_PAGE\_FAULT\_COUNT**  
= 3

Number of CPU page faults

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_KIND\_COUNT**  
**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_KIND\_FORCE\_INT =**  
**0x7fffffff**

## enum CUpti\_ActivityUnifiedMemoryCounterScope

Scope of the unified memory counter.

### Values

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_SCOPE\_UNKNOWN = 0**

The unified memory counter scope is not known.

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_SCOPE\_PROCESS\_SINGLE\_DEVICE**  
= 1

Collect unified memory counter for single process on one device

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_SCOPE\_PROCESS\_ALL\_DEVICES**  
= 2

Collect unified memory counter for single process across all devices

**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_SCOPE\_COUNT**  
**CUPTI\_ACTIVITY\_UNIFIED\_MEMORY\_COUNTER\_SCOPE\_FORCE\_INT =**  
**0x7fffffff**

## enum CUpti\_EnvironmentClocksThrottleReason

Reasons for clock throttling.

The possible reasons that a clock can be throttled. There can be more than one reason that a clock is being throttled so these types can be combined by bitwise OR. These are used in the clocksThrottleReason field in the Environment Activity Record.

### Values

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_GPU\_IDLE = 0x00000001**

Nothing is running on the GPU and the clocks are dropping to idle state.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_USER\_DEFINED\_CLOCKS = 0x00000002**

The GPU clocks are limited by a user specified limit.



**CUPTI\_CLOCKS\_THROTTLE\_REASON\_SW\_POWER\_CAP = 0x00000004**

A software power scaling algorithm is reducing the clocks below requested clocks.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_HW\_SLOWDOWN = 0x00000008**

Hardware slowdown to reduce the clock by a factor of two or more is engaged. This is an indicator of one of the following: 1) Temperature is too high, 2) External power brake assertion is being triggered (e.g. by the system power supply), 3) Change in power state.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_UNKNOWN = 0x80000000**

Some unspecified factor is reducing the clocks.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_UNSUPPORTED = 0x40000000**

Throttle reason is not supported for this GPU.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_NONE = 0x00000000**

No clock throttling.

**CUPTI\_CLOCKS\_THROTTLE\_REASON\_FORCE\_INT = 0x7fffffff**

```
typedef (*CUpti_BuffersCallbackCompleteFunc)
(CUcontext context, uint32_t streamId, uint8_t* buffer,
size_t size, size_t validSize)
```

Function type for callback used by CUPTI to return a buffer of activity records.

This callback function returns to the CUPTI client a buffer containing activity records. The buffer contains `validSize` bytes of activity records which should be read using `cuptiActivityGetNextRecord`. The number of dropped records can be read using `cuptiActivityGetNumDroppedRecords`. After this call CUPTI relinquished ownership of the buffer and will not use it anymore. The client may return the buffer to CUPTI using the `CUpti_BuffersCallbackRequestFunc` callback. Note: CUDA 6.0 onwards, all buffers returned by this callback are global buffers i.e. there is no context/stream specific buffer. User needs to parse the global buffer to extract the context/stream specific activity records.

```
typedef (*CUpti_BuffersCallbackRequestFunc) (uint8_t*
*buffer, size_t* size, size_t* maxNumRecords)
```

Function type for callback used by CUPTI to request an empty buffer for storing activity records.

This callback function signals the CUPTI client that an activity buffer is needed by CUPTI. The activity buffer is used by CUPTI to store activity records. The callback function can decline the request by setting `*buffer` to NULL. In this case CUPTI may drop activity records.

## CUptiResult

### cuptiActivityConfigureUnifiedMemoryCounter

## `(CUpti_ActivityUnifiedMemoryCounterConfig *config, uint32_t count)`

Set Unified Memory Counter configuration.

### Parameters

#### **config**

A pointer to `CUpti_ActivityUnifiedMemoryCounterConfig` structures containing Unified Memory counter configuration.

#### **count**

Number of Unified Memory counter configuration structures

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`
  - if `config` is `NULL` or any parameter in the `config` structures is not a valid value
- ▶ `CUPTI_ERROR_NOT_SUPPORTED`

Indicates that the system/device does not support the unified memory counters

## `CUptiResult cuptiActivityDisable (CUpti_ActivityKind kind)`

Disable collection of a specific kind of activity record.

### Parameters

#### **kind**

The kind of activity record to stop collecting

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_KIND`

if the activity kind is not supported

### Description

Disable collection of a specific kind of activity record. Multiple kinds can be disabled by calling this function multiple times. By default all activity kinds are disabled for collection.

## CUptiResult cuptiActivityDisableContext (CUcontext context, CUpti\_ActivityKind kind)

Disable collection of a specific kind of activity record for a context.

### Parameters

#### context

The context for which activity is to be disabled

#### kind

The kind of activity record to stop collecting

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_KIND  
if the activity kind is not supported

### Description

Disable collection of a specific kind of activity record for a context. This setting done by this API will supersede the global settings for activity records. Multiple kinds can be enabled by calling this function multiple times.

## CUptiResult cuptiActivityEnable (CUpti\_ActivityKind kind)

Enable collection of a specific kind of activity record.

### Parameters

#### kind

The kind of activity record to collect

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_NOT\_COMPATIBLE  
if the activity kind cannot be enabled
- ▶ CUPTI\_ERROR\_INVALID\_KIND

if the activity kind is not supported

### Description

Enable collection of a specific kind of activity record. Multiple kinds can be enabled by calling this function multiple times. By default all activity kinds are disabled for collection.

## CUptiResult cuptiActivityEnableContext (CUcontext context, CUpti\_ActivityKind kind)

Enable collection of a specific kind of activity record for a context.

### Parameters

#### context

The context for which activity is to be enabled

#### kind

The kind of activity record to collect

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_NOT\_COMPATIBLE
  - if the activity kind cannot be enabled
- ▶ CUPTI\_ERROR\_INVALID\_KIND
  - if the activity kind is not supported

### Description

Enable collection of a specific kind of activity record for a context. This setting done by this API will supersede the global settings for activity records enabled by [cuptiActivityEnable](#). Multiple kinds can be enabled by calling this function multiple times.

## CUptiResult cuptiActivityFlush (CUcontext context, uint32\_t streamId, uint32\_t flag)

Wait for all activity records are delivered via the completion callback.

### Parameters

#### context

A valid CUcontext or NULL.

#### streamId

The stream ID.

#### flag

The flag can be set to indicate a forced flush. See CUpti\_ActivityFlag

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_CUPTI\_ERROR\_INVALID\_OPERATION
  - if not preceded by a successful call to cuptiActivityRegisterCallbacks
- ▶ CUPTI\_ERROR\_UNKNOWN
  - an internal error occurred

### Description

This function does not return until all activity records associated with the specified context/stream are returned to the CUPTI client using the callback registered in cuptiActivityRegisterCallbacks. To ensure that all activity records are complete, the requested stream(s), if any, are synchronized.

If `context` is NULL, the global activity records (i.e. those not associated with a particular stream) are flushed (in this case no streams are synchronized). If `context` is a valid CUcontext and `streamId` is 0, the buffers of all streams of this context are flushed. Otherwise, the buffers of the specified stream in this context is flushed.

Before calling this function, the buffer handling callback api must be activated by calling cuptiActivityRegisterCallbacks.

**\*\*DEPRECATED\*\*** This method is deprecated CONTEXT and STREAMID will be ignored. Use cuptiActivityFlushAll to flush all data.

## CUptiResult cuptiActivityFlushAll (uint32\_t flag)

Wait for all activity records are delivered via the completion callback.

### Parameters

#### flag

The flag can be set to indicate a forced flush. See CUpti\_ActivityFlag

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if not preceded by a successful call to cuptiActivityRegisterCallbacks
- ▶ CUPTI\_ERROR\_UNKNOWN
  - an internal error occurred

### Description

This function does not return until all activity records associated with all contexts/streams (and the global buffers not associated with any stream) are returned to the CUPTI client using the callback registered in cuptiActivityRegisterCallbacks. To ensure that all activity records are complete, the requested stream(s), if any, are synchronized.

Before calling this function, the buffer handling callback api must be activated by calling cuptiActivityRegisterCallbacks.

## CUptiResult cuptiActivityGetAttribute (CUpti\_ActivityAttribute attr, size\_t \*valueSize, void \*value)

Read an activity API attribute.

### Parameters

#### attr

The attribute to read

#### valueSize

Size of buffer pointed by the value, and returns the number of bytes written to value

#### value

Returns the value of the attribute

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `valueSize` or `value` is NULL, or if `attr` is not an activity attribute
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - Indicates that the `value` buffer is too small to hold the attribute value.

**Description**

Read an activity API attribute and return it in `*value`.

## CUptiResult cuptiActivityGetNextRecord (uint8\_t \*buffer, size\_t validBufferSizeBytes, CUpti\_Activity \*\*record)

Iterate over the activity records in a buffer.

**Parameters****buffer**

The buffer containing activity records

**validBufferSizeBytes**

The number of valid bytes in the buffer.

**record**

Inputs the previous record returned by `cuptiActivityGetNextRecord` and returns the next activity record from the buffer. If input value is NULL, returns the first activity record in the buffer. Records of kind `CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL` may contain invalid (0) timestamps, indicating that no timing information could be collected for lack of device memory.

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_MAX\_LIMIT\_REACHED
  - if no more records in the buffer
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `buffer` is NULL.

## Description

This is a helper function to iterate over the activity records in a buffer. A buffer of activity records is typically obtained by using the `cuptiActivityDequeueBuffer()` function or by receiving a `CUpti_BuffersCallbackCompleteFunc` callback.

An example of typical usage:

```
↑ CUpti_Activity *record = NULL;
   CUptiResult status = CUPTI_SUCCESS;
   do {
       status = cuptiActivityGetNextRecord(buffer, validSize, &record);
       if(status == CUPTI_SUCCESS) {
           // Use record here...
       }
       else if (status == CUPTI_ERROR_MAX_LIMIT_REACHED)
           break;
       else {
           goto Error;
       }
   } while (1);
```

## CUptiResult cuptiActivityGetNumDroppedRecords (CUcontext context, uint32\_t streamId, size\_t \*dropped)

Get the number of activity records that were dropped of insufficient buffer space.

### Parameters

#### context

The context, or NULL to get dropped count from global queue

#### streamId

The stream ID

#### dropped

The number of records that were dropped since the last call to this function.

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `dropped` is NULL

## Description

Get the number of records that were dropped because of insufficient buffer space. The dropped count includes records that could not be recorded because CUPTI did not have activity buffer space available for the record (because the `CUpti_BuffersCallbackRequestFunc` callback did not return an empty buffer of sufficient size) and also CDP records that could not be record because the device-size buffer was



full (size is controlled by the CUPTI\_ACTIVITY\_ATTR\_DEVICE\_BUFFER\_SIZE\_CDP attribute). The dropped count maintained for the queue is reset to zero when this function is called.

## CUptiResult cuptiActivityRegisterCallbacks (CUpti\_BuffersCallbackRequestFunc funcBufferRequested, CUpti\_BuffersCallbackCompleteFunc funcBufferCompleted)

Registers callback functions with CUPTI for activity buffer handling.

### Parameters

#### **funcBufferRequested**

callback which is invoked when an empty buffer is requested by CUPTI

#### **funcBufferCompleted**

callback which is invoked when a buffer containing activity records is available from CUPTI

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if either `funcBufferRequested` or `funcBufferCompleted` is NULL

### Description

This function registers two callback functions to be used in asynchronous buffer handling. If registered, activity record buffers are handled using asynchronous requested/completed callbacks from CUPTI.

Registering these callbacks prevents the client from using CUPTI's blocking enqueue/dequeue functions.

## CUptiResult cuptiActivitySetAttribute (CUpti\_ActivityAttribute attr, size\_t \*valueSize, void \*value)

Write an activity API attribute.

### Parameters

#### attr

The attribute to write

#### valueSize

The size, in bytes, of the value

#### value

The attribute value to write

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `valueSize` or `value` is NULL, or if `attr` is not an activity attribute
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - Indicates that the `value` buffer is too small to hold the attribute value.

### Description

Write an activity API attribute.

## CUptiResult cuptiGetAutoBoostState (CUcontext context, CUpti\_ActivityAutoBoostState \*state)

Get auto boost state.

### Parameters

#### context

A valid CUcontext.

#### state

A pointer to [CUpti\\_ActivityAutoBoostState](#) structure which contains the current state and the id of the process that has requested the current state

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `CUcontext` or `state` is NULL
- ▶ CUPTI\_ERROR\_NOT\_SUPPORTED  
Indicates that the device does not support auto boost
- ▶ CUPTI\_ERROR\_UNKNOWN  
an internal error occurred

**Description**

The profiling results can be inconsistent in case auto boost is enabled. CUPTI tries to disable auto boost while profiling. It can fail to disable in cases where user does not have the permissions or `CUDA_AUTO_BOOST` env variable is set. The function can be used to query whether auto boost is enabled.

## CUptiResult cuptiGetContextId (CUcontext context, uint32\_t \*contextId)

Get the ID of a context.

**Parameters****context**

The context

**contextId**

Returns a process-unique ID for the context

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT  
The context is NULL or not valid.
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `contextId` is NULL

**Description**

Get the ID of a context.

## CUptiResult cuptiGetDeviceId (CUcontext context, uint32\_t \*deviceId)

Get the ID of a device.

### Parameters

#### context

The context, or NULL to indicate the current context.

#### deviceId

Returns the ID of the device that is current for the calling thread.

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
  - if unable to get device ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `deviceId` is NULL

### Description

If `context` is NULL, returns the ID of the device that contains the currently active context. If `context` is non-NULL, returns the ID of the device which contains that context. Operates in a similar manner to `cudaGetDevice()` or `cuCtxGetDevice()` but may be called from within callback functions.

## CUptiResult cuptiGetStreamId (CUcontext context, CUstream stream, uint32\_t \*streamId)

Get the ID of a stream.

### Parameters

#### context

If non-NULL then the stream is checked to ensure that it belongs to this context.

Typically this parameter should be null.

#### stream

The stream

#### streamId

Returns a context-unique ID for the stream

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_STREAM
  - if unable to get stream ID, or if `context` is non-NULL and `stream` does not belong to the context
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `streamId` is NULL

**Description**

Get the ID of a stream. The stream ID is unique within a context (i.e. all streams within a context will have unique stream IDs).

**See also:**

`cuptiActivityEnqueueBuffer`

`cuptiActivityDequeueBuffer`

**CUptiResult cuptiGetTimestamp (uint64\_t \*timestamp)**

Get the CUPTI timestamp.

**Parameters****timestamp**

Returns the CUPTI timestamp

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `timestamp` is NULL

**Description**

Returns a timestamp normalized to correspond with the start and end timestamps reported in the CUPTI activity records. The timestamp is reported in nanoseconds.

**#define CUPTI\_AUTO\_BOOST\_INVALID\_CLIENT\_PID 0**

An invalid/unknown process id.

## **#define CUPTI\_CORRELATION\_ID\_UNKNOWN 0**

An invalid/unknown correlation ID. A correlation ID of this value indicates that there is no correlation for the activity record.

## **#define CUPTI\_GRID\_ID\_UNKNOWN 0LL**

An invalid/unknown grid ID.

## **#define CUPTI\_SOURCE\_LOCATOR\_ID\_UNKNOWN 0**

The source-locator ID that indicates an unknown source location. There is not an actual `CUpti_ActivitySourceLocator` object corresponding to this value.

## **#define CUPTI\_TIMESTAMP\_UNKNOWN 0LL**

An invalid/unknown timestamp for a start, end, queued, submitted, or completed time.

## 2.4. CUPTI Callback API

Functions, types, and enums that implement the CUPTI Callback API.

### **struct CUpti\_CallbackData**

Data passed into a runtime or driver API callback function.

### **struct CUpti\_ModuleResourceData**

Module data passed into a resource callback function.

### **struct CUpti\_NvtxData**

Data passed into a NVTX callback function.

### **struct CUpti\_ResourceData**

Data passed into a resource callback function.

### **struct CUpti\_SynchronizeData**

Data passed into a synchronize callback function.

### **enum CUpti\_ApiCallbackSite**

Specifies the point in an API call that a callback is issued.

Specifies the point in an API call that a callback is issued. This value is communicated to the callback function via `CUpti_CallbackData::callbackSite`.

**Values****CUPTI\_API\_ENTER = 0**

The callback is at the entry of the API call.

**CUPTI\_API\_EXIT = 1**

The callback is at the exit of the API call.

**CUPTI\_API\_CBSITE\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_CallbackDomain

Callback domains.

Callback domains. Each domain represents callback points for a group of related API functions or CUDA driver activity.

**Values****CUPTI\_CB\_DOMAIN\_INVALID = 0**

Invalid domain.

**CUPTI\_CB\_DOMAIN\_DRIVER\_API = 1**

Domain containing callback points for all driver API functions.

**CUPTI\_CB\_DOMAIN\_RUNTIME\_API = 2**

Domain containing callback points for all runtime API functions.

**CUPTI\_CB\_DOMAIN\_RESOURCE = 3**

Domain containing callback points for CUDA resource tracking.

**CUPTI\_CB\_DOMAIN\_SYNCHRONIZE = 4**

Domain containing callback points for CUDA synchronization.

**CUPTI\_CB\_DOMAIN\_NVTX = 5**

Domain containing callback points for NVTX API functions.

**CUPTI\_CB\_DOMAIN\_SIZE = 6****CUPTI\_CB\_DOMAIN\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_CallbackIdResource

Callback IDs for resource domain.

Callback IDs for resource domain, CUPTI\_CB\_DOMAIN\_RESOURCE. This value is communicated to the callback function via the `cbid` parameter.**Values****CUPTI\_CBID\_RESOURCE\_INVALID = 0**

Invalid resource callback ID.

**CUPTI\_CBID\_RESOURCE\_CONTEXT\_CREATED = 1**

A new context has been created.

**CUPTI\_CBID\_RESOURCE\_CONTEXT\_DESTROY\_STARTING = 2**

A context is about to be destroyed.

**CUPTI\_CBID\_RESOURCE\_STREAM\_CREATED = 3**

A new stream has been created.

**CUPTI\_CBID\_RESOURCE\_STREAM\_DESTROY\_STARTING = 4**

A stream is about to be destroyed.

**CUPTI\_CBID\_RESOURCE\_CU\_INIT\_FINISHED = 5**

The driver has finished initializing.

**CUPTI\_CBID\_RESOURCE\_MODULE\_LOADED = 6**

A module has been loaded.

**CUPTI\_CBID\_RESOURCE\_MODULE\_UNLOAD\_STARTING = 7**

A module is about to be unloaded.

**CUPTI\_CBID\_RESOURCE\_MODULE\_PROFILED = 8**

The current module which is being profiled.

**CUPTI\_CBID\_RESOURCE\_SIZE**

**CUPTI\_CBID\_RESOURCE\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_CallbackIdSync

Callback IDs for synchronization domain.

Callback IDs for synchronization domain, CUPTI\_CB\_DOMAIN\_SYNCHRONIZE. This value is communicated to the callback function via the `cbid` parameter.

### Values

**CUPTI\_CBID\_SYNCHRONIZE\_INVALID = 0**

Invalid synchronize callback ID.

**CUPTI\_CBID\_SYNCHRONIZE\_STREAM\_SYNCHRONIZED = 1**

Stream synchronization has completed for the stream.

**CUPTI\_CBID\_SYNCHRONIZE\_CONTEXT\_SYNCHRONIZED = 2**

Context synchronization has completed for the context.

**CUPTI\_CBID\_SYNCHRONIZE\_SIZE**

**CUPTI\_CBID\_SYNCHRONIZE\_FORCE\_INT = 0x7fffffff**

```
typedef (*CUpti_CallbackFunc) (void* userdata,
CUpti_CallbackDomain domain, CUpti_CallbackId cbid,
const void* cbdata)
```

Function type for a callback.

Function type for a callback. The type of the data passed to the callback in `cbdata` depends on the `domain`. If `domain` is CUPTI\_CB\_DOMAIN\_DRIVER\_API or CUPTI\_CB\_DOMAIN\_RUNTIME\_API the type of `cbdata` will be [CUpti\\_CallbackData](#). If `domain` is CUPTI\_CB\_DOMAIN\_RESOURCE the type of `cbdata` will be [CUpti\\_ResourceData](#). If `domain` is CUPTI\_CB\_DOMAIN\_SYNCHRONIZE the type of `cbdata` will be [CUpti\\_SynchronizeData](#). If `domain` is CUPTI\_CB\_DOMAIN\_NVTX the type of `cbdata` will be [CUpti\\_NvtxData](#).



## typedef uint32\_t CUpti\_CallbackId

An ID for a driver API, runtime API, resource or synchronization callback.

An ID for a driver API, runtime API, resource or synchronization callback. Within a driver API callback this should be interpreted as a `CUpti_driver_api_trace_cbid` value (these values are defined in `cupti_driver_cbid.h`). Within a runtime API callback this should be interpreted as a `CUpti_runtime_api_trace_cbid` value (these values are defined in `cupti_runtime_cbid.h`). Within a resource API callback this should be interpreted as a `CUpti_CallbackIdResource` value. Within a synchronize API callback this should be interpreted as a `CUpti_CallbackIdSync` value.

## typedef CUpti\_DomainTable

Pointer to an array of callback domains.

## typedef struct CUpti\_Subscriber\_st \*CUpti\_SubscriberHandle

A callback subscriber.

## CUptiResult cuptiEnableAllDomains (uint32\_t enable, CUpti\_SubscriberHandle subscriber)

Enable or disable all callbacks in all domains.

### Parameters

#### **enable**

New enable state for all callbacks in all domain. Zero disables all callbacks, non-zero enables all callbacks.

#### **subscriber**

- Handle to callback subscription

### Returns

- ▶ `CUPTI_SUCCESS`  
on success
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`  
if unable to initialize CUPTI
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`  
if `subscriber` is invalid

## Description

Enable or disable all callbacks in all domains.



**Thread-safety:** a subscriber must serialize access to `cuptiGetCallbackState`, `cuptiEnableCallback`, `cuptiEnableDomain`, and `cuptiEnableAllDomains`. For example, if `cuptiGetCallbackState(sub, d, *)` and `cuptiEnableAllDomains(sub)` are called concurrently, the results are undefined.

## CUptiResult cuptiEnableCallback (uint32\_t enable, CUpti\_SubscriberHandle subscriber, CUpti\_CallbackDomain domain, CUpti\_CallbackId cbid)

Enable or disabled callbacks for a specific domain and callback ID.

### Parameters

#### enable

New enable state for the callback. Zero disables the callback, non-zero enables the callback.

#### subscriber

- Handle to callback subscription

#### domain

The domain of the callback

#### cbid

The ID of the callback

### Returns

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED  
if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `subscriber`, `domain` or `cbid` is invalid.

## Description

Enable or disabled callbacks for a subscriber for a specific domain and callback ID.



**Thread-safety:** a subscriber must serialize access to `cuptiGetCallbackState`, `cuptiEnableCallback`, `cuptiEnableDomain`, and `cuptiEnableAllDomains`. For example,

if `cuptiGetCallbackState(sub, d, c)` and `cuptiEnableCallback(sub, d, c)` are called concurrently, the results are undefined.

## CUptiResult cuptiEnableDomain (uint32\_t enable, CUpti\_SubscriberHandle subscriber, CUpti\_CallbackDomain domain)

Enable or disabled all callbacks for a specific domain.

### Parameters

#### **enable**

New enable state for all callbacks in the domain. Zero disables all callbacks, non-zero enables all callbacks.

#### **subscriber**

- Handle to callback subscription

#### **domain**

The domain of the callback

### Returns

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED  
if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `subscriber` or `domain` is invalid

### Description

Enable or disabled all callbacks for a specific domain.



**Thread-safety:** a subscriber must serialize access to `cuptiGetCallbackState`, `cuptiEnableCallback`, `cuptiEnableDomain`, and `cuptiEnableAllDomains`. For example, if `cuptiGetCallbackEnabled(sub, d, *)` and `cuptiEnableDomain(sub, d)` are called concurrently, the results are undefined.

## CUptiResult cuptiGetCallbackName (CUpti\_CallbackDomain domain, uint32\_t cbid, const char \*\*name)

Get the name of a callback for a specific domain and callback ID.

### Parameters

#### domain

The domain of the callback

#### cbid

The ID of the callback

#### name

Returns pointer to the name string on success, NULL otherwise

### Returns

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if name is NULL, or if domain or cbid is invalid.

### Description

Returns a pointer to the name c\_string in \*\*name.



Names are available only for the DRIVER and RUNTIME domains.

## CUptiResult cuptiGetCallbackState (uint32\_t \*enable, CUpti\_SubscriberHandle subscriber, CUpti\_CallbackDomain domain, CUpti\_CallbackId cbid)

Get the current enabled/disabled state of a callback for a specific domain and function ID.

### Parameters

#### enable

Returns non-zero if callback enabled, zero if not enabled

#### subscriber

Handle to the initialize subscriber

#### domain

The domain of the callback

**cbid**

The ID of the callback

**Returns**

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED  
if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `enabled` is NULL, or if `subscriber`, `domain` or `cbid` is invalid.

**Description**

Returns non-zero in `*enable` if the callback for a domain and callback ID is enabled, and zero if not enabled.



**Thread-safety:** a subscriber must serialize access to `cuptiGetCallbackState`, `cuptiEnableCallback`, `cuptiEnableDomain`, and `cuptiEnableAllDomains`. For example, if `cuptiGetCallbackState(sub, d, c)` and `cuptiEnableCallback(sub, d, c)` are called concurrently, the results are undefined.

## CuptiResult cuptiSubscribe (Cupti\_SubscriberHandle \*subscriber, Cupti\_CallbackFunc callback, void \*userdata)

Initialize a callback subscriber with a callback function and user data.

**Parameters****subscriber**

Returns handle to initialize subscriber

**callback**

The callback function

**userdata**

A pointer to user data. This data will be passed to the callback function via the `userdata` parameter.

**Returns**

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED

- ▶ if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_MAX\_LIMIT\_REACHED
- ▶ if there is already a CUPTI subscriber
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- ▶ if subscriber is NULL

### Description

Initializes a callback subscriber with a callback function and (optionally) a pointer to user data. The returned subscriber handle can be used to enable and disable the callback for specific domains and callback IDs.

- ▶ Only a single subscriber can be registered at a time.
- ▶ This function does not enable any callbacks.
- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiSupportedDomains (size\_t \*domainCount, CUpti\_DomainTable \*domainTable)

Get the available callback domains.

### Parameters

#### domainCount

Returns number of callback domains

#### domainTable

Returns pointer to array of available callback domains

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- ▶ if domainCount or domainTable are NULL

**Description**

Returns in `*domainTable` an array of size `*domainCount` of all the available callback domains.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiUnsubscribe (CUpti\_SubscriberHandle subscriber)

Unregister a callback subscriber.

**Parameters****subscriber**

Handle to the initialize subscriber

**Returns**

- ▶ CUPTI\_SUCCESS  
on success
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED  
if unable to initialize CUPTI
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `subscriber` is NULL or not initialized

**Description**

Removes a callback subscriber so that no future callbacks will be issued to that subscriber.



**Thread-safety:** this function is thread safe.

## 2.5. CUPTI Event API

Functions, types, and enums that implement the CUPTI Event API.

## struct CUpti\_EventGroupSet

A set of event groups.

## struct CUpti\_EventGroupSets

A set of event group sets.

## enum CUpti\_DeviceAttribute

Device attributes.

CUPTI device attributes. These attributes can be read using `cuptiDeviceGetAttribute`.

### Values

**CUPTI\_DEVICE\_ATTR\_MAX\_EVENT\_ID = 1**

Number of event IDs for a device. Value is a `uint32_t`.

**CUPTI\_DEVICE\_ATTR\_MAX\_EVENT\_DOMAIN\_ID = 2**

Number of event domain IDs for a device. Value is a `uint32_t`.

**CUPTI\_DEVICE\_ATTR\_GLOBAL\_MEMORY\_BANDWIDTH = 3**

Get global memory bandwidth in Kbytes/sec. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_INSTRUCTION\_PER\_CYCLE = 4**

Get theoretical maximum number of instructions per cycle. Value is a `uint32_t`.

**CUPTI\_DEVICE\_ATTR\_INSTRUCTION\_THROUGHPUT\_SINGLE\_PRECISION = 5**

Get theoretical maximum number of single precision instructions that can be executed per second. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_MAX\_FRAME\_BUFFERS = 6**

Get number of frame buffers for device. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_PCIE\_LINK\_RATE = 7**

Get PCIE link rate in Mega bits/sec for device. Return 0 if bus-type is non-PCIE. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_PCIE\_LINK\_WIDTH = 8**

Get PCIE link width for device. Return 0 if bus-type is non-PCIE. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_PCIE\_GEN = 9**

Get PCIE generation for device. Return 0 if bus-type is non-PCIE. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_DEVICE\_CLASS = 10**

Get the class for the device. Value is a `CUpti_DeviceAttributeDeviceClass`.

**CUPTI\_DEVICE\_ATTR\_FLOP\_SP\_PER\_CYCLE = 11**

Get the peak single precision flop per cycle. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_FLOP\_DP\_PER\_CYCLE = 12**

Get the peak double precision flop per cycle. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_MAX\_L2\_UNITS = 13**

Get number of L2 units. Value is a `uint64_t`.

**CUPTI\_DEVICE\_ATTR\_FORCE\_INT = 0x7fffffff**



## enum CUpti\_DeviceAttributeDeviceClass

Device class.

Enumeration of device classes for device attribute  
CUPTI\_DEVICE\_ATTR\_DEVICE\_CLASS.

### Values

CUPTI\_DEVICE\_ATTR\_DEVICE\_CLASS\_TESLA = 0  
 CUPTI\_DEVICE\_ATTR\_DEVICE\_CLASS\_QUADRO = 1  
 CUPTI\_DEVICE\_ATTR\_DEVICE\_CLASS\_GEFORCE = 2

## enum CUpti\_EventAttribute

Event attributes.

Event attributes. These attributes can be read using `cuptiEventGetAttribute`.

### Values

CUPTI\_EVENT\_ATTR\_NAME = 0  
 Event name. Value is a null terminated const c-string.  
 CUPTI\_EVENT\_ATTR\_SHORT\_DESCRIPTION = 1  
 Short description of event. Value is a null terminated const c-string.  
 CUPTI\_EVENT\_ATTR\_LONG\_DESCRIPTION = 2  
 Long description of event. Value is a null terminated const c-string.  
 CUPTI\_EVENT\_ATTR\_CATEGORY = 3  
 Category of event. Value is CUpti\_EventCategory.  
 CUPTI\_EVENT\_ATTR\_FORCE\_INT = 0x7fffffff

## enum CUpti\_EventCategory

An event category.

Each event is assigned to a category that represents the general type of the event. A event's category is accessed using `cuptiEventGetAttribute` and the CUPTI\_EVENT\_ATTR\_CATEGORY attribute.

### Values

CUPTI\_EVENT\_CATEGORY\_INSTRUCTION = 0  
 An instruction related event.  
 CUPTI\_EVENT\_CATEGORY\_MEMORY = 1  
 A memory related event.  
 CUPTI\_EVENT\_CATEGORY\_CACHE = 2  
 A cache related event.  
 CUPTI\_EVENT\_CATEGORY\_PROFILE\_TRIGGER = 3

A profile-trigger event.

**CUPTI\_EVENT\_CATEGORY\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_EventCollectionMethod

The collection method used for an event.

The collection method indicates how an event is collected.

### Values

**CUPTI\_EVENT\_COLLECTION\_METHOD\_PM = 0**

Event is collected using a hardware global performance monitor.

**CUPTI\_EVENT\_COLLECTION\_METHOD\_SM = 1**

Event is collected using a hardware SM performance monitor.

**CUPTI\_EVENT\_COLLECTION\_METHOD\_INSTRUMENTED = 2**

Event is collected using software instrumentation.

**CUPTI\_EVENT\_COLLECTION\_METHOD\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_EventCollectionMode

Event collection modes.

The event collection mode determines the period over which the events within the enabled event groups will be collected.

### Values

**CUPTI\_EVENT\_COLLECTION\_MODE\_CONTINUOUS = 0**

Events are collected for the entire duration between the `cuptiEventGroupEnable` and `cuptiEventGroupDisable` calls. For devices with compute capability less than 2.0, event values are reset when a kernel is launched. For all other devices event values are only reset when the events are read. For CUDA toolkit v6.0 and older this was the default mode. From CUDA toolkit v6.5 this mode is supported on Tesla devices only.

**CUPTI\_EVENT\_COLLECTION\_MODE\_KERNEL = 1**

Events are collected only for the durations of kernel executions that occur between the `cuptiEventGroupEnable` and `cuptiEventGroupDisable` calls. Event collection begins when a kernel execution begins, and stops when kernel execution completes. Event values are reset to zero when each kernel execution begins. If multiple kernel executions occur between the `cuptiEventGroupEnable` and `cuptiEventGroupDisable` calls then the event values must be read after each kernel launch if those events need to be associated with the specific kernel launch. This is the default mode from CUDA toolkit v6.5, and it is the only supported mode for non-Tesla (Quadro, GeForce etc.) devices.

**CUPTI\_EVENT\_COLLECTION\_MODE\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_EventDomainAttribute

Event domain attributes.

Event domain attributes. Except where noted, all the attributes can be read using either [cuptiDeviceGetEventDomainAttribute](#) or [cuptiEventDomainGetAttribute](#).

### Values

**CUPTI\_EVENT\_DOMAIN\_ATTR\_NAME = 0**

Event domain name. Value is a null terminated const c-string.

**CUPTI\_EVENT\_DOMAIN\_ATTR\_INSTANCE\_COUNT = 1**

Number of instances of the domain for which event counts will be collected.

The domain may have additional instances that cannot be profiled (see [CUPTI\\_EVENT\\_DOMAIN\\_ATTR\\_TOTAL\\_INSTANCE\\_COUNT](#)). Can be read only with [cuptiDeviceGetEventDomainAttribute](#). Value is a `uint32_t`.

**CUPTI\_EVENT\_DOMAIN\_ATTR\_TOTAL\_INSTANCE\_COUNT = 3**

Total number of instances of the domain, including instances that cannot be profiled. Use [CUPTI\\_EVENT\\_DOMAIN\\_ATTR\\_INSTANCE\\_COUNT](#) to get the number of instances that can be profiled. Can be read only with [cuptiDeviceGetEventDomainAttribute](#). Value is a `uint32_t`.

**CUPTI\_EVENT\_DOMAIN\_ATTR\_COLLECTION\_METHOD = 4**

Collection method used for events contained in the event domain. Value is a [CUpti\\_EventCollectionMethod](#).

**CUPTI\_EVENT\_DOMAIN\_ATTR\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_EventGroupAttribute

Event group attributes.

Event group attributes. These attributes can be read using [cuptiEventGroupGetAttribute](#). Attributes marked [rw] can also be written using [cuptiEventGroupSetAttribute](#).

### Values

**CUPTI\_EVENT\_GROUP\_ATTR\_EVENT\_DOMAIN\_ID = 0**

The domain to which the event group is bound. This attribute is set when the first event is added to the group. Value is a [CUpti\\_EventDomainID](#).

**CUPTI\_EVENT\_GROUP\_ATTR\_PROFILE\_ALL\_DOMAIN\_INSTANCES = 1**

[rw] Profile all the instances of the domain for this eventgroup. This feature can be used to get load balancing across all instances of a domain. Value is an integer.

**CUPTI\_EVENT\_GROUP\_ATTR\_USER\_DATA = 2**

[rw] Reserved for user data.

**CUPTI\_EVENT\_GROUP\_ATTR\_NUM\_EVENTS = 3**

Number of events in the group. Value is a `uint32_t`.

**CUPTI\_EVENT\_GROUP\_ATTR\_EVENTS = 4**

Enumerates events in the group. Value is a pointer to buffer of size `sizeof(CUpti_EventID) * num_of_events` in the eventgroup. `num_of_events` can be queried using `CUPTI_EVENT_GROUP_ATTR_NUM_EVENTS`.

**CUPTI\_EVENT\_GROUP\_ATTR\_INSTANCE\_COUNT = 5**

Number of instances of the domain bound to this event group that will be counted. Value is a `uint32_t`.

**CUPTI\_EVENT\_GROUP\_ATTR\_FORCE\_INT = 0x7fffffff**

## enum CUpti\_ReadEventFlags

Flags for `cuptiEventGroupReadEvent` and `cuptiEventGroupReadAllEvents`.

Flags for `cuptiEventGroupReadEvent` and `cuptiEventGroupReadAllEvents`.

### Values

**CUPTI\_EVENT\_READ\_FLAG\_NONE = 0**

No flags.

**CUPTI\_EVENT\_READ\_FLAG\_FORCE\_INT = 0x7fffffff**

## typedef uint32\_t CUpti\_EventDomainID

ID for an event domain.

ID for an event domain. An event domain represents a group of related events. A device may have multiple instances of a domain, indicating that the device can simultaneously record multiple instances of each event within that domain.

## typedef void \*CUpti\_EventGroup

A group of events.

An event group is a collection of events that are managed together. All events in an event group must belong to the same domain.

## typedef uint32\_t CUpti\_EventID

ID for an event.

An event represents a countable activity, action, or occurrence on the device.

## CUptiResult cuptiDeviceEnumEventDomains (CUdevice device, size\_t \*arraySizeBytes, CUpti\_EventDomainID \*domainArray)

Get the event domains for a device.

### Parameters

#### device

The CUDA device

#### arraySizeBytes

The size of `domainArray` in bytes, and returns the number of bytes written to `domainArray`

#### domainArray

Returns the IDs of the event domains for the device

### Returns

- ▶ CUPTI\_SUCCESS
  - ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
  - ▶ CUPTI\_ERROR\_INVALID\_DEVICE
  - ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- if `arraySizeBytes` or `domainArray` are NULL

### Description

Returns the event domains IDs in `domainArray` for a device. The size of the `domainArray` buffer is given by `*arraySizeBytes`. The size of the `domainArray` buffer must be at least `numdomains * sizeof(CUpti_EventDomainID)` or else all domains will not be returned. The value returned in `*arraySizeBytes` contains the number of bytes returned in `domainArray`.



**Thread-safety:** this function is thread safe.

## CuptiResult cuptiDeviceGetAttribute (CUdevice device, CUpti\_DeviceAttribute attrib, size\_t \*valueSize, void \*value)

Read a device attribute.

### Parameters

#### device

The CUDA device

#### attrib

The attribute to read

#### valueSize

Size of buffer pointed by the value, and returns the number of bytes written to value

#### value

Returns the value of the attribute

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if valueSize or value is NULL, or if attrib is not a device attribute

- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT

For non-c-string attribute values, indicates that the value buffer is too small to hold the attribute value.

### Description

Read a device attribute and return it in \*value.



**Thread-safety:** this function is thread safe.

## CuptiResult cuptiDeviceGetEventDomainAttribute (CUdevice device, CUpti\_EventDomainID eventDomain,

## CUpti\_EventDomainAttribute attrib, size\_t \*valueSize, void \*value)

Read an event domain attribute.

### Parameters

#### device

The CUDA device

#### eventDomain

ID of the event domain

#### attrib

The event domain attribute to read

#### valueSize

The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

#### value

Returns the attribute's value

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_DOMAIN\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `valueSize` or `value` is NULL, or if `attrib` is not an event domain attribute
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

### Description

Returns an event domain attribute in `*value`. The size of the `value` buffer is given by `*valueSize`. The value returned in `*valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than `*valueSize`, then only the first `*valueSize` characters will be returned and there will be no terminating null byte.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiDeviceGetNumEventDomains (CUdevice device, uint32\_t \*numDomains)

Get the number of domains for a device.

### Parameters

#### device

The CUDA device

#### numDomains

Returns the number of domains

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if numDomains is NULL

### Description

Returns the number of domains in numDomains for a device.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiDeviceGetTimestamp (CUcontext context, uint64\_t \*timestamp)

Read a device timestamp.

### Parameters

#### context

A context on the device from which to get the timestamp

#### timestamp

Returns the device timestamp

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED



- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

is `timestamp` is NULL

### Description

Returns the device timestamp in `*timestamp`. The timestamp is reported in nanoseconds and indicates the time since the device was last reset.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiDisableKernelReplayMode (CUcontext context)

Disable kernel replay mode.

### Parameters

#### context

The context

### Returns

- ▶ CUPTI\_SUCCESS

### Description

Set profiling mode for the context to non-replay (default) mode. Event collection mode will be set to CUPTI\_EVENT\_COLLECTION\_MODE\_KERNEL. All previously enabled event groups and event group sets will be disabled.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEnableKernelReplayMode (CUcontext context)

Enable kernel replay mode.

### Parameters

#### context

The context

**Returns**

- ▶ CUPTI\_SUCCESS

**Description**

Set profiling mode for the context to replay mode. In this mode, any number of events can be collected in one run of the kernel. The event collection mode will automatically switch to CUPTI\_EVENT\_COLLECTION\_MODE\_KERNEL. In this mode, `cuptiSetEventCollectionMode` will return CUPTI\_ERROR\_INVALID\_OPERATION.



- ▶ **Kernels** might take longer to run if many events are enabled.
- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEnumEventDomains (size\_t \*arraySizeBytes, CUpti\_EventDomainID \*domainArray)

Get the event domains available on any device.

**Parameters****arraySizeBytes**

The size of `domainArray` in bytes, and returns the number of bytes written to `domainArray`

**domainArray**

Returns all the event domains

**Returns**

- ▶ CUPTI\_SUCCESS
  - ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- if `arraySizeBytes` or `domainArray` are NULL

**Description**

Returns all the event domains available on any CUDA-capable device. Event domain IDs are returned in `domainArray`. The size of the `domainArray` buffer is given by `*arraySizeBytes`. The size of the `domainArray` buffer must be at least `numDomains * sizeof(CUpti_EventDomainID)` or all domains will not be returned. The value returned in `*arraySizeBytes` contains the number of bytes returned in `domainArray`.



- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventDomainEnumEvents (CUpti\_EventDomainID eventDomain, size\_t \*arraySizeBytes, CUpti\_EventID \*eventArray)

Get the events in a domain.

### Parameters

#### eventDomain

ID of the event domain

#### arraySizeBytes

The size of `eventArray` in bytes, and returns the number of bytes written to `eventArray`

#### eventArray

Returns the IDs of the events in the domain

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_DOMAIN\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `arraySizeBytes` or `eventArray` are NULL

### Description

Returns the event IDs in `eventArray` for a domain. The size of the `eventArray` buffer is given by `*arraySizeBytes`. The size of the `eventArray` buffer must be at least `numdomainevents * sizeof(CUpti_EventID)` or else all events will not be returned. The value returned in `*arraySizeBytes` contains the number of bytes returned in `eventArray`.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventDomainGetAttribute (CUpti\_EventDomainID eventDomain,

## CUpti\_EventDomainAttribute attrib, size\_t \*valueSize, void \*value)

Read an event domain attribute.

### Parameters

#### eventDomain

ID of the event domain

#### attrib

The event domain attribute to read

#### valueSize

The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

#### value

Returns the attribute's value

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_DOMAIN\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `valueSize` or `value` is NULL, or if `attrib` is not an event domain attribute
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

### Description

Returns an event domain attribute in `*value`. The size of the `value` buffer is given by `*valueSize`. The value returned in `*valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than `*valueSize`, then only the first `*valueSize` characters will be returned and there will be no terminating null byte.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventDomainGetNumEvents (CUpti\_EventDomainID eventDomain, uint32\_t \*numEvents)

Get number of events in a domain.

### Parameters

#### eventDomain

ID of the event domain

#### numEvents

Returns the number of events in the domain

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_DOMAIN\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `numEvents` is NULL

### Description

Returns the number of events in `numEvents` for a domain.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGetAttribute (CUpti\_EventID event, CUpti\_EventAttribute attrib, size\_t \*valueSize, void \*value)

Get an event attribute.

### Parameters

#### event

ID of the event

#### attrib

The event attribute to read

**valueSize**

The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

**value**

Returns the attribute's value

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `valueSize` or `value` is NULL, or if `attrib` is not an event attribute

- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT

For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

**Description**

Returns an event attribute in `*value`. The size of the `value` buffer is given by `*valueSize`. The value returned in `*valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than `*valueSize`, then only the first `*valueSize` characters will be returned and there will be no terminating null byte.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGetIdFromName (CUdevice device, const char \*eventName, CUpti\_EventID \*event)

Find an event by name.

**Parameters****device**

The CUDA device

**eventName**

The name of the event to find

**event**

Returns the ID of the found event or undefined if unable to find the event

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_NAME
  - if unable to find an event with name `eventName`. In this case `*event` is undefined
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `eventName` or `event` are NULL

**Description**

Find an event by name and return the event ID in `*event`.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupAddEvent (CUpti\_EventGroup eventGroup, CUpti\_EventID event)

Add an event to an event group.

**Parameters****eventGroup**

The event group

**event**

The event to add to the group

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_ID
- ▶ CUPTI\_ERROR\_OUT\_OF\_MEMORY
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if `eventGroup` is enabled
- ▶ CUPTI\_ERROR\_NOT\_COMPATIBLE

if `event` belongs to a different event domain than the events already in `eventGroup`, or if a device limitation prevents `event` from being collected at the same time as the events already in `eventGroup`

- ▶ CUPTI\_ERROR\_MAX\_LIMIT\_REACHED

if `eventGroup` is full

- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `eventGroup` is NULL

### Description

Add an event to an event group. The event add can fail for a number of reasons:

- ▶ The event group is enabled
- ▶ The event does not belong to the same event domain as the events that are already in the event group
- ▶ Device limitations on the events that can belong to the same group
- ▶ The event group is full



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupCreate (CUcontext context, CUpti\_EventGroup \*eventGroup, uint32\_t flags)

Create a new event group for a context.

### Parameters

#### **context**

The context for the event group

#### **eventGroup**

Returns the new event group

#### **flags**

Reserved - must be zero

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT
- ▶ CUPTI\_ERROR\_OUT\_OF\_MEMORY
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER



if `eventGroup` is `NULL`

### Description

Creates a new event group for `context` and returns the new group in `*eventGroup`.



- ▶ `flags` are reserved for future use and should be set to zero.
- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupDestroy (CUpti\_EventGroup eventGroup)

Destroy an event group.

### Parameters

#### **eventGroup**

The event group to destroy

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_OPERATION`  
if the event group is enabled
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`  
if `eventGroup` is `NULL`

### Description

Destroy an `eventGroup` and free its resources. An event group cannot be destroyed if it is enabled.



- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupDisable (CUpti\_EventGroup eventGroup)

Disable an event group.

### Parameters

#### **eventGroup**

The event group

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `eventGroup` is NULL

### Description

Disable an event group. Disabling an event group stops collection of events contained in the group.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupEnable (CUpti\_EventGroup eventGroup)

Enable an event group.

### Parameters

#### **eventGroup**

The event group


### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_NOT\_READY

- ▶ if `eventGroup` does not contain any events
- ▶ `CUPTI_ERROR_NOT_COMPATIBLE`
- ▶ if `eventGroup` cannot be enabled due to other already enabled event groups
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`
- ▶ if `eventGroup` is `NULL`
- ▶ `CUPTI_ERROR_HARDWARE_BUSY`
- ▶ if another client is profiling and hardware is busy

### Description

Enable an event group. Enabling an event group zeros the value of all the events in the group and then starts collection of those events.

 **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupGetAttribute (CUpti\_EventGroup eventGroup, CUpti\_EventGroupAttribute attrib, size\_t \*valueSize, void \*value)

Read an event group attribute.

### Parameters

#### **eventGroup**

The event group

#### **attrib**

The attribute to read

#### **valueSize**

Size of buffer pointed by the value, and returns the number of bytes written to `value`

#### **value**

Returns the value of the attribute

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`
- ▶ if `valueSize` or `value` is `NULL`, or if `attrib` is not an eventgroup attribute
- ▶ `CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT`

For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

## Description

Read an event group attribute and return it in `*value`.



**Thread-safety:** this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to `cuptiEventGroupDestroy`, `cuptiEventGroupAddEvent`, etc.), and must guard against simultaneous destruction of the context in which `eventGroup` was created (for example, client must guard against simultaneous calls to `cudaDeviceReset`, `cuCtxDestroy`, etc.).

**CUptiResult** `cuptiEventGroupReadAllEvents`  
 (CUpti\_EventGroup `eventGroup`, CUpti\_ReadEventFlags `flags`, `size_t` `*eventValueBufferSizeBytes`, `uint64_t` `*eventValueBuffer`, `size_t` `*eventIdArraySizeBytes`, CUpti\_EventID `*eventIdArray`, `size_t` `*numEventIdsRead`)

Read the values for all the events in an event group.

## Parameters

### `eventGroup`

The event group

### `flags`

Flags controlling the reading mode

### `eventValueBufferSizeBytes`

The size of `eventValueBuffer` in bytes, and returns the number of bytes written to `eventValueBuffer`

### `eventValueBuffer`

Returns the event values

### `eventIdArraySizeBytes`

The size of `eventIdArray` in bytes, and returns the number of bytes written to `eventIdArray`

### `eventIdArray`

Returns the IDs of the events in the same order as the values return in `eventValueBuffer`.

### `numEventIdsRead`

Returns the number of event IDs returned in `eventIdArray`

## Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if eventGroup is disabled
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if eventGroup, eventValueBufferSizeBytes, eventValueBuffer, eventIdArraySizeBytes, eventIdArray or numEventIdsRead is NULL

## Description

Read the values for all the events in an event group. The event values are returned in the eventValueBuffer buffer. eventValueBufferSizeBytes indicates the size of eventValueBuffer. The buffer must be at least (sizeof(uint64) \* number of events in group) if CUPTI\_EVENT\_GROUP\_ATTR\_PROFILE\_ALL\_DOMAIN\_INSTANCES is not set on the group containing the events. The buffer must be at least (sizeof(uint64) \* number of domain instances \* number of events in group) if CUPTI\_EVENT\_GROUP\_ATTR\_PROFILE\_ALL\_DOMAIN\_INSTANCES is set on the group.

The data format returned in eventValueBuffer is:

- ▶ domain instance 0: event0 event1 ... eventN
- ▶ domain instance 1: event0 event1 ... eventN
- ▶ ...
- ▶ domain instance M: event0 event1 ... eventN

The event order in eventValueBuffer is returned in eventIdArray. The size of eventIdArray is specified in eventIdArraySizeBytes. The size should be at least (sizeof(CUpti\_EventID) \* number of events in group).

If any instance of any event counter overflows, the value returned for that event instance will be CUPTI\_EVENT\_OVERFLOW.

The only allowed value for flags is CUPTI\_EVENT\_READ\_FLAG\_NONE.

Reading events from a disabled event group is not allowed. After being read, an event's value is reset to zero.



**Thread-safety:** this function is thread safe but client must guard against simultaneous destruction or modification of eventGroup (for example, client must guard against simultaneous calls to cuptiEventGroupDestroy, cuptiEventGroupAddEvent, etc.), and must guard against simultaneous destruction of the context in which

`eventGroup` was created (for example, client must guard against simultaneous calls to `cudaDeviceReset`, `cuCtxDestroy`, etc.). If `cuptiEventGroupResetAllEvents` is called simultaneously with this function, then returned event values are undefined.

## CUptiResult cuptiEventGroupReadEvent (CUpti\_EventGroup eventGroup, CUpti\_ReadEventFlags flags, CUpti\_EventID event, size\_t \*eventValueBufferSizeBytes, uint64\_t \*eventValueBuffer)

Read the value for an event in an event group.

### Parameters

#### **eventGroup**

The event group

#### **flags**

Flags controlling the reading mode

#### **event**

The event to read

#### **eventValueBufferSizeBytes**

The size of `eventValueBuffer` in bytes, and returns the number of bytes written to `eventValueBuffer`

#### **eventValueBuffer**

Returns the event value(s)

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_ID
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if `eventGroup` is disabled
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `eventGroup`, `eventValueBufferSizeBytes` or `eventValueBuffer` is NULL

## Description

Read the value for an event in an event group. The event value is returned in the `eventValueBuffer` buffer. `eventValueBufferSizeBytes` indicates the size of the `eventValueBuffer` buffer. The buffer must be at least `sizeof(uint64)` if `CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES` is not set on the group containing the event. The buffer must be at least `(sizeof(uint64) * number of domain instances)` if `CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES` is set on the group.

If any instance of an event counter overflows, the value returned for that event instance will be `CUPTI_EVENT_OVERFLOW`.

The only allowed value for `flags` is `CUPTI_EVENT_READ_FLAG_NONE`.

Reading an event from a disabled event group is not allowed. After being read, an event's value is reset to zero.



**Thread-safety:** this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to `cuptiEventGroupDestroy`, `cuptiEventGroupAddEvent`, etc.), and must guard against simultaneous destruction of the context in which `eventGroup` was created (for example, client must guard against simultaneous calls to `cudaDeviceReset`, `cuCtxDestroy`, etc.). If `cuptiEventGroupResetAllEvents` is called simultaneously with this function, then returned event values are undefined.

## CuptiResult cuptiEventGroupRemoveAllEvents (CUpti\_EventGroup eventGroup)

Remove all events from an event group.

### Parameters

#### `eventGroup`

The event group

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_OPERATION`
  - if `eventGroup` is enabled
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`

if `eventGroup` is NULL

### Description

Remove all events from an event group. Events cannot be removed if the event group is enabled.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupRemoveEvent (CUpti\_EventGroup eventGroup, CUpti\_EventID event)

Remove an event from an event group.

### Parameters

#### **eventGroup**

The event group

#### **event**

The event to remove from the group

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_ID
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if `eventGroup` is enabled
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `eventGroup` is NULL

### Description

Remove `event` from the an event group. The event cannot be removed if the event group is enabled.



**Thread-safety:** this function is thread safe.



## CUptiResult cuptiEventGroupResetAllEvents (CUpti\_EventGroup eventGroup)

Zero all the event counts in an event group.

### Parameters

#### eventGroup

The event group

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if eventGroup is NULL

### Description

Zero all the event counts in an event group.



**Thread-safety:** this function is thread safe but client must guard against simultaneous destruction or modification of eventGroup (for example, client must guard against simultaneous calls to `cuptiEventGroupDestroy`, `cuptiEventGroupAddEvent`, etc.), and must guard against simultaneous destruction of the context in which eventGroup was created (for example, client must guard against simultaneous calls to `cudaDeviceReset`, `cuCtxDestroy`, etc.).

## CUptiResult cuptiEventGroupSetAttribute (CUpti\_EventGroup eventGroup, CUpti\_EventGroupAttribute attrib, size\_t valueSize, void \*value)

Write an event group attribute.

### Parameters

#### eventGroup

The event group

#### attrib

The attribute to write

**valueSize**

The size, in bytes, of the value

**value**

The attribute value to write

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `valueSize` or `value` is NULL, or if `attrib` is not an event group attribute, or if `attrib` is not a writable attribute

- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT

Indicates that the `value` buffer is too small to hold the attribute value.

**Description**

Write an event group attribute.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupSetDisable (CUpti\_EventGroupSet \*eventGroupSet)

Disable an event group set.

**Parameters****eventGroupSet**

The pointer to the event group set

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `eventGroupSet` is NULL

## Description

Disable a set of event groups. Disabling a set of event groups stops collection of events contained in the groups.



- ▶ **Thread-safety:** this function is thread safe.
- ▶ If this call fails, some of the event groups in the set may be disabled and other event groups may remain enabled.

## CUptiResult cuptiEventGroupSetEnable (CUpti\_EventGroupSet \*eventGroupSet)

Enable an event group set.

### Parameters

#### eventGroupSet

The pointer to the event group set

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_HARDWARE
- ▶ CUPTI\_ERROR\_NOT\_READY
  - if eventGroup does not contain any events
- ▶ CUPTI\_ERROR\_NOT\_COMPATIBLE
  - if eventGroup cannot be enabled due to other already enabled event groups
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if eventGroupSet is NULL
- ▶ CUPTI\_ERROR\_HARDWARE\_BUSY
  - if other client is profiling and hardware is busy

## Description

Enable a set of event groups. Enabling a set of event groups zeros the value of all the events in all the groups and then starts collection of those events.



- ▶ **Thread-safety:** this function is thread safe.

## CUptiResult cuptiEventGroupSetsCreate (CUcontext context, size\_t eventIdArraySizeBytes, CUpti\_EventID \*eventIdArray, CUpti\_EventGroupSets \*\*eventGroupPasses)

For a set of events, get the grouping that indicates the number of passes and the event groups necessary to collect the events.

### Parameters

#### context

The context for event collection

#### eventIdArraySizeBytes

Size of `eventIdArray` in bytes

#### eventIdArray

Array of event IDs that need to be grouped

#### eventGroupPasses

Returns a `CUpti_EventGroupSets` object that indicates the number of passes required to collect the events and the events to collect on each pass

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `eventIdArray` or `eventGroupPasses` is NULL

### Description

The number of events that can be collected simultaneously varies by device and by the type of the events. When events can be collected simultaneously, they may need to be grouped into multiple event groups because they are from different event domains. This function takes a set of events and determines how many passes are required to collect all those events, and which events can be collected simultaneously in each pass.

The `CUpti_EventGroupSets` returned in `eventGroupPasses` indicates how many passes are required to collect the events with the `numSets` field. Within each event

group set, the `sets` array indicates the event groups that should be collected on each pass.



**Thread-safety:** this function is thread safe, but client must guard against another thread simultaneously destroying `context`.

## CUptiResult cuptiEventGroupSetsDestroy (CUpti\_EventGroupSets \*eventGroupSets)

Destroy a CUpti\_EventGroupSets object.

### Parameters

#### eventGroupSets

The object to destroy

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if any of the event groups contained in the sets is enabled
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if eventGroupSets is NULL

### Description

Destroy a CUpti\_EventGroupSets object.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiGetNumEventDomains (uint32\_t \*numDomains)

Get the number of event domains available on any device.

### Parameters

#### numDomains

Returns the number of domains

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `numDomains` is NULL

**Description**

Returns the total number of event domains available on any CUDA-capable device.



**Thread-safety:** this function is thread safe.

## CUptiResult cuptiSetEventCollectionMode (CUcontext context, CUpti\_EventCollectionMode mode)

Set the event collection mode.

**Parameters****context**

The context

**mode**

The event collection mode

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
  - if called when replay mode is enabled
- ▶ CUPTI\_ERROR\_NOT\_SUPPORTED
  - if mode is not supported on the device

**Description**

Set the event collection mode for a `context`. The `mode` controls the event collection behavior of all events in event groups created in the `context`. This API is invalid in kernel replay mode.



**Thread-safety:** this function is thread safe.

```
#define CUPTI_EVENT_OVERFLOW
((uint64_t)0xFFFFFFFFFFFFFFFFULL)
```

The overflow value for a CUPTI event.

The CUPTI event value that indicates an overflow.

## 2.6. CUPTI Metric API

Functions, types, and enums that implement the CUPTI Metric API.

### union CUpti\_MetricValue

A metric value.

### enum CUpti\_MetricAttribute

Metric attributes.

Metric attributes describe properties of a metric. These attributes can be read using [cuprtiMetricGetAttribute](#).

#### Values

**CUPTI\_METRIC\_ATTR\_NAME = 0**

Metric name. Value is a null terminated const c-string.

**CUPTI\_METRIC\_ATTR\_SHORT\_DESCRIPTION = 1**

Short description of metric. Value is a null terminated const c-string.

**CUPTI\_METRIC\_ATTR\_LONG\_DESCRIPTION = 2**

Long description of metric. Value is a null terminated const c-string.

**CUPTI\_METRIC\_ATTR\_CATEGORY = 3**

Category of the metric. Value is of type CUpti\_MetricCategory.

**CUPTI\_METRIC\_ATTR\_VALUE\_KIND = 4**

Value type of the metric. Value is of type CUpti\_MetricValueKind.

**CUPTI\_METRIC\_ATTR\_EVALUATION\_MODE = 5**

Metric evaluation mode. Value is of type CUpti\_MetricEvaluationMode.

**CUPTI\_METRIC\_ATTR\_FORCE\_INT = 0x7fffffff**

### enum CUpti\_MetricCategory

A metric category.

Each metric is assigned to a category that represents the general type of the metric. A metric's category is accessed using [cuprtiMetricGetAttribute](#) and the CUPTI\_METRIC\_ATTR\_CATEGORY attribute.

**Values****CUPTI\_METRIC\_CATEGORY\_MEMORY = 0**

A memory related metric.

**CUPTI\_METRIC\_CATEGORY\_INSTRUCTION = 1**

An instruction related metric.

**CUPTI\_METRIC\_CATEGORY\_MULTIPROCESSOR = 2**

A multiprocessor related metric.

**CUPTI\_METRIC\_CATEGORY\_CACHE = 3**

A cache related metric.

**CUPTI\_METRIC\_CATEGORY\_TEXTURE = 4**

A texture related metric.

**CUPTI\_METRIC\_CATEGORY\_FORCE\_INT = 0x7fffffff****enum CUpti\_MetricEvaluationMode**

A metric evaluation mode.

A metric can be evaluated per hardware instance to know the load balancing across instances of a domain or the metric can be evaluated in aggregate mode when the events involved in metric evaluation are from different event domains. It might be possible to evaluate some metrics in both modes for convenience. A metric's evaluation mode is accessed using [CUpti\\_MetricEvaluationMode](#) and the `CUPTI_METRIC_ATTR_EVALUATION_MODE` attribute.

**Values****CUPTI\_METRIC\_EVALUATION\_MODE\_PER\_INSTANCE = 1**

If this bit is set, the metric can be profiled for each instance of the domain. The event values passed to [cuptiMetricGetValue](#) can contain values for one instance of the domain. And [cuptiMetricGetValue](#) can be called for each instance.

**CUPTI\_METRIC\_EVALUATION\_MODE\_AGGREGATE = 1<<1**

If this bit is set, the metric can be profiled over all instances. The event values passed to [cuptiMetricGetValue](#) can be aggregated values of events for all instances of the domain.

**CUPTI\_METRIC\_EVALUATION\_MODE\_FORCE\_INT = 0x7fffffff****enum CUpti\_MetricPropertyDeviceClass**

Device class.

Enumeration of device classes for metric property

`CUPTI_METRIC_PROPERTY_DEVICE_CLASS.`**Values****CUPTI\_METRIC\_PROPERTY\_DEVICE\_CLASS\_TESLA = 0****CUPTI\_METRIC\_PROPERTY\_DEVICE\_CLASS\_QUADRO = 1**



**CUPTI\_METRIC\_PROPERTY\_DEVICE\_CLASS\_GEFORCE = 2**

## enum CUpti\_MetricPropertyID

Metric device properties.

Metric device properties describe device properties which are needed for a metric. Some of these properties can be collected using `cuDeviceGetAttribute`.

### Values

**CUPTI\_METRIC\_PROPERTY\_MULTIPROCESSOR\_COUNT**  
**CUPTI\_METRIC\_PROPERTY\_WARPS\_PER\_MULTIPROCESSOR**  
**CUPTI\_METRIC\_PROPERTY\_KERNEL\_GPU\_TIME**  
**CUPTI\_METRIC\_PROPERTY\_CLOCK\_RATE**  
**CUPTI\_METRIC\_PROPERTY\_FRAME\_BUFFER\_COUNT**  
**CUPTI\_METRIC\_PROPERTY\_GLOBAL\_MEMORY\_BANDWIDTH**  
**CUPTI\_METRIC\_PROPERTY\_PCIE\_LINK\_RATE**  
**CUPTI\_METRIC\_PROPERTY\_PCIE\_LINK\_WIDTH**  
**CUPTI\_METRIC\_PROPERTY\_PCIE\_GEN**  
**CUPTI\_METRIC\_PROPERTY\_DEVICE\_CLASS**  
**CUPTI\_METRIC\_PROPERTY\_FLOP\_SP\_PER\_CYCLE**  
**CUPTI\_METRIC\_PROPERTY\_FLOP\_DP\_PER\_CYCLE**  
**CUPTI\_METRIC\_PROPERTY\_L2\_UNITS**

## enum CUpti\_MetricValueKind

Kinds of metric values.

Metric values can be one of several different kinds. Corresponding to each kind is a member of the `CUpti_MetricValue` union. The metric value returned by `cuptiMetricGetValue` should be accessed using the appropriate member of that union based on its value kind.

### Values

**CUPTI\_METRIC\_VALUE\_KIND\_DOUBLE = 0**

The metric value is a 64-bit double.

**CUPTI\_METRIC\_VALUE\_KIND\_UINT64 = 1**

The metric value is a 64-bit unsigned integer.

**CUPTI\_METRIC\_VALUE\_KIND\_PERCENT = 2**

The metric value is a percentage represented by a 64-bit double. For example, 57.5% is represented by the value 57.5.

**CUPTI\_METRIC\_VALUE\_KIND\_THROUGHPUT = 3**

The metric value is a throughput represented by a 64-bit integer. The unit for throughput values is bytes/second.

**CUPTI\_METRIC\_VALUE\_KIND\_INT64 = 4**

The metric value is a 64-bit signed integer.

**CUPTI\_METRIC\_VALUE\_KIND\_UTILIZATION\_LEVEL = 5**

The metric value is a utilization level, as represented by

`CUpti_MetricValueUtilizationLevel`.

**CUPTI\_METRIC\_VALUE\_KIND\_FORCE\_INT = 0x7fffffff**

## enum `CUpti_MetricValueUtilizationLevel`

Enumeration of utilization levels for metrics values of kind

`CUPTI_METRIC_VALUE_KIND_UTILIZATION_LEVEL`. Utilization values can vary from IDLE (0) to MAX (10) but the enumeration only provides specific names for a few values.

### Values

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_IDLE = 0**

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_LOW = 2**

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_MID = 5**

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_HIGH = 8**

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_MAX = 10**

**CUPTI\_METRIC\_VALUE\_UTILIZATION\_FORCE\_INT = 0x7fffffff**

## typedef `uint32_t CUpti_MetricID`

ID for a metric.

A metric provides a measure of some aspect of the device.

## `CUptiResult cuptiDeviceEnumMetrics (CUdevice device, size_t *arraySizeBytes, CUpti_MetricID *metricArray)`

Get the metrics for a device.

### Parameters

#### **device**

The CUDA device

#### **arraySizeBytes**

The size of `metricArray` in bytes, and returns the number of bytes written to `metricArray`

#### **metricArray**

Returns the IDs of the metrics for the device

### Returns

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`

- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
  - ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- if `arraySizeBytes` or `metricArray` are NULL

### Description

Returns the metric IDs in `metricArray` for a device. The size of the `metricArray` buffer is given by `*arraySizeBytes`. The size of the `metricArray` buffer must be at least `numMetrics * sizeof(CUpti_MetricID)` or else all metric IDs will not be returned. The value returned in `*arraySizeBytes` contains the number of bytes returned in `metricArray`.

## CUptiResult cuptiDeviceGetNumMetrics (CUdevice device, uint32\_t \*numMetrics)

Get the number of metrics for a device.

### Parameters

#### device

The CUDA device

#### numMetrics

Returns the number of metrics available for the device

### Returns

- ▶ CUPTI\_SUCCESS
  - ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
  - ▶ CUPTI\_ERROR\_INVALID\_DEVICE
  - ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
- if `numMetrics` is NULL

### Description

Returns the number of metrics available for a device.

## CUptiResult cuptiEnumMetrics (size\_t \*arraySizeBytes, CUpti\_MetricID \*metricArray)

Get all the metrics available on any device.

### Parameters

#### arraySizeBytes

The size of `metricArray` in bytes, and returns the number of bytes written to `metricArray`

#### metricArray

Returns the IDs of the metrics

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `arraySizeBytes` or `metricArray` are NULL

### Description

Returns the metric IDs in `metricArray` for all CUDA-capable devices. The size of the `metricArray` buffer is given by `*arraySizeBytes`. The size of the `metricArray` buffer must be at least `numMetrics * sizeof(CUpti_MetricID)` or all metric IDs will not be returned. The value returned in `*arraySizeBytes` contains the number of bytes returned in `metricArray`.

## CUptiResult cuptiGetNumMetrics (uint32\_t \*numMetrics)

Get the total number of metrics available on any device.

### Parameters

#### numMetrics

Returns the number of metrics

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `numMetrics` is NULL

### Description

Returns the total number of metrics available on any CUDA-capable devices.

## CUptiResult cuptiMetricCreateEventGroupSets (CUcontext context, size\_t metricIdArraySizeBytes, CUpti\_MetricID \*metricIdArray, CUpti\_EventGroupSets \*\*eventGroupPasses)

For a set of metrics, get the grouping that indicates the number of passes and the event groups necessary to collect the events required for those metrics.

### Parameters

#### context

The context for event collection

#### metricIdArraySizeBytes

Size of the metricIdArray in bytes

#### metricIdArray

Array of metric IDs

#### eventGroupPasses

Returns a [CUpti\\_EventGroupSets](#) object that indicates the number of passes required to collect the events and the events to collect on each pass

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_CONTEXT
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `metricIdArray` or `eventGroupPasses` is NULL

### Description

For a set of metrics, get the grouping that indicates the number of passes and the event groups necessary to collect the events required for those metrics.

### See also:

[cuptiEventGroupSetsCreate](#) for details on event group set creation.

## CUptiResult cuptiMetricEnumEvents (CUpti\_MetricID metric, size\_t \*eventIdArraySizeBytes, CUpti\_EventID \*eventIdArray)

Get the events required to calculating a metric.

### Parameters

#### **metric**

ID of the metric

#### **eventIdArraySizeBytes**

The size of `eventIdArray` in bytes, and returns the number of bytes written to `eventIdArray`

#### **eventIdArray**

Returns the IDs of the events required to calculate `metric`

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `eventIdArraySizeBytes` or `eventIdArray` are NULL.

### Description

Gets the event IDs in `eventIdArray` required to calculate a `metric`. The size of the `eventIdArray` buffer is given by `*eventIdArraySizeBytes` and must be at least `numEvents * sizeof(CUpti_EventID)` or all events will not be returned. The value returned in `*eventIdArraySizeBytes` contains the number of bytes returned in `eventIdArray`.

## CUptiResult cuptiMetricEnumProperties (CUpti\_MetricID metric, size\_t \*propIdArraySizeBytes, CUpti\_MetricPropertyID \*propIdArray)

Get the properties required to calculating a metric.

### Parameters

#### **metric**

ID of the metric

**propIdArraySizeBytes**

The size of `propIdArray` in bytes, and returns the number of bytes written to `propIdArray`

**propIdArray**

Returns the IDs of the properties required to calculate `metric`

**Returns**

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `propIdArraySizeBytes` or `propIdArray` are NULL.

**Description**

Gets the property IDs in `propIdArray` required to calculate a `metric`. The size of the `propIdArray` buffer is given by `*propIdArraySizeBytes` and must be at least `numProp * sizeof(CUpti_DeviceAttribute)` or all properties will not be returned. The value returned in `*propIdArraySizeBytes` contains the number of bytes returned in `propIdArray`.

## CUptiResult cuptiMetricGetAttribute (CUpti\_MetricID metric, CUpti\_MetricAttribute attrib, size\_t \*valueSize, void \*value)

Get a metric attribute.

**Parameters****metric**

ID of the metric

**attrib**

The metric attribute to read

**valueSize**

The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

**value**

Returns the attribute's value

**Returns**

- ▶ CUPTI\_SUCCESS

- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if `valueSize` or `value` is NULL, or if `attrib` is not a metric attribute
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

### Description

Returns a metric attribute in `*value`. The size of the `value` buffer is given by `*valueSize`. The value returned in `*valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than `*valueSize`, then only the first `*valueSize` characters will be returned and there will be no terminating null byte.

## CUptiResult cuptiMetricGetIdFromName (CUdevice device, const char \*metricName, CUpti\_MetricID \*metric)

Find an metric by name.

### Parameters

#### **device**

The CUDA device

#### **metricName**

The name of metric to find

#### **metric**

Returns the ID of the found metric or undefined if unable to find the metric

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_DEVICE
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_NAME
  - if unable to find a metric with name `metricName`. In this case `*metric` is undefined
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER



if `metricName` or `metric` are NULL.

### Description

Find a metric by name and return the metric ID in `*metric`.

## CUptiResult cuptiMetricGetNumEvents (CUpti\_MetricID metric, uint32\_t \*numEvents)

Get number of events required to calculate a metric.

### Parameters

#### **metric**

ID of the metric

#### **numEvents**

Returns the number of events required for the metric

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER

if `numEvents` is NULL

### Description

Returns the number of events in `numEvents` that are required to calculate a metric.

## CUptiResult cuptiMetricGetNumProperties (CUpti\_MetricID metric, uint32\_t \*numProp)

Get number of properties required to calculate a metric.

### Parameters

#### **metric**

ID of the metric

#### **numProp**

Returns the number of properties required for the metric

### Returns

- ▶ CUPTI\_SUCCESS

- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER
  - if numProp is NULL

### Description

Returns the number of properties in numProp that are required to calculate a metric.

## CUptiResult cuptiMetricGetRequiredEventGroupSets (CUcontext context, CUpti\_MetricID metric, CUpti\_EventGroupSets \*\*eventGroupSets)

For a metric get the groups of events that must be collected in the same pass.

### Parameters

#### context

The context for event collection

#### metric

The metric ID

#### eventGroupSets

Returns a CUpti\_EventGroupSets object that indicates the events that must be collected in the same pass to ensure the metric is calculated correctly. Returns NULL if no grouping is required for metric

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID

### Description

For a metric get the groups of events that must be collected in the same pass to ensure that the metric is calculated correctly. If the events are not collected as specified then the metric value may be inaccurate.

The function returns NULL if a metric does not have any required event group. In this case the events needed for the metric can be grouped in any manner for collection.

**CUptiResult cuptiMetricGetValue (CUdevice device, CUpti\_MetricID metric, size\_t eventIdArraySizeBytes, CUpti\_EventID \*eventIdArray, size\_t eventValueArraySizeBytes, uint64\_t \*eventValueArray, uint64\_t timeDuration, CUpti\_MetricValue \*metricValue)**

Calculate the value for a metric.

### Parameters

#### **device**

The CUDA device that the metric is being calculated for

#### **metric**

The metric ID

#### **eventIdArraySizeBytes**

The size of `eventIdArray` in bytes

#### **eventIdArray**

The event IDs required to calculate `metric`

#### **eventValueArraySizeBytes**

The size of `eventValueArray` in bytes

#### **eventValueArray**

The normalized event values required to calculate `metric`. The values must be order to match the order of events in `eventIdArray`

#### **timeDuration**

The duration over which the events were collected, in ns

#### **metricValue**

Returns the value for the metric

### Returns

- ▶ CUPTI\_SUCCESS
- ▶ CUPTI\_ERROR\_NOT\_INITIALIZED
- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_ID
- ▶ CUPTI\_ERROR\_INVALID\_OPERATION
- ▶ CUPTI\_ERROR\_PARAMETER\_SIZE\_NOT\_SUFFICIENT
  - if the `eventIdArray` does not contain all the events needed for `metric`
- ▶ CUPTI\_ERROR\_INVALID\_EVENT\_VALUE
  - if any of the event values required for the metric is `CUPTI_EVENT_OVERFLOW`

- ▶ CUPTI\_ERROR\_INVALID\_METRIC\_VALUE  
if the computed metric value cannot be represented in the metric's value type. For example, if the metric value type is unsigned and the computed metric value is negative
- ▶ CUPTI\_ERROR\_INVALID\_PARAMETER  
if `metricValue`, `eventIdArray` or `eventValueArray` is NULL

### Description

Use the events collected for a metric to calculate the metric value. Metric value evaluation depends on the evaluation mode `CUpti_MetricEvaluationMode` that the metric supports. If a metric has evaluation mode as `CUPTI_METRIC_EVALUATION_MODE_PER_INSTANCE`, then it assumes that the input event value is for one domain instance. If a metric has evaluation mode as `CUPTI_METRIC_EVALUATION_MODE_AGGREGATE`, it assumes that input event values are normalized to represent all domain instances on a device. For the most accurate metric collection, the events required for the metric should be collected for all profiled domain instances. For example, to collect all instances of an event, set the `CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES` attribute on the group containing the event to 1. The normalized value for the event is then:  $(\text{sum\_event\_values} * \text{totalInstanceCount}) / \text{instanceCount}$ , where `sum_event_values` is the summation of the event values across all profiled domain instances, `totalInstanceCount` is obtained from querying `CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT` and `instanceCount` is obtained from querying `CUPTI_EVENT_GROUP_ATTR_INSTANCE_COUNT` (or `CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT`).

**CUptiResult cuptiMetricGetValue2 (CUpti\_MetricID metric, size\_t eventIdArraySizeBytes, CUpti\_EventID \*eventIdArray, size\_t eventValueArraySizeBytes, uint64\_t \*eventValueArray, size\_t propIdArraySizeBytes, CUpti\_MetricPropertyID \*propIdArray, size\_t propValueArraySizeBytes, uint64\_t \*propValueArray, CUpti\_MetricValue \*metricValue)**

Calculate the value for a metric.

### Parameters

#### **metric**

The metric ID

**eventIdArraySizeBytes**

The size of `eventIdArray` in bytes

**eventIdArray**

The event IDs required to calculate `metric`

**eventValueArraySizeBytes**

The size of `eventValueArray` in bytes

**eventValueArray**

The normalized event values required to calculate `metric`. The values must be order to match the order of events in `eventIdArray`

**propIdArraySizeBytes**

The size of `propIdArray` in bytes

**propIdArray**

The metric property IDs required to calculate `metric`

**propValueArraySizeBytes**

The size of `propValueArray` in bytes

**propValueArray**

The metric property values required to calculate `metric`. The values must be order to match the order of metric properties in `propIdArray`

**metricValue**

Returns the value for the metric

**Returns**

- ▶ `CUPTI_SUCCESS`
- ▶ `CUPTI_ERROR_NOT_INITIALIZED`
- ▶ `CUPTI_ERROR_INVALID_METRIC_ID`
- ▶ `CUPTI_ERROR_INVALID_OPERATION`
- ▶ `CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT`
  - if the `eventIdArray` does not contain all the events needed for `metric`
- ▶ `CUPTI_ERROR_INVALID_EVENT_VALUE`
  - if any of the event values required for the metric is `CUPTI_EVENT_OVERFLOW`
- ▶ `CUPTI_ERROR_NOT_COMPATIBLE`
  - if the computed metric value cannot be represented in the metric's value type. For example, if the metric value type is unsigned and the computed metric value is negative
- ▶ `CUPTI_ERROR_INVALID_PARAMETER`
  - if `metricValue`, `eventIdArray` or `eventValueArray` is `NULL`

## Description

Use the events and properties collected for a metric to calculate the metric value. Metric value evaluation depends on the evaluation mode `CUpti_MetricEvaluationMode` that the metric supports. If a metric has evaluation mode as `CUPTI_METRIC_EVALUATION_MODE_PER_INSTANCE`, then it assumes that the input event value is for one domain instance. If a metric has evaluation mode as `CUPTI_METRIC_EVALUATION_MODE_AGGREGATE`, it assumes that input event values are normalized to represent all domain instances on a device. For the most accurate metric collection, the events required for the metric should be collected for all profiled domain instances. For example, to collect all instances of an event, set the `CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES` attribute on the group containing the event to 1. The normalized value for the event is then:  $(\text{sum\_event\_values} * \text{totalInstanceCount}) / \text{instanceCount}$ , where `sum_event_values` is the summation of the event values across all profiled domain instances, `totalInstanceCount` is obtained from querying `CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT` and `instanceCount` is obtained from querying `CUPTI_EVENT_GROUP_ATTR_INSTANCE_COUNT` (or `CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT`).

# Chapter 3.

## DATA STRUCTURES

Here are the data structures with brief descriptions:

### **CUpti\_Activity**

The base activity record

### **CUpti\_ActivityAPI**

The activity record for a driver or runtime API invocation

### **CUpti\_ActivityAutoBoostState**

Device auto boost state structure

### **CUpti\_ActivityBranch**

The activity record for source level result branch. (deprecated)

### **CUpti\_ActivityBranch2**

The activity record for source level result branch

### **CUpti\_ActivityCdpKernel**

The activity record for CDP (CUDA Dynamic Parallelism) kernel

### **CUpti\_ActivityContext**

The activity record for a context

### **CUpti\_ActivityDevice**

The activity record for a device

### **CUpti\_ActivityDeviceAttribute**

The activity record for a device attribute

### **CUpti\_ActivityEnvironment**

The activity record for CUPTI environmental data

### **CUpti\_ActivityEvent**

The activity record for a CUPTI event

### **CUpti\_ActivityEventInstance**

The activity record for a CUPTI event with instance information

### **CUpti\_ActivityFunction**

The activity record for global/device functions

### **CUpti\_ActivityGlobalAccess**

The activity record for source-level global access. (deprecated)

**CUpti\_ActivityGlobalAccess2**

The activity record for source-level global access

**CUpti\_ActivityInstructionExecution**

The activity record for source-level sass/source line-by-line correlation

**CUpti\_ActivityKernel**

The activity record for kernel. (deprecated)

**CUpti\_ActivityKernel2**

The activity record for a kernel (CUDA 5.5 onwards)

**CUpti\_ActivityMarker**

The activity record providing a marker which is an instantaneous point in time

**CUpti\_ActivityMarkerData**

The activity record providing detailed information for a marker

**CUpti\_ActivityMemcpy**

The activity record for memory copies

**CUpti\_ActivityMemcpy2**

The activity record for peer-to-peer memory copies

**CUpti\_ActivityMemset**

The activity record for memset

**CUpti\_ActivityMetric**

The activity record for a CUPTI metric

**CUpti\_ActivityMetricInstance**

The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (CUPTI\_ACTIVITY\_KIND\_METRIC\_INSTANCE). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect metric data may choose to use this type to store the collected metric data. This activity record should be used when metric domain instance information needs to be associated with the metric

**CUpti\_ActivityModule**

The activity record for a CUDA module

**CUpti\_ActivityName**

The activity record providing a name

**CUpti\_ActivityObjectKindId**

Identifiers for object kinds as specified by CUpti\_ActivityObjectKind

**CUpti\_ActivityOverhead**

The activity record for CUPTI and driver overheads

**CUpti\_ActivityPreemption**

The activity record for a preemption of a CDP kernel

**CUpti\_ActivitySharedAccess**

The activity record for source-level shared access

**CUpti\_ActivitySourceLocator**

The activity record for source locator



**CUpti\_ActivityUnifiedMemoryCounter**

The activity record for Unified Memory counters

**CUpti\_ActivityUnifiedMemoryCounterConfig**

Unified Memory counters configuration structure

**CUpti\_CallbackData**

Data passed into a runtime or driver API callback function

**CUpti\_EventGroupSet**

A set of event groups

**CUpti\_EventGroupSets**

A set of event group sets

**CUpti\_MetricValue**

A metric value

**CUpti\_ModuleResourceData**

Module data passed into a resource callback function

**CUpti\_NvtxData**

Data passed into a NVTX callback function

**CUpti\_ResourceData**

Data passed into a resource callback function

**CUpti\_SynchronizeData**

Data passed into a synchronize callback function

## 3.1. CUpti\_Activity Struct Reference

The base activity record.

The activity API uses a [CUpti\\_Activity](#) as a generic representation for any activity.

The 'kind' field is used to determine the specific activity kind, and from that the [CUpti\\_Activity](#) object can be cast to the specific activity record type appropriate for that kind.

Note that all activity record types are padded and aligned to ensure that each member of the record is naturally aligned.

**See also:**

[CUpti\\_ActivityKind](#)

### CUpti\_ActivityKind CUpti\_Activity::kind

The kind of this activity.

## 3.2. CUpti\_ActivityAPI Struct Reference

The activity record for a driver or runtime API invocation.

This activity record represents an invocation of a driver or runtime API (CUPTI\_ACTIVITY\_KIND\_DRIVER and CUPTI\_ACTIVITY\_KIND\_RUNTIME).

### CUpti\_CallbackId CUpti\_ActivityAPI::cbid

The ID of the driver or runtime function.

### uint32\_t CUpti\_ActivityAPI::correlationId

The correlation ID of the driver or runtime CUDA function. Each function invocation is assigned a unique correlation ID that is identical to the correlation ID in the memcpy, memset, or kernel activity record that is associated with this function.

### uint64\_t CUpti\_ActivityAPI::end

The end timestamp for the function, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the function.

### CUpti\_ActivityKind CUpti\_ActivityAPI::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_DRIVER or CUPTI\_ACTIVITY\_KIND\_RUNTIME.

### uint32\_t CUpti\_ActivityAPI::processId

The ID of the process where the driver or runtime CUDA function is executing.

### uint32\_t CUpti\_ActivityAPI::returnValue

The return value for the function. For a CUDA driver function with will be a CUresult value, and for a CUDA runtime function this will be a cudaError\_t value.

### uint64\_t CUpti\_ActivityAPI::start

The start timestamp for the function, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the function.

### uint32\_t CUpti\_ActivityAPI::threadId

The ID of the thread where the driver or runtime CUDA function is executing.

### 3.3. CUpti\_ActivityAutoBoostState Struct Reference

Device auto boost state structure.

This structure defines auto boost state for a device. See function /ref `cuptiGetAutoBoostState`

#### `uint32_t CUpti_ActivityAutoBoostState::enabled`

Returned auto boost state. 1 is returned in case auto boost is enabled, 0 otherwise

#### `uint32_t CUpti_ActivityAutoBoostState::pid`

Id of process that has set the current boost state. The value will be `CUPTI_AUTO_BOOST_INVALID_CLIENT_PID` if the user does not have the permission to query process ids or there is an error in querying the process id.

### 3.4. CUpti\_ActivityBranch Struct Reference

The activity record for source level result branch. (deprecated).

This activity record the locations of the branches in the source (`CUPTI_ACTIVITY_KIND_BRANCH`). Branch activities are now reported using the `CUpti_ActivityBranch2` activity record.

#### `uint32_t CUpti_ActivityBranch::correlationId`

The correlation ID of the kernel to which this result is associated.

#### `uint32_t CUpti_ActivityBranch::diverged`

Number of times this branch diverged

#### `uint32_t CUpti_ActivityBranch::executed`

The number of times this branch was executed

#### `CUpti_ActivityKind CUpti_ActivityBranch::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_BRANCH`.

## `uint32_t CUpti_ActivityBranch::pcOffset`

The pc offset for the branch.

## `uint32_t CUpti_ActivityBranch::sourceLocatorId`

The ID for source locator.

## `uint64_t CUpti_ActivityBranch::threadsExecuted`

This increments each time when this instruction is executed by number of threads that executed this instruction

## 3.5. `CUpti_ActivityBranch2` Struct Reference

The activity record for source level result branch.

This activity record the locations of the branches in the source (`CUPTI_ACTIVITY_KIND_BRANCH`).

## `uint32_t CUpti_ActivityBranch2::correlationId`

The correlation ID of the kernel to which this result is associated.

## `uint32_t CUpti_ActivityBranch2::diverged`

Number of times this branch diverged

## `uint32_t CUpti_ActivityBranch2::executed`

The number of times this branch was executed

## `uint32_t CUpti_ActivityBranch2::functionId`

Correlation ID with global/device function name

## `CUpti_ActivityKind CUpti_ActivityBranch2::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_BRANCH`.

## `uint32_t CUpti_ActivityBranch2::pad`

Undefined. Reserved for internal use.

## `uint32_t CUpti_ActivityBranch2::pcOffset`

The pc offset for the branch.

## `uint32_t CUpti_ActivityBranch2::sourceLocatorId`

The ID for source locator.

## `uint64_t CUpti_ActivityBranch2::threadsExecuted`

This increments each time when this instruction is executed by number of threads that executed this instruction

## 3.6. `CUpti_ActivityCdpKernel` Struct Reference

The activity record for CDP (CUDA Dynamic Parallelism) kernel.

This activity record represents a CDP kernel execution.

## `int32_t CUpti_ActivityCdpKernel::blockX`

The X-dimension block size for the kernel.

## `int32_t CUpti_ActivityCdpKernel::blockY`

The Y-dimension block size for the kernel.

## `int32_t CUpti_ActivityCdpKernel::blockZ`

The Z-dimension grid size for the kernel.

## `uint64_t CUpti_ActivityCdpKernel::completed`

The timestamp when kernel is marked as completed, in ns. A value of `CUPTI_TIMESTAMP_UNKNOWN` indicates that the completion time is unknown.

## `uint32_t CUpti_ActivityCdpKernel::contextId`

The ID of the context where the kernel is executing.

## `uint32_t CUpti_ActivityCdpKernel::correlationId`

The correlation ID of the kernel. Each kernel execution is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the kernel.

## `uint32_t CUpti_ActivityCdpKernel::deviceId`

The ID of the device where the kernel is executing.

## `int32_t`

## `CUpti_ActivityCdpKernel::dynamicSharedMemory`

The dynamic shared memory reserved for the kernel, in bytes.

## `uint64_t CUpti_ActivityCdpKernel::end`

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `uint8_t CUpti_ActivityCdpKernel::executed`

The cache configuration used for the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `int64_t CUpti_ActivityCdpKernel::gridId`

The grid ID of the kernel. Each kernel execution is assigned a unique grid ID.

## `int32_t CUpti_ActivityCdpKernel::gridX`

The X-dimension grid size for the kernel.

## `int32_t CUpti_ActivityCdpKernel::gridY`

The Y-dimension grid size for the kernel.

## `int32_t CUpti_ActivityCdpKernel::gridZ`

The Z-dimension grid size for the kernel.

## `CUpti_ActivityKind CUpti_ActivityCdpKernel::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_CDP_KERNEL`

## `uint32_t`

## `CUpti_ActivityCdpKernel::localMemoryPerThread`

The amount of local memory reserved for each thread, in bytes.

## `uint32_t CUpti_ActivityCdpKernel::localMemoryTotal`

The total amount of local memory reserved for the kernel, in bytes.

## `const char *CUpti_ActivityCdpKernel::name`

The name of the kernel. This name is shared across all activity records representing the same kernel, and so should not be modified.

## `uint32_t CUpti_ActivityCdpKernel::parentBlockX`

The X-dimension of the parent block.

## `uint32_t CUpti_ActivityCdpKernel::parentBlockY`

The Y-dimension of the parent block.

## `uint32_t CUpti_ActivityCdpKernel::parentBlockZ`

The Z-dimension of the parent block.

## `int64_t CUpti_ActivityCdpKernel::parentGridId`

The grid ID of the parent kernel.

## `uint64_t CUpti_ActivityCdpKernel::queued`

The timestamp when kernel is queued up, in ns. A value of `CUPTI_TIMESTAMP_UNKNOWN` indicates that the queued time is unknown.

## `uint16_t CUpti_ActivityCdpKernel::registersPerThread`

The number of registers required for each thread executing the kernel.

## `uint8_t CUpti_ActivityCdpKernel::requested`

The cache configuration requested by the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `uint8_t CUpti_ActivityCdpKernel::sharedMemoryConfig`

The shared memory configuration used for the kernel. The value is one of the `CUsharedconfig` enumeration values from `cuda.h`.

## `uint64_t CUpti_ActivityCdpKernel::start`

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `int32_t CUpti_ActivityCdpKernel::staticSharedMemory`

The static shared memory allocated for the kernel, in bytes.

## `uint32_t CUpti_ActivityCdpKernel::streamId`

The ID of the stream where the kernel is executing.

## `uint64_t CUpti_ActivityCdpKernel::submitted`

The timestamp when kernel is submitted to the gpu, in ns. A value of `CUPTI_TIMESTAMP_UNKNOWN` indicates that the submission time is unknown.

## 3.7. `CUpti_ActivityContext` Struct Reference

The activity record for a context.

This activity record represents information about a context (`CUPTI_ACTIVITY_KIND_CONTEXT`).

## `uint16_t CUpti_ActivityContext::computeApiKind`

The compute API kind.

**See also:**

`CUpti_ActivityComputeApiKind`

## `uint32_t CUpti_ActivityContext::contextId`

The context ID.

## `uint32_t CUpti_ActivityContext::deviceId`

The device ID.

## `CUpti_ActivityKind CUpti_ActivityContext::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_CONTEXT`.



## `uint16_t CUpti_ActivityContext::nullStreamId`

The ID for the NULL stream in this context

## 3.8. `CUpti_ActivityDevice` Struct Reference

The activity record for a device.

This activity record represents information about a GPU device (`CUPTI_ACTIVITY_KIND_DEVICE`).

## `uint32_t CUpti_ActivityDevice::computeCapabilityMajor`

Compute capability for the device, major number.

## `uint32_t CUpti_ActivityDevice::computeCapabilityMinor`

Compute capability for the device, minor number.

## `uint32_t CUpti_ActivityDevice::constantMemorySize`

The amount of constant memory on the device, in bytes.

## `uint32_t CUpti_ActivityDevice::coreClockRate`

The core clock rate of the device, in kHz.

## `CUpti_ActivityFlag CUpti_ActivityDevice::flags`

The flags associated with the device.

**See also:**

`CUpti_ActivityFlag`

## `uint64_t CUpti_ActivityDevice::globalMemoryBandwidth`

The global memory bandwidth available on the device, in kBytes/sec.

## `uint64_t CUpti_ActivityDevice::globalMemorySize`

The amount of global memory on the device, in bytes.

## `uint32_t CUpti_ActivityDevice::id`

The device ID.

## `CUpti_ActivityKind CUpti_ActivityDevice::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_DEVICE`.

## `uint32_t CUpti_ActivityDevice::l2CacheSize`

The size of the L2 cache on the device, in bytes.

## `uint32_t CUpti_ActivityDevice::maxBlockDimX`

Maximum allowed X dimension for a block.

## `uint32_t CUpti_ActivityDevice::maxBlockDimY`

Maximum allowed Y dimension for a block.

## `uint32_t CUpti_ActivityDevice::maxBlockDimZ`

Maximum allowed Z dimension for a block.

## `uint32_t`

## `CUpti_ActivityDevice::maxBlocksPerMultiprocessor`

Maximum number of blocks that can be present on a multiprocessor at any given time.

## `uint32_t CUpti_ActivityDevice::maxGridDimX`

Maximum allowed X dimension for a grid.

## `uint32_t CUpti_ActivityDevice::maxGridDimY`

Maximum allowed Y dimension for a grid.

## `uint32_t CUpti_ActivityDevice::maxGridDimZ`

Maximum allowed Z dimension for a grid.

## `uint32_t CUpti_ActivityDevice::maxIPC`

The maximum "instructions per cycle" possible on each device multiprocessor.

## `uint32_t CUpti_ActivityDevice::maxRegistersPerBlock`

Maximum number of registers that can be allocated to a block.

## `uint32_t`

## `CUpti_ActivityDevice::maxSharedMemoryPerBlock`

Maximum amount of shared memory that can be assigned to a block, in bytes.

## `uint32_t CUpti_ActivityDevice::maxThreadsPerBlock`

Maximum number of threads allowed in a block.

## `uint32_t`

## `CUpti_ActivityDevice::maxWarpsPerMultiprocessor`

Maximum number of warps that can be present on a multiprocessor at any given time.

## `const char *CUpti_ActivityDevice::name`

The device name. This name is shared across all activity records representing instances of the device, and so should not be modified.

## `uint32_t CUpti_ActivityDevice::numMemcpyEngines`

Number of memory copy engines on the device.

## `uint32_t CUpti_ActivityDevice::numMultiprocessors`

Number of multiprocessors on the device.

## `uint32_t CUpti_ActivityDevice::numThreadsPerWarp`

The number of threads per warp on the device.

## 3.9. `CUpti_ActivityDeviceAttribute` Struct Reference

The activity record for a device attribute.

This activity record represents information about a GPU device: either a `CUpti_DeviceAttribute` or `CUdevice_attribute` value (`CUPTI_ACTIVITY_KIND_DEVICE_ATTRIBUTE`).

## CUpti\_ActivityDeviceAttribute::@6 CUpti\_ActivityDeviceAttribute::attribute

The attribute, either a CUpti\_DeviceAttribute or CUdevice\_attribute. Flag CUPTI\_ACTIVITY\_FLAG\_DEVICE\_ATTRIBUTE\_CUDEVICE is used to indicate what kind of attribute this is. If CUPTI\_ACTIVITY\_FLAG\_DEVICE\_ATTRIBUTE\_CUDEVICE is 1 then CUdevice\_attribute field is value, otherwise CUpti\_DeviceAttribute field is valid.

## uint32\_t CUpti\_ActivityDeviceAttribute::deviceId

The ID of the device that this attribute applies to.

## CUpti\_ActivityFlag CUpti\_ActivityDeviceAttribute::flags

The flags associated with the device.

**See also:**

[CUpti\\_ActivityFlag](#)

## CUpti\_ActivityKind CUpti\_ActivityDeviceAttribute::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_DEVICE\_ATTRIBUTE.

## CUpti\_ActivityDeviceAttribute::@7 CUpti\_ActivityDeviceAttribute::value

The value for the attribute. See CUpti\_DeviceAttribute and CUdevice\_attribute for the type of the value for a given attribute.

## 3.10. CUpti\_ActivityEnvironment Struct Reference

The activity record for CUPTI environmental data.

This activity record provides CUPTI environmental data, include power, clocks, and thermals. This information is sampled at various rates and returned in this activity record. The consumer of the record needs to check the environmentKind field to figure out what kind of environmental record this is.

**CUpti\_EnvironmentClocksThrottleReason**  
**CUpti\_ActivityEnvironment::clocksThrottleReasons**

The clocks throttle reasons.

**CUpti\_ActivityEnvironment::@8::@12**  
**CUpti\_ActivityEnvironment::cooling**

Data returned for CUPTI\_ACTIVITY\_ENVIRONMENT\_COOLING environment kind.

**uint32\_t CUpti\_ActivityEnvironment::deviceId**

The ID of the device

**CUpti\_ActivityEnvironmentKind**  
**CUpti\_ActivityEnvironment::environmentKind**

The kind of data reported in this record.

**uint32\_t CUpti\_ActivityEnvironment::fanSpeed**

The fan speed as percentage of maximum.

**uint32\_t CUpti\_ActivityEnvironment::gpuTemperature**

The GPU temperature in degrees C.

**CUpti\_ActivityKind CUpti\_ActivityEnvironment::kind**

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_ENVIRONMENT.

**uint32\_t CUpti\_ActivityEnvironment::memoryClock**

The memory frequency in MHz

**uint32\_t CUpti\_ActivityEnvironment::pcieLinkGen**

The PCIe link generation.

**uint32\_t CUpti\_ActivityEnvironment::pcieLinkWidth**

The PCIe link width.

## CUpti\_ActivityEnvironment::@8::@11

### CUpti\_ActivityEnvironment::power

Data returned for CUPTI\_ACTIVITY\_ENVIRONMENT\_POWER environment kind.

### uint32\_t CUpti\_ActivityEnvironment::power

The power in milliwatts consumed by GPU and associated circuitry.

### uint32\_t CUpti\_ActivityEnvironment::powerLimit

The power in milliwatts that will trigger power management algorithm.

### uint32\_t CUpti\_ActivityEnvironment::smClock

The SM frequency in MHz

## CUpti\_ActivityEnvironment::@8::@9

### CUpti\_ActivityEnvironment::speed

Data returned for CUPTI\_ACTIVITY\_ENVIRONMENT\_SPEED environment kind.

## CUpti\_ActivityEnvironment::@8::@10

### CUpti\_ActivityEnvironment::temperature

Data returned for CUPTI\_ACTIVITY\_ENVIRONMENT\_TEMPERATURE environment kind.

### uint64\_t CUpti\_ActivityEnvironment::timestamp

The timestamp when this sample was retrieved, in ns. A value of 0 indicates that timestamp information could not be collected for the marker.

## 3.11. CUpti\_ActivityEvent Struct Reference

The activity record for a CUPTI event.

This activity record represents a CUPTI event value (CUPTI\_ACTIVITY\_KIND\_EVENT). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect event data may choose to use this type to store the collected event data.

## `uint32_t CUpti_ActivityEvent::correlationId`

The correlation ID of the event. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the event was gathered.

## `CUpti_EventDomainID CUpti_ActivityEvent::domain`

The event domain ID.

## `CUpti_EventID CUpti_ActivityEvent::id`

The event ID.

## `CUpti_ActivityKind CUpti_ActivityEvent::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_EVENT`.

## `uint64_t CUpti_ActivityEvent::value`

The event value.

## 3.12. `CUpti_ActivityEventInstance` Struct Reference

The activity record for a CUPTI event with instance information.

This activity record represents the a CUPTI event value for a specific event domain instance (`CUPTI_ACTIVITY_KIND_EVENT_INSTANCE`). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect event data may choose to use this type to store the collected event data. This activity record should be used when event domain instance information needs to be associated with the event.

## `uint32_t CUpti_ActivityEventInstance::correlationId`

The correlation ID of the event. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the event was gathered.

## `CUpti_EventDomainID CUpti_ActivityEventInstance::domain`

The event domain ID.

**CUpti\_EventID CUpti\_ActivityEventInstance::id**

The event ID.

**uint32\_t CUpti\_ActivityEventInstance::instance**

The event domain instance.

**CUpti\_ActivityKind CUpti\_ActivityEventInstance::kind**

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_EVENT\_INSTANCE.

**uint32\_t CUpti\_ActivityEventInstance::pad**

Undefined. Reserved for internal use.

**uint64\_t CUpti\_ActivityEventInstance::value**

The event value.

**3.13. CUpti\_ActivityFunction Struct Reference**

The activity record for global/device functions.

This activity records function name and corresponding module information. (CUPTI\_ACTIVITY\_KIND\_FUNCTION).

**uint32\_t CUpti\_ActivityFunction::contextId**

The ID of the context where the function is launched.

**uint32\_t CUpti\_ActivityFunction::functionIndex**

The function's unique symbol index in the module.

**uint32\_t CUpti\_ActivityFunction::id**

ID to uniquely identify the record

**CUpti\_ActivityKind CUpti\_ActivityFunction::kind**

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_FUNCTION.



## `uint32_t CUpti_ActivityFunction::moduleId`

The module ID in which this global/device function is present.

## `const char *CUpti_ActivityFunction::name`

The name of the function. This name is shared across all activity records representing the same kernel, and so should not be modified.

## 3.14. `CUpti_ActivityGlobalAccess` Struct Reference

The activity record for source-level global access. (deprecated).

This activity records the locations of the global accesses in the source (`CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS`). Global access activities are now reported using the `CUpti_ActivityGlobalAccess2` activity record.

## `uint32_t CUpti_ActivityGlobalAccess::correlationId`

The correlation ID of the kernel to which this result is associated.

## `uint32_t CUpti_ActivityGlobalAccess::executed`

The number of times this instruction was executed

## `CUpti_ActivityFlag CUpti_ActivityGlobalAccess::flags`

The properties of this global access.

## `CUpti_ActivityKind CUpti_ActivityGlobalAccess::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS`.

## `uint64_t CUpti_ActivityGlobalAccess::l2_transactions`

The total number of 32 bytes transactions to L2 cache generated by this access

## `uint32_t CUpti_ActivityGlobalAccess::pcOffset`

The pc offset for the access.

## `uint32_t CUpti_ActivityGlobalAccess::sourceLocatorId`

The ID for source locator.

## `uint64_t CUpti_ActivityGlobalAccess::threadsExecuted`

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

## 3.15. `CUpti_ActivityGlobalAccess2` Struct Reference

The activity record for source-level global access.

This activity records the locations of the global accesses in the source (`CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS`).

## `uint32_t CUpti_ActivityGlobalAccess2::correlationId`

The correlation ID of the kernel to which this result is associated.

## `uint32_t CUpti_ActivityGlobalAccess2::executed`

The number of times this instruction was executed

## `CUpti_ActivityFlag CUpti_ActivityGlobalAccess2::flags`

The properties of this global access.

## `uint32_t CUpti_ActivityGlobalAccess2::functionId`

Correlation ID with global/device function name

## `CUpti_ActivityKind CUpti_ActivityGlobalAccess2::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS`.

## `uint64_t CUpti_ActivityGlobalAccess2::l2_transactions`

The total number of 32 bytes transactions to L2 cache generated by this access

## `uint32_t CUpti_ActivityGlobalAccess2::pad`

Undefined. Reserved for internal use.

## `uint32_t CUpti_ActivityGlobalAccess2::pcOffset`

The pc offset for the access.

## `uint32_t CUpti_ActivityGlobalAccess2::sourceLocatorId`

The ID for source locator.

## `uint64_t`

## `CUpti_ActivityGlobalAccess2::theoreticalL2Transactions`

The minimum number of L2 transactions possible based on the access pattern.

## `uint64_t CUpti_ActivityGlobalAccess2::threadsExecuted`

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

## 3.16. `CUpti_ActivityInstructionExecution` Struct Reference

The activity record for source-level sass/source line-by-line correlation.

This activity records source level sass/source correlation information. (`CUPTI_ACTIVITY_KIND_INSTRUCTION_EXECUTION`).

## `uint32_t`

## `CUpti_ActivityInstructionExecution::correlationId`

The correlation ID of the kernel to which this result is associated.

## `uint32_t CUpti_ActivityInstructionExecution::executed`

The number of times this instruction was executed.

## `CUpti_ActivityFlag`

## `CUpti_ActivityInstructionExecution::flags`

The properties of this instruction execution. Check mask `CUPTI_ACTIVITY_FLAG_INSTRUCTION_VALUE_INVALID` to determine whether `threadsExecuted`, `notPredOffThreadsExecuted` and `executed` are valid for the instruction. Check mask `CUPTI_ACTIVITY_FLAG_INSTRUCTION_CLASS_MASK` to identify the instruction class. See [CUpti\\_ActivityInstructionClass](#).

## uint32\_t CUpti\_ActivityInstructionExecution::functionId

Correlation ID with global/device function name

## CUpti\_ActivityKind

## CUpti\_ActivityInstructionExecution::kind

The activity record kind, must be  
CUPTI\_ACTIVITY\_KIND\_INSTRUCTION\_EXECUTION.

## uint64\_t

## CUpti\_ActivityInstructionExecution::notPredOffThreadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

## uint32\_t CUpti\_ActivityInstructionExecution::pad

Undefined. Reserved for internal use.

## uint32\_t CUpti\_ActivityInstructionExecution::pcOffset

The pc offset for the instruction.

## uint32\_t

## CUpti\_ActivityInstructionExecution::sourceLocatorId

The ID for source locator.

## uint64\_t

## CUpti\_ActivityInstructionExecution::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction, regardless of predicate or condition code.

## 3.17. CUpti\_ActivityKernel Struct Reference

The activity record for kernel. (deprecated).

This activity record represents a kernel execution (CUPTI\_ACTIVITY\_KIND\_KERNEL and CUPTI\_ACTIVITY\_KIND\_CONCURRENT\_KERNEL) but is no longer generated by CUPTI. Kernel activities are now reported using the [CUpti\\_ActivityKernel2](#) activity record.

## `int32_t CUpti_ActivityKernel::blockX`

The X-dimension block size for the kernel.

## `int32_t CUpti_ActivityKernel::blockY`

The Y-dimension block size for the kernel.

## `int32_t CUpti_ActivityKernel::blockZ`

The Z-dimension grid size for the kernel.

## `uint8_t CUpti_ActivityKernel::cacheConfigExecuted`

The cache configuration used for the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `uint8_t CUpti_ActivityKernel::cacheConfigRequested`

The cache configuration requested by the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `uint32_t CUpti_ActivityKernel::contextId`

The ID of the context where the kernel is executing.

## `uint32_t CUpti_ActivityKernel::correlationId`

The correlation ID of the kernel. Each kernel execution is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the kernel.

## `uint32_t CUpti_ActivityKernel::deviceId`

The ID of the device where the kernel is executing.

## `int32_t CUpti_ActivityKernel::dynamicSharedMemory`

The dynamic shared memory reserved for the kernel, in bytes.

## `uint64_t CUpti_ActivityKernel::end`

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `int32_t CUpti_ActivityKernel::gridX`

The X-dimension grid size for the kernel.

## `int32_t CUpti_ActivityKernel::gridY`

The Y-dimension grid size for the kernel.

## `int32_t CUpti_ActivityKernel::gridZ`

The Z-dimension grid size for the kernel.

## `CUpti_ActivityKind CUpti_ActivityKernel::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_KERNEL` or `CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL`.

## `uint32_t CUpti_ActivityKernel::localMemoryPerThread`

The amount of local memory reserved for each thread, in bytes.

## `uint32_t CUpti_ActivityKernel::localMemoryTotal`

The total amount of local memory reserved for the kernel, in bytes.

## `const char *CUpti_ActivityKernel::name`

The name of the kernel. This name is shared across all activity records representing the same kernel, and so should not be modified.

## `uint32_t CUpti_ActivityKernel::pad`

Undefined. Reserved for internal use.

## `uint16_t CUpti_ActivityKernel::registersPerThread`

The number of registers required for each thread executing the kernel.

## `void *CUpti_ActivityKernel::reserved0`

Undefined. Reserved for internal use.

## `uint32_t CUpti_ActivityKernel::runtimeCorrelationId`

The runtime correlation ID of the kernel. Each kernel execution is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the kernel.

## `uint64_t CUpti_ActivityKernel::start`

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `int32_t CUpti_ActivityKernel::staticSharedMemory`

The static shared memory allocated for the kernel, in bytes.

## `uint32_t CUpti_ActivityKernel::streamId`

The ID of the stream where the kernel is executing.

## 3.18. `CUpti_ActivityKernel2` Struct Reference

The activity record for a kernel (CUDA 5.5 onwards).

This activity record represents a kernel execution (`CUPTI_ACTIVITY_KIND_KERNEL` and `CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL`).

## `int32_t CUpti_ActivityKernel2::blockX`

The X-dimension block size for the kernel.

## `int32_t CUpti_ActivityKernel2::blockY`

The Y-dimension block size for the kernel.

## `int32_t CUpti_ActivityKernel2::blockZ`

The Z-dimension grid size for the kernel.

## `uint64_t CUpti_ActivityKernel2::completed`

The completed timestamp for the kernel execution, in ns. It represents the completion of all it's child kernels and the kernel itself. A value of `CUPTI_TIMESTAMP_UNKNOWN` indicates that the completion time is unknown.

## `uint32_t CUpti_ActivityKernel2::contextId`

The ID of the context where the kernel is executing.

## `uint32_t CUpti_ActivityKernel2::correlationId`

The correlation ID of the kernel. Each kernel execution is assigned a unique correlation ID that is identical to the correlation ID in the driver or runtime API activity record that launched the kernel.

## `uint32_t CUpti_ActivityKernel2::deviceId`

The ID of the device where the kernel is executing.

## `int32_t CUpti_ActivityKernel2::dynamicSharedMemory`

The dynamic shared memory reserved for the kernel, in bytes.

## `uint64_t CUpti_ActivityKernel2::end`

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `uint8_t CUpti_ActivityKernel2::executed`

The cache configuration used for the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `int64_t CUpti_ActivityKernel2::gridId`

The grid ID of the kernel. Each kernel is assigned a unique grid ID at runtime.

## `int32_t CUpti_ActivityKernel2::gridX`

The X-dimension grid size for the kernel.

## `int32_t CUpti_ActivityKernel2::gridY`

The Y-dimension grid size for the kernel.

## `int32_t CUpti_ActivityKernel2::gridZ`

The Z-dimension grid size for the kernel.



## `CUpti_ActivityKind CUpti_ActivityKernel2::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_KERNEL` or `CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL`.

## `uint32_t CUpti_ActivityKernel2::localMemoryPerThread`

The amount of local memory reserved for each thread, in bytes.

## `uint32_t CUpti_ActivityKernel2::localMemoryTotal`

The total amount of local memory reserved for the kernel, in bytes.

## `const char *CUpti_ActivityKernel2::name`

The name of the kernel. This name is shared across all activity records representing the same kernel, and so should not be modified.

## `uint16_t CUpti_ActivityKernel2::registersPerThread`

The number of registers required for each thread executing the kernel.

## `uint8_t CUpti_ActivityKernel2::requested`

The cache configuration requested by the kernel. The value is one of the `CUfunc_cache` enumeration values from `cuda.h`.

## `void *CUpti_ActivityKernel2::reserved0`

Undefined. Reserved for internal use.

## `uint8_t CUpti_ActivityKernel2::sharedMemoryConfig`

The shared memory configuration used for the kernel. The value is one of the `CUsharedconfig` enumeration values from `cuda.h`.

## `uint64_t CUpti_ActivityKernel2::start`

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

## `int32_t CUpti_ActivityKernel2::staticSharedMemory`

The static shared memory allocated for the kernel, in bytes.

## `uint32_t CUpti_ActivityKernel2::streamId`

The ID of the stream where the kernel is executing.

## 3.19. CUpti\_ActivityMarker Struct Reference

The activity record providing a marker which is an instantaneous point in time.

The marker is specified with a descriptive name and unique id (CUPTI\_ACTIVITY\_KIND\_MARKER).

### `CUpti_ActivityFlag CUpti_ActivityMarker::flags`

The flags associated with the marker.

**See also:**

[CUpti\\_ActivityFlag](#)

### `uint32_t CUpti_ActivityMarker::id`

The marker ID.

### `CUpti_ActivityKind CUpti_ActivityMarker::kind`

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MARKER.

### `const char *CUpti_ActivityMarker::name`

The marker name for an instantaneous or start marker. This will be NULL for an end marker.

### `CUpti_ActivityMarker::objectId`

The identifier for the activity object associated with this marker. 'objectKind' indicates which ID is valid for this record.

### `CUpti_ActivityObjectKind`

### `CUpti_ActivityMarker::objectKind`

The kind of activity object associated with this marker.

## uint64\_t CUpti\_ActivityMarker::timestamp

The timestamp for the marker, in ns. A value of 0 indicates that timestamp information could not be collected for the marker.

## 3.20. CUpti\_ActivityMarkerData Struct Reference

The activity record providing detailed information for a marker.

The marker data contains color, payload, and category.  
(CUPTI\_ACTIVITY\_KIND\_MARKER\_DATA).

## uint32\_t CUpti\_ActivityMarkerData::category

The category for the marker.

## uint32\_t CUpti\_ActivityMarkerData::color

The color for the marker.

## CUpti\_ActivityFlag CUpti\_ActivityMarkerData::flags

The flags associated with the marker.

**See also:**

[CUpti\\_ActivityFlag](#)

## uint32\_t CUpti\_ActivityMarkerData::id

The marker ID.

## CUpti\_ActivityKind CUpti\_ActivityMarkerData::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MARKER\_DATA.

## CUpti\_ActivityMarkerData::payload

The payload value.

## CUpti\_MetricValueKind

## CUpti\_ActivityMarkerData::payloadKind

Defines the payload format for the value associated with the marker.

## 3.21. CUpti\_ActivityMemcpy Struct Reference

The activity record for memory copies.

This activity record represents a memory copy (CUPTI\_ACTIVITY\_KIND\_MEMCPY).

### uint64\_t CUpti\_ActivityMemcpy::bytes

The number of bytes transferred by the memory copy.

### uint32\_t CUpti\_ActivityMemcpy::contextId

The ID of the context where the memory copy is occurring.

### uint8\_t CUpti\_ActivityMemcpy::copyKind

The kind of the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemcpyKind](#)

### uint32\_t CUpti\_ActivityMemcpy::correlationId

The correlation ID of the memory copy. Each memory copy is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the memory copy.

### uint32\_t CUpti\_ActivityMemcpy::deviceId

The ID of the device where the memory copy is occurring.

### uint8\_t CUpti\_ActivityMemcpy::dstKind

The destination memory kind read by the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemoryKind](#)

## uint64\_t CUpti\_ActivityMemcpy::end

The end timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

## uint8\_t CUpti\_ActivityMemcpy::flags

The flags associated with the memory copy.

**See also:**

[CUpti\\_ActivityFlag](#)

## CUpti\_ActivityKind CUpti\_ActivityMemcpy::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MEMCPY.

## void \*CUpti\_ActivityMemcpy::reserved0

Undefined. Reserved for internal use.

## uint32\_t CUpti\_ActivityMemcpy::runtimeCorrelationId

The runtime correlation ID of the memory copy. Each memory copy is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the memory copy.

## uint8\_t CUpti\_ActivityMemcpy::srcKind

The source memory kind read by the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemoryKind](#)

## uint64\_t CUpti\_ActivityMemcpy::start

The start timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

## uint32\_t CUpti\_ActivityMemcpy::streamId

The ID of the stream where the memory copy is occurring.

## 3.22. CUpti\_ActivityMemcpy2 Struct Reference

The activity record for peer-to-peer memory copies.

This activity record represents a peer-to-peer memory copy (CUPTI\_ACTIVITY\_KIND\_MEMCPY2).

### uint64\_t CUpti\_ActivityMemcpy2::bytes

The number of bytes transferred by the memory copy.

### uint32\_t CUpti\_ActivityMemcpy2::contextId

The ID of the context where the memory copy is occurring.

### uint8\_t CUpti\_ActivityMemcpy2::copyKind

The kind of the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemcpyKind](#)

### uint32\_t CUpti\_ActivityMemcpy2::correlationId

The correlation ID of the memory copy. Each memory copy is assigned a unique correlation ID that is identical to the correlation ID in the driver and runtime API activity record that launched the memory copy.

### uint32\_t CUpti\_ActivityMemcpy2::deviceId

The ID of the device where the memory copy is occurring.

### uint32\_t CUpti\_ActivityMemcpy2::dstContextId

The ID of the context owning the memory being copied to.

### uint32\_t CUpti\_ActivityMemcpy2::dstDeviceId

The ID of the device where memory is being copied to.

### uint8\_t CUpti\_ActivityMemcpy2::dstKind

The destination memory kind read by the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemoryKind](#)

## uint64\_t CUpti\_ActivityMemcpy2::end

The end timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

## uint8\_t CUpti\_ActivityMemcpy2::flags

The flags associated with the memory copy.

**See also:**

[CUpti\\_ActivityFlag](#)

## CUpti\_ActivityKind CUpti\_ActivityMemcpy2::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MEMCPY2.

## uint32\_t CUpti\_ActivityMemcpy2::pad

Undefined. Reserved for internal use.

## void \*CUpti\_ActivityMemcpy2::reserved0

Undefined. Reserved for internal use.

## uint32\_t CUpti\_ActivityMemcpy2::srcContextId

The ID of the context owning the memory being copied from.

## uint32\_t CUpti\_ActivityMemcpy2::srcDeviceId

The ID of the device where memory is being copied from.

## uint8\_t CUpti\_ActivityMemcpy2::srcKind

The source memory kind read by the memory copy, stored as a byte to reduce record size.

**See also:**

[CUpti\\_ActivityMemoryKind](#)

## `uint64_t CUpti_ActivityMemcpy2::start`

The start timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

## `uint32_t CUpti_ActivityMemcpy2::streamId`

The ID of the stream where the memory copy is occurring.

## 3.23. CUpti\_ActivityMemset Struct Reference

The activity record for memset.

This activity record represents a memory set operation (CUPTI\_ACTIVITY\_KIND\_MEMSET).

## `uint64_t CUpti_ActivityMemset::bytes`

The number of bytes being set by the memory set.

## `uint32_t CUpti_ActivityMemset::contextId`

The ID of the context where the memory set is occurring.

## `uint32_t CUpti_ActivityMemset::correlationId`

The correlation ID of the memory set. Each memory set is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the memory set.

## `uint32_t CUpti_ActivityMemset::deviceId`

The ID of the device where the memory set is occurring.

## `uint64_t CUpti_ActivityMemset::end`

The end timestamp for the memory set, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory set.

## `CUpti_ActivityKind CUpti_ActivityMemset::kind`

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MEMSET.



## `void *CUpti_ActivityMemset::reserved0`

Undefined. Reserved for internal use.

## `uint32_t CUpti_ActivityMemset::runtimeCorrelationId`

The runtime correlation ID of the memory set. Each memory set is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the memory set.

## `uint64_t CUpti_ActivityMemset::start`

The start timestamp for the memory set, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory set.

## `uint32_t CUpti_ActivityMemset::streamId`

The ID of the stream where the memory set is occurring.

## `uint32_t CUpti_ActivityMemset::value`

The value being assigned to memory by the memory set.

## 3.24. CUpti\_ActivityMetric Struct Reference

The activity record for a CUPTI metric.

This activity record represents the collection of a CUPTI metric value (CUPTI\_ACTIVITY\_KIND\_METRIC). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect metric data may choose to use this type to store the collected metric data.

## `uint32_t CUpti_ActivityMetric::correlationId`

The correlation ID of the metric. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the metric was gathered.

## `uint8_t CUpti_ActivityMetric::flags`

The properties of this metric.

**See also:**

`CUpti_ActivityFlag`

`CUpti_MetricID CUpti_ActivityMetric::id`

The metric ID.

`CUpti_ActivityKind CUpti_ActivityMetric::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_METRIC`.

`uint8_t CUpti_ActivityMetric::pad`

Undefined. Reserved for internal use.

`CUpti_ActivityMetric::value`

The metric value.

## 3.25. `CUpti_ActivityMetricInstance` Struct Reference

The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (`CUPTI_ACTIVITY_KIND_METRIC_INSTANCE`). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect metric data may choose to use this type to store the collected metric data. This activity record should be used when metric domain instance information needs to be associated with the metric.

`uint32_t CUpti_ActivityMetricInstance::correlationId`

The correlation ID of the metric. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the metric was gathered.

`uint8_t CUpti_ActivityMetricInstance::flags`

The properties of this metric.

**See also:**

`CUpti_ActivityFlag`

## `CUpti_MetricID CUpti_ActivityMetricInstance::id`

The metric ID.

## `uint32_t CUpti_ActivityMetricInstance::instance`

The metric domain instance.

## `CUpti_ActivityKind CUpti_ActivityMetricInstance::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_METRIC_INSTANCE`.

## `uint8_t CUpti_ActivityMetricInstance::pad`

Undefined. Reserved for internal use.

## `CUpti_ActivityMetricInstance::value`

The metric value.

## 3.26. `CUpti_ActivityModule` Struct Reference

The activity record for a CUDA module.

This activity record represents a CUDA module (`CUPTI_ACTIVITY_KIND_MODULE`). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect module data from the module callback may choose to use this type to store the collected module data.

## `uint32_t CUpti_ActivityModule::contextId`

The ID of the context where the module is loaded.

## `const void *CUpti_ActivityModule::cubin`

The pointer to cubin.

## `uint32_t CUpti_ActivityModule::cubinSize`

The cubin size.

## `uint32_t CUpti_ActivityModule::id`

The module ID.

## CUpti\_ActivityKind CUpti\_ActivityModule::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_MODULE.

## uint32\_t CUpti\_ActivityModule::pad

Undefined. Reserved for internal use.

## 3.27. CUpti\_ActivityName Struct Reference

The activity record providing a name.

This activity record provides a name for a device, context, thread, etc. (CUPTI\_ACTIVITY\_KIND\_NAME).

### CUpti\_ActivityKind CUpti\_ActivityName::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_NAME.

### const char \*CUpti\_ActivityName::name

The name.

### CUpti\_ActivityName::objectId

The identifier for the activity object. 'objectKind' indicates which ID is valid for this record.

### CUpti\_ActivityObjectKind CUpti\_ActivityName::objectKind

The kind of activity object being named.

## 3.28. CUpti\_ActivityObjectKindId Union Reference

Identifiers for object kinds as specified by CUpti\_ActivityObjectKind.

**See also:**

[CUpti\\_ActivityObjectKind](#)

## CUpti\_ActivityObjectKindId::@1 CUpti\_ActivityObjectKindId::dcs

A device object requires that we identify the device ID. A context object requires that we identify both the device and context ID. A stream object requires that we identify device, context, and stream ID.

## CUpti\_ActivityObjectKindId::@0 CUpti\_ActivityObjectKindId::pt

A process object requires that we identify the process ID. A thread object requires that we identify both the process and thread ID.

## 3.29. CUpti\_ActivityOverhead Struct Reference

The activity record for CUPTI and driver overheads.

This activity record provides CUPTI and driver overhead information (CUPTI\_ACTIVITY\_OVERHEAD).

### uint64\_t CUpti\_ActivityOverhead::end

The end timestamp for the overhead, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the overhead.

### CUpti\_ActivityKind CUpti\_ActivityOverhead::kind

The activity record kind, must be CUPTI\_ACTIVITY\_OVERHEAD.

### CUpti\_ActivityOverhead::objectId

The identifier for the activity object. 'objectKind' indicates which ID is valid for this record.

### CUpti\_ActivityObjectKind CUpti\_ActivityOverhead::objectKind

The kind of activity object that the overhead is associated with.

## CUpti\_ActivityOverheadKind CUpti\_ActivityOverhead::overheadKind

The kind of overhead, CUPTI, DRIVER, COMPILER etc.

## uint64\_t CUpti\_ActivityOverhead::start

The start timestamp for the overhead, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the overhead.

### 3.30. CUpti\_ActivityPreemption Struct Reference

The activity record for a preemption of a CDP kernel.

This activity record represents a preemption of a CDP kernel.

## uint32\_t CUpti\_ActivityPreemption::blockX

The X-dimension of the block that is preempted

## uint32\_t CUpti\_ActivityPreemption::blockY

The Y-dimension of the block that is preempted

## uint32\_t CUpti\_ActivityPreemption::blockZ

The Z-dimension of the block that is preempted

## int64\_t CUpti\_ActivityPreemption::gridId

The grid-id of the block that is preempted

## CUpti\_ActivityKind CUpti\_ActivityPreemption::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_PREEMPTION

## uint32\_t CUpti\_ActivityPreemption::pad

Undefined. Reserved for internal use.

## CUpti\_ActivityPreemptionKind CUpti\_ActivityPreemption::preemptionKind

kind of the preemption

## uint64\_t CUpti\_ActivityPreemption::timestamp

The timestamp of the preemption, in ns. A value of 0 indicates that timestamp information could not be collected for the preemption.

## 3.31. CUpti\_ActivitySharedAccess Struct Reference

The activity record for source-level shared access.

This activity records the locations of the shared accesses in the source (CUPTI\_ACTIVITY\_KIND\_SHARED\_ACCESS).

## uint32\_t CUpti\_ActivitySharedAccess::correlationId

The correlation ID of the kernel to which this result is associated.

## uint32\_t CUpti\_ActivitySharedAccess::executed

The number of times this instruction was executed

## CUpti\_ActivityFlag CUpti\_ActivitySharedAccess::flags

The properties of this shared access.

## uint32\_t CUpti\_ActivitySharedAccess::functionId

Correlation ID with global/device function name

## CUpti\_ActivityKind CUpti\_ActivitySharedAccess::kind

The activity record kind, must be CUPTI\_ACTIVITY\_KIND\_SHARED\_ACCESS.

## uint32\_t CUpti\_ActivitySharedAccess::pad

Undefined. Reserved for internal use.

## `uint32_t CUpti_ActivitySharedAccess::pcOffset`

The pc offset for the access.

## `uint64_t`

## `CUpti_ActivitySharedAccess::sharedTransactions`

The total number of shared memory transactions generated by this access

## `uint32_t CUpti_ActivitySharedAccess::sourceLocatorId`

The ID for source locator.

## `uint64_t`

## `CUpti_ActivitySharedAccess::theoreticalSharedTransactions`

The minimum number of shared memory transactions possible based on the access pattern.

## `uint64_t CUpti_ActivitySharedAccess::threadsExecuted`

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

## 3.32. `CUpti_ActivitySourceLocator` Struct Reference

The activity record for source locator.

This activity record represents a source locator (`CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR`).

## `const char *CUpti_ActivitySourceLocator::fileName`

The path for the file.

## `uint32_t CUpti_ActivitySourceLocator::id`

The ID for the source path, will be used in all the source level results.

## `CUpti_ActivityKind CUpti_ActivitySourceLocator::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR`.



## `uint32_t CUpti_ActivitySourceLocator::lineNumber`

The line number in the source .

## 3.33. `CUpti_ActivityUnifiedMemoryCounter` Struct Reference

The activity record for Unified Memory counters.

This activity record represents a Unified Memory counter (`CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER`).

### `CUpti_ActivityUnifiedMemoryCounterKind` `CUpti_ActivityUnifiedMemoryCounter::counterKind`

The Unified Memory counter kind. See /ref `CUpti_ActivityUnifiedMemoryCounterKind`

### `uint32_t CUpti_ActivityUnifiedMemoryCounter::deviceId`

The ID of the device involved in the memory transfer operation. It is not relevant if the scope of the counter is global (all devices).

### `CUpti_ActivityKind` `CUpti_ActivityUnifiedMemoryCounter::kind`

The activity record kind, must be `CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER`

### `uint32_t CUpti_ActivityUnifiedMemoryCounter::pad`

Undefined. Reserved for internal use.

### `uint32_t` `CUpti_ActivityUnifiedMemoryCounter::processId`

The ID of the process to which this record belongs to. In case of global scope, `processId` is undefined.

## CUpti\_ActivityUnifiedMemoryCounterScope CUpti\_ActivityUnifiedMemoryCounter::scope

Scope of the Unified Memory counter. See /ref  
CUpti\_ActivityUnifiedMemoryCounterScope

## uint64\_t

### CUpti\_ActivityUnifiedMemoryCounter::timestamp

The timestamp when this sample was retrieved, in ns. A value of 0 indicates that timestamp information could not be collected

## uint64\_t CUpti\_ActivityUnifiedMemoryCounter::value

Value of the counter

## 3.34. CUpti\_ActivityUnifiedMemoryCounterConfig Struct Reference

Unified Memory counters configuration structure.

This structure controls the enable/disable of the various Unified Memory counters consisting of scope, kind and other parameters. See function /ref  
cuptiActivityConfigureUnifiedMemoryCounter

## uint32\_t

### CUpti\_ActivityUnifiedMemoryCounterConfig::deviceId

Device id of the target device. This is relevant only for single device scopes.

## uint32\_t

### CUpti\_ActivityUnifiedMemoryCounterConfig::enable

Control to enable/disable the counter. To enable the counter set it to non-zero value while disable is indicated by zero.

## CUpti\_ActivityUnifiedMemoryCounterKind

### CUpti\_ActivityUnifiedMemoryCounterConfig::kind

Unified Memory counter Counter kind

## CUpti\_ActivityUnifiedMemoryCounterScope CUpti\_ActivityUnifiedMemoryCounterConfig::scope

Unified Memory counter Counter scope

### 3.35. CUpti\_CallbackData Struct Reference

Data passed into a runtime or driver API callback function.

Data passed into a runtime or driver API callback function as the `cbdata` argument to `CUpti_CallbackFunc`. The `cbdata` will be this type for domain equal to `CUPTI_CB_DOMAIN_DRIVER_API` or `CUPTI_CB_DOMAIN_RUNTIME_API`. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data. For example, if you make a shallow copy of `CUpti_CallbackData` within a callback, you cannot dereference `functionParams` outside of that callback to access the function parameters. `functionName` is an exception: the string pointed to by `functionName` is a global constant and so may be accessed outside of the callback.

#### CUpti\_ApiCallbackSite CUpti\_CallbackData::callbackSite

Point in the runtime or driver function from where the callback was issued.

#### CUcontext CUpti\_CallbackData::context

Driver context current to the thread, or null if no context is current. This value can change from the entry to exit callback of a runtime API function if the runtime initializes a context.

#### uint32\_t CUpti\_CallbackData::contextUid

Unique ID for the CUDA context associated with the thread. The UIDs are assigned sequentially as contexts are created and are unique within a process.

#### uint64\_t \*CUpti\_CallbackData::correlationData

Pointer to data shared between the entry and exit callbacks of a given runtime or driver API function invocation. This field can be used to pass 64-bit values from the entry callback to the corresponding exit callback.

#### uint32\_t CUpti\_CallbackData::correlationId

The activity record correlation ID for this callback. For a driver domain callback (i.e. domain `CUPTI_CB_DOMAIN_DRIVER_API`) this ID will equal the correlation ID in

the `CUpti_ActivityAPI` record corresponding to the CUDA driver function call. For a runtime domain callback (i.e. `domain CUPTI_CB_DOMAIN_RUNTIME_API`) this ID will equal the correlation ID in the `CUpti_ActivityAPI` record corresponding to the CUDA runtime function call. Within the callback, this ID can be recorded to correlate user data with the activity record. This field is new in 4.1.

### `const char *CUpti_CallbackData::functionName`

Name of the runtime or driver API function which issued the callback. This string is a global constant and so may be accessed outside of the callback.

### `const void *CUpti_CallbackData::functionParams`

Pointer to the arguments passed to the runtime or driver API call. See `generated_cuda_runtime_api_meta.h` and `generated_cuda_meta.h` for structure definitions for the parameters for each runtime and driver API function.

### `void *CUpti_CallbackData::functionReturnValue`

Pointer to the return value of the runtime or driver API call. This field is only valid within the `exit::CUPTI_API_EXIT` callback. For a runtime API `functionReturnValue` points to a `cudaError_t`. For a driver API `functionReturnValue` points to a `CUresult`.

### `const char *CUpti_CallbackData::symbolName`

Name of the symbol operated on by the runtime or driver API function which issued the callback. This entry is valid only for driver and runtime launch callbacks, where it returns the name of the kernel.

## 3.36. `CUpti_EventGroupSet` Struct Reference

A set of event groups.

A set of event groups. When returned by `cuptiEventGroupSetsCreate` and `cuptiMetricCreateEventGroupSets` a set indicates that event groups that can be enabled at the same time (i.e. all the events in the set can be collected simultaneously).

### `CUpti_EventGroup *CUpti_EventGroupSet::eventGroups`

An array of `numEventGroups` event groups.

### `uint32_t CUpti_EventGroupSet::numEventGroups`

The number of event groups in the set.

### 3.37. CUpti\_EventGroupSets Struct Reference

A set of event group sets.

A set of event group sets. When returned by `cuptiEventGroupSetsCreate` and `cuptiMetricCreateEventGroupSets` a `CUpti_EventGroupSets` indicates the number of passes required to collect all the events, and the event groups that should be collected during each pass.

#### `uint32_t CUpti_EventGroupSets::numSets`

Number of event group sets.

#### `CUpti_EventGroupSet *CUpti_EventGroupSets::sets`

An array of `numSets` event group sets.

### 3.38. CUpti\_MetricValue Union Reference

A metric value.

Metric values can be one of several different kinds. Corresponding to each kind is a member of the `CUpti_MetricValue` union. The metric value returned by `cuptiMetricGetValue` should be accessed using the appropriate member of that union based on its value kind.

### 3.39. CUpti\_ModuleResourceData Struct Reference

Module data passed into a resource callback function.

CUDA module data passed into a resource callback function as the `cbdata` argument to `CUpti_CallbackFunc`. The `cbdata` will be this type for `domain` equal to `CUPTI_CB_DOMAIN_RESOURCE`. The module data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

#### `size_t CUpti_ModuleResourceData::cubinSize`

The size of the cubin.

#### `uint32_t CUpti_ModuleResourceData::moduleId`

Identifier to associate with the CUDA module.

## `const char *CUpti_ModuleResourceData::pCubin`

Pointer to the associated cubin.

### 3.40. CUpti\_NvtxData Struct Reference

Data passed into a NVTX callback function.

Data passed into a NVTX callback function as the `cbdata` argument to `CUpti_CallbackFunc`. The `cbdata` will be this type for `domain` equal to `CUPTI_CB_DOMAIN_NVTX`. Unless otherwise notes, the callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

## `const char *CUpti_NvtxData::functionName`

Name of the NVTX API function which issued the callback. This string is a global constant and so may be accessed outside of the callback.

## `const void *CUpti_NvtxData::functionParams`

Pointer to the arguments passed to the NVTX API call. See `generated_nvtx_meta.h` for structure definitions for the parameters for each NVTX API function.

### 3.41. CUpti\_ResourceData Struct Reference

Data passed into a resource callback function.

Data passed into a resource callback function as the `cbdata` argument to `CUpti_CallbackFunc`. The `cbdata` will be this type for `domain` equal to `CUPTI_CB_DOMAIN_RESOURCE`. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

## `CUcontext CUpti_ResourceData::context`

For `CUPTI_CBID_RESOURCE_CONTEXT_CREATED` and `CUPTI_CBID_RESOURCE_CONTEXT_DESTROY_STARTING`, the context being created or destroyed. For `CUPTI_CBID_RESOURCE_STREAM_CREATED` and `CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING`, the context containing the stream being created or destroyed.

## `void *CUpti_ResourceData::resourceDescriptor`

Reserved for future use.

## `CUstream CUpti_ResourceData::stream`

For `CUPTI_CBID_RESOURCE_STREAM_CREATED` and `CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING`, the stream being created or destroyed.

## 3.42. `CUpti_SynchronizeData` Struct Reference

Data passed into a synchronize callback function.

Data passed into a synchronize callback function as the `cbdata` argument to `CUpti_CallbackFunc`. The `cbdata` will be this type for domain equal to `CUPTI_CB_DOMAIN_SYNCHRONIZE`. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

## `CUcontext CUpti_SynchronizeData::context`

The context of the stream being synchronized.

## `CUstream CUpti_SynchronizeData::stream`

The stream being synchronized.

# Chapter 4.

## DATA FIELDS

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

### A

#### **attribute**

[CUpti\\_ActivityDeviceAttribute](#)

### B

#### **blockX**

[CUpti\\_ActivityKernel](#)  
[CUpti\\_ActivityKernel2](#)  
[CUpti\\_ActivityPreemption](#)  
[CUpti\\_ActivityCdpKernel](#)

#### **blockY**

[CUpti\\_ActivityPreemption](#)  
[CUpti\\_ActivityKernel](#)  
[CUpti\\_ActivityKernel2](#)  
[CUpti\\_ActivityCdpKernel](#)

#### **blockZ**

[CUpti\\_ActivityCdpKernel](#)  
[CUpti\\_ActivityKernel](#)  
[CUpti\\_ActivityKernel2](#)  
[CUpti\\_ActivityPreemption](#)

#### **bytes**

[CUpti\\_ActivityMemset](#)  
[CUpti\\_ActivityMemcpy](#)  
[CUpti\\_ActivityMemcpy2](#)



**C****cacheConfigExecuted**

CUpti\_ActivityKernel

**cacheConfigRequested**

CUpti\_ActivityKernel

**callbackSite**

CUpti\_CallbackData

**category**

CUpti\_ActivityMarkerData

**cbid**

CUpti\_ActivityAPI

**clocksThrottleReasons**

CUpti\_ActivityEnvironment

**color**

CUpti\_ActivityMarkerData

**completed**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

**computeApiKind**

CUpti\_ActivityContext

**computeCapabilityMajor**

CUpti\_ActivityDevice

**computeCapabilityMinor**

CUpti\_ActivityDevice

**constantMemorySize**

CUpti\_ActivityDevice

**context**

CUpti\_SynchronizeData

CUpti\_CallbackData

CUpti\_ResourceData

**contextId**

CUpti\_ActivityMemcpy

CUpti\_ActivityMemcpy2

CUpti\_ActivityMemset

CUpti\_ActivityKernel

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

CUpti\_ActivityContext

CUpti\_ActivityFunction

CUpti\_ActivityModule

**contextUid**

CUpti\_CallbackData

**cooling**

CUpti\_ActivityEnvironment

**copyKind**

CUpti\_ActivityMemcpy

CUpti\_ActivityMemcpy2

**coreClockRate**

CUpti\_ActivityDevice

**correlationData**

CUpti\_CallbackData

**correlationId**

CUpti\_ActivityKernel

CUpti\_ActivityBranch2

CUpti\_ActivityBranch

CUpti\_ActivityCdpKernel

CUpti\_ActivityMetric

CUpti\_ActivityKernel2

CUpti\_ActivityGlobalAccess2

CUpti\_ActivityMemcpy

CUpti\_ActivityInstructionExecution

CUpti\_ActivityAPI

CUpti\_ActivityMetricInstance

CUpti\_ActivityEvent

CUpti\_ActivityMemset

CUpti\_ActivityMemcpy2

CUpti\_ActivityEventInstance

CUpti\_ActivitySharedAccess

CUpti\_CallbackData

CUpti\_ActivityGlobalAccess

**counterKind**

CUpti\_ActivityUnifiedMemoryCounter

**cubin**

CUpti\_ActivityModule

**cubinSize**

CUpti\_ModuleResourceData

CUpti\_ActivityModule

**D****dcs**

CUpti\_ActivityObjectKindId

**deviceId**

CUpti\_ActivityUnifiedMemoryCounterConfig

CUpti\_ActivityMemcpy2

CUpti\_ActivityCdpKernel

CUpti\_ActivityDeviceAttribute  
 CUpti\_ActivityMemset  
 CUpti\_ActivityContext  
 CUpti\_ActivityEnvironment  
 CUpti\_ActivityMemcpy  
 CUpti\_ActivityKernel  
 CUpti\_ActivityUnifiedMemoryCounter  
 CUpti\_ActivityKernel2  
**diverged**  
 CUpti\_ActivityBranch  
 CUpti\_ActivityBranch2  
**domain**  
 CUpti\_ActivityEventInstance  
 CUpti\_ActivityEvent  
**dstContextId**  
 CUpti\_ActivityMemcpy2  
**dstDeviceId**  
 CUpti\_ActivityMemcpy2  
**dstKind**  
 CUpti\_ActivityMemcpy2  
 CUpti\_ActivityMemcpy  
**dynamicSharedMemory**  
 CUpti\_ActivityKernel  
 CUpti\_ActivityCdpKernel  
 CUpti\_ActivityKernel2  
  
**E**  
**enable**  
 CUpti\_ActivityUnifiedMemoryCounterConfig  
**enabled**  
 CUpti\_ActivityAutoBoostState  
**end**  
 CUpti\_ActivityMemcpy2  
 CUpti\_ActivityCdpKernel  
 CUpti\_ActivityAPI  
 CUpti\_ActivityMemset  
 CUpti\_ActivityOverhead  
 CUpti\_ActivityMemcpy  
 CUpti\_ActivityKernel  
 CUpti\_ActivityKernel2  
**environmentKind**  
 CUpti\_ActivityEnvironment

**eventGroups**

CUpti\_EventGroupSet

**executed**

CUpti\_ActivityInstructionExecution

CUpti\_ActivitySharedAccess

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

CUpti\_ActivityGlobalAccess

CUpti\_ActivityBranch2

CUpti\_ActivityBranch

CUpti\_ActivityGlobalAccess2

**F****fanSpeed**

CUpti\_ActivityEnvironment

**fileName**

CUpti\_ActivitySourceLocator

**flags**

CUpti\_ActivityMemcpy2

CUpti\_ActivityGlobalAccess2

CUpti\_ActivityDevice

CUpti\_ActivityMetric

CUpti\_ActivityDeviceAttribute

CUpti\_ActivityMarker

CUpti\_ActivityMemcpy

CUpti\_ActivityMetricInstance

CUpti\_ActivityMarkerData

CUpti\_ActivitySharedAccess

CUpti\_ActivityInstructionExecution

CUpti\_ActivityGlobalAccess

**functionId**

CUpti\_ActivityBranch2

CUpti\_ActivityInstructionExecution

CUpti\_ActivitySharedAccess

CUpti\_ActivityGlobalAccess2

**functionIndex**

CUpti\_ActivityFunction

**functionName**

CUpti\_NvtxData

CUpti\_CallbackData

**functionParams**

CUpti\_CallbackData

CUpti\_NvtxData

**functionReturnValue**

CUpti\_CallbackData

**G****globalMemoryBandwidth**

CUpti\_ActivityDevice

**globalMemorySize**

CUpti\_ActivityDevice

**gpuTemperature**

CUpti\_ActivityEnvironment

**gridId**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

CUpti\_ActivityPreemption

**gridX**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

CUpti\_ActivityKernel

**gridY**

CUpti\_ActivityKernel2

CUpti\_ActivityKernel

CUpti\_ActivityCdpKernel

**gridZ**

CUpti\_ActivityKernel

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

**I****id**

CUpti\_ActivityEvent

CUpti\_ActivityEventInstance

CUpti\_ActivityMetricInstance

CUpti\_ActivityModule

CUpti\_ActivityMarkerData

CUpti\_ActivityFunction

CUpti\_ActivityMarker

CUpti\_ActivityDevice

CUpti\_ActivitySourceLocator

CUpti\_ActivityMetric

**instance**

CUpti\_ActivityEventInstance

CUpti\_ActivityMetricInstance

**K****kind**

CUpti\_ActivityUnifiedMemoryCounterConfig  
 CUpti\_ActivitySharedAccess  
 CUpti\_ActivityModule  
 CUpti\_ActivityFunction  
 CUpti\_ActivityUnifiedMemoryCounter  
 CUpti\_ActivityInstructionExecution  
 CUpti\_ActivityEnvironment  
 CUpti\_ActivityOverhead  
 CUpti\_ActivityMarkerData  
 CUpti\_ActivityMarker  
 CUpti\_ActivityName  
 CUpti\_ActivityContext  
 CUpti\_ActivityDeviceAttribute  
 CUpti\_ActivityDevice  
 CUpti\_ActivityBranch2  
 CUpti\_ActivityBranch  
 CUpti\_ActivityGlobalAccess2  
 CUpti\_ActivityGlobalAccess  
 CUpti\_ActivitySourceLocator  
 CUpti\_ActivityMetricInstance  
 CUpti\_ActivityMetric  
 CUpti\_ActivityEventInstance  
 CUpti\_ActivityEvent  
 CUpti\_ActivityAPI  
 CUpti\_ActivityPreemption  
 CUpti\_ActivityCdpKernel  
 CUpti\_ActivityKernel2  
 CUpti\_ActivityKernel  
 CUpti\_ActivityMemset  
 CUpti\_ActivityMemcpy2  
 CUpti\_ActivityMemcpy  
 CUpti\_Activity

**L****l2\_transactions**

CUpti\_ActivityGlobalAccess  
 CUpti\_ActivityGlobalAccess2

**l2CacheSize**

CUpti\_ActivityDevice

**lineNumber**

CUpti\_ActivitySourceLocator

**localMemoryPerThread**

CUpti\_ActivityCdpKernel  
 CUpti\_ActivityKernel  
 CUpti\_ActivityKernel2

**localMemoryTotal**

CUpti\_ActivityCdpKernel  
 CUpti\_ActivityKernel  
 CUpti\_ActivityKernel2

**M****maxBlockDimX**

CUpti\_ActivityDevice

**maxBlockDimY**

CUpti\_ActivityDevice

**maxBlockDimZ**

CUpti\_ActivityDevice

**maxBlocksPerMultiprocessor**

CUpti\_ActivityDevice

**maxGridDimX**

CUpti\_ActivityDevice

**maxGridDimY**

CUpti\_ActivityDevice

**maxGridDimZ**

CUpti\_ActivityDevice

**maxIPC**

CUpti\_ActivityDevice

**maxRegistersPerBlock**

CUpti\_ActivityDevice

**maxSharedMemoryPerBlock**

CUpti\_ActivityDevice

**maxThreadsPerBlock**

CUpti\_ActivityDevice

**maxWarpsPerMultiprocessor**

CUpti\_ActivityDevice

**memoryClock**

CUpti\_ActivityEnvironment

**moduleId**

CUpti\_ModuleResourceData  
 CUpti\_ActivityFunction

**N****name**

CUpti\_ActivityKernel

CUpti\_ActivityKernel2  
 CUpti\_ActivityDevice  
 CUpti\_ActivityName  
 CUpti\_ActivityCdpKernel  
 CUpti\_ActivityMarker  
 CUpti\_ActivityFunction  
**notPredOffThreadsExecuted**  
 CUpti\_ActivityInstructionExecution  
**nullStreamId**  
 CUpti\_ActivityContext  
**numEventGroups**  
 CUpti\_EventGroupSet  
**numMemcpyEngines**  
 CUpti\_ActivityDevice  
**numMultiprocessors**  
 CUpti\_ActivityDevice  
**numSets**  
 CUpti\_EventGroupSets  
**numThreadsPerWarp**  
 CUpti\_ActivityDevice

**O**

**objectId**  
 CUpti\_ActivityName  
 CUpti\_ActivityMarker  
 CUpti\_ActivityOverhead

**objectKind**  
 CUpti\_ActivityMarker  
 CUpti\_ActivityName  
 CUpti\_ActivityOverhead

**overheadKind**  
 CUpti\_ActivityOverhead

**P**

**pad**  
 CUpti\_ActivityMemcpy2  
 CUpti\_ActivityKernel  
 CUpti\_ActivityEventInstance  
 CUpti\_ActivityBranch2  
 CUpti\_ActivityInstructionExecution  
 CUpti\_ActivityMetric  
 CUpti\_ActivityUnifiedMemoryCounter  
 CUpti\_ActivityModule



- CUpti\_ActivityPreemption
- CUpti\_ActivityMetricInstance
- CUpti\_ActivitySharedAccess
- CUpti\_ActivityGlobalAccess2
- parentBlockX**
  - CUpti\_ActivityCdpKernel
- parentBlockY**
  - CUpti\_ActivityCdpKernel
- parentBlockZ**
  - CUpti\_ActivityCdpKernel
- parentGridId**
  - CUpti\_ActivityCdpKernel
- payload**
  - CUpti\_ActivityMarkerData
- payloadKind**
  - CUpti\_ActivityMarkerData
- pcieLinkGen**
  - CUpti\_ActivityEnvironment
- pcieLinkWidth**
  - CUpti\_ActivityEnvironment
- pcOffset**
  - CUpti\_ActivityGlobalAccess
  - CUpti\_ActivityGlobalAccess2
  - CUpti\_ActivityInstructionExecution
  - CUpti\_ActivityBranch2
  - CUpti\_ActivitySharedAccess
  - CUpti\_ActivityBranch
- pCubin**
  - CUpti\_ModuleResourceData
- pid**
  - CUpti\_ActivityAutoBoostState
- power**
  - CUpti\_ActivityEnvironment
- powerLimit**
  - CUpti\_ActivityEnvironment
- preemptionKind**
  - CUpti\_ActivityPreemption
- processId**
  - CUpti\_ActivityUnifiedMemoryCounter
  - CUpti\_ActivityAPI
- pt**
  - CUpti\_ActivityObjectKindId

**Q****queued**

CUpti\_ActivityCdpKernel

**R****registersPerThread**

CUpti\_ActivityKernel

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

**requested**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

**reserved0**

CUpti\_ActivityMemset

CUpti\_ActivityKernel

CUpti\_ActivityKernel2

CUpti\_ActivityMemcpy

CUpti\_ActivityMemcpy2

**resourceDescriptor**

CUpti\_ResourceData

**returnValue**

CUpti\_ActivityAPI

**runtimeCorrelationId**

CUpti\_ActivityMemset

CUpti\_ActivityMemcpy

CUpti\_ActivityKernel

**S****scope**

CUpti\_ActivityUnifiedMemoryCounterConfig

CUpti\_ActivityUnifiedMemoryCounter

**sets**

CUpti\_EventGroupSets

**sharedMemoryConfig**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

**sharedTransactions**

CUpti\_ActivitySharedAccess

**smClock**

CUpti\_ActivityEnvironment

**sourceLocatorId**

CUpti\_ActivityGlobalAccess

CUpti\_ActivityGlobalAccess2

CUpti\_ActivityBranch  
 CUpti\_ActivityBranch2  
 CUpti\_ActivityInstructionExecution  
 CUpti\_ActivitySharedAccess

**speed**

CUpti\_ActivityEnvironment

**srcContextId**

CUpti\_ActivityMemcpy2

**srcDeviceId**

CUpti\_ActivityMemcpy2

**srcKind**

CUpti\_ActivityMemcpy

CUpti\_ActivityMemcpy2

**start**

CUpti\_ActivityKernel2

CUpti\_ActivityMemcpy

CUpti\_ActivityMemcpy2

CUpti\_ActivityMemset

CUpti\_ActivityKernel

CUpti\_ActivityCdpKernel

CUpti\_ActivityAPI

CUpti\_ActivityOverhead

**staticSharedMemory**

CUpti\_ActivityKernel2

CUpti\_ActivityCdpKernel

CUpti\_ActivityKernel

**stream**

CUpti\_ResourceData

CUpti\_SynchronizeData

**streamId**

CUpti\_ActivityKernel2

CUpti\_ActivityMemcpy2

CUpti\_ActivityMemset

CUpti\_ActivityMemcpy

CUpti\_ActivityCdpKernel

CUpti\_ActivityKernel

**submitted**

CUpti\_ActivityCdpKernel

**symbolName**

CUpti\_CallbackData

**T****temperature**

CUpti\_ActivityEnvironment

**theoreticalL2Transactions**

CUpti\_ActivityGlobalAccess2

**theoreticalSharedTransactions**

CUpti\_ActivitySharedAccess

**threadId**

CUpti\_ActivityAPI

**threadsExecuted**

CUpti\_ActivityBranch2

CUpti\_ActivitySharedAccess

CUpti\_ActivityInstructionExecution

CUpti\_ActivityGlobalAccess

CUpti\_ActivityBranch

CUpti\_ActivityGlobalAccess2

**timestamp**

CUpti\_ActivityMarker

CUpti\_ActivityPreemption

CUpti\_ActivityEnvironment

CUpti\_ActivityUnifiedMemoryCounter

**V****value**

CUpti\_ActivityMemset

CUpti\_ActivityUnifiedMemoryCounter

CUpti\_ActivityDeviceAttribute

CUpti\_ActivityMetricInstance

CUpti\_ActivityMetric

CUpti\_ActivityEventInstance

CUpti\_ActivityEvent

## **Notice**

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

## **Trademarks**

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## **Copyright**

© 2007-2014 NVIDIA Corporation. All rights reserved.