Protocol

MTLDevice

The main Metal interface to a GPU that apps use to draw graphics and run computations in parallel.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 13.1+ | macOS 10.11+ | tvOS | visionOS 1.0+

```
protocol MTLDevice : NSObjectProtocol, Sendable
```

Mentioned in

- Understanding the Metal 4 core API
- Finding multiple GPUs on an Intel-based Mac
- Confirming which counters and counter sets a GPU supports
- Converting GPU timestamps into CPU time
- Creating an indirect command buffer

Overview

You can get the default `MTLDevice` at runtime by calling `MTLCreateSystemDefaultDevice()` (see Getting the default GPU). Each Metal device instance represents a GPU and is the main starting point for your app's interaction with it. With a Metal device instance, you can inspect a GPU's features and capabilities (see Device inspection) and create subsidiary type instances with its factory methods.

- Buffers, textures, and other resources store, synchronize, and pass data between the GPU and CPU (see Resource fundamentals).
- Input/Output command queues efficiently load resources from the file system (see Resource loading).

- Command queues create command encoders and schedule work for the GPU, including rendering and compute commands (see Render passes and Compute passes).
- Pipeline states store render or compute pipeline configurations — which can be expensive to create — so that you can reuse them, potentially many times.

If your app uses more than one GPU (see Multi-GPU systems), ensure that instances of these types only interact with others from the same device. For example, your app can pass a texture to a command encoder that comes from the same Metal device, but not to another device.

Topics

Working with GPU devices

Device inspection

Locate and identify a GPU and the features it supports, and sample its counters.

Work submission

Create queues that submit work to the GPU or load assets into GPU resources, and indirect command buffers that group your frequent commands together.

Pipeline state creation

Create pipeline states for render and compute passes, samplers, depth and stencil states, and indirect command buffers.

Resource creation

Load assets with input/output queues and make various resource instances, such as buffers, textures, acceleration structures, and memory heaps.

Shader library and archive creation

Create static and dynamic shader libraries, and binary shader archives.

Instance Properties

`var maximumConcurrentCompilationTaskCount: Int`

The maximum number of concurrent compilation tasks the device is running.

Required

`var shouldMaximizeConcurrentCompilation: Bool`

A Boolean value that indicates whether the device uses additional CPU threads for compilation tasks.

Required

Instance Methods

```
func functionHandle(function: any MTLFunction) -> (any MTLFunction
Handle)?
```

Required

```
func functionHandle(function: any MTL4BinaryFunction) -> (any
MTLFunctionHandle)?
```

Get the function handle for the specified binary-linked function from the pipeline state.

Required

```
func makeArchive(url: URL) throws -> any MTL4Archive
```

Creates a new archive from data available at an NSURL address.

Required

```
func makeArgumentTable(descriptor: MTL4ArgumentTableDescriptor) throws
-> any MTL4ArgumentTable
```

Creates a new argument table from an argument table descriptor.

Required

```
func makeBuffer(length: Int, options: MTLResourceOptions, placement
SparsePageSize: MTLSparsePageSize) -> (any MTLBuffer)?
```

Creates a new placement sparse buffer of a specific length.

Required

```
func makeCommandAllocator() -> (any MTL4CommandAllocator)?
```

Creates a new command allocator.

Required

```
func makeCommandAllocator(descriptor: MTL4CommandAllocatorDescriptor)
throws -> any MTL4CommandAllocator
```

Creates a new command allocator from a command allocator descriptor.

Required

```
func makeCommandBuffer() -> (any MTL4CommandBuffer)?
```

Creates a new command buffer.

Required

```
func makeCommandQueue(descriptor: MTLCommandQueueDescriptor) -> (any
MTLCommandQueue)?
```

Creates a command queue with the provided configuration.

Required

```
func makeCompiler(descriptor: MTL4CompilerDescriptor) throws -> any MTL4Compiler
```

Creates a new compiler from a compiler descriptor.

Required

```
func makeCounterHeap(descriptor: MTL4CounterHeapDescriptor) throws -> any MTL4CounterHeap
```

Creates a new counter heap configured from a counter heap descriptor.

Required

```
func makeLogState(descriptor: MTLLogStateDescriptor) throws -> any MTLLogState
```

Creates a shader log state with the provided configuration.

Required

```
func makeMTL4CommandQueue() -> (any MTL4CommandQueue)?
```

Creates a new command queue.

Required

```
func makeMTL4CommandQueue(descriptor: MTL4CommandQueueDescriptor) throws -> any MTL4CommandQueue
```

Creates a new command queue from a queue descriptor.

Required

```
func makePipelineDataSetSerializer(descriptor: MTL4PipelineDataSetSerializerDescriptor) -> any MTL4PipelineDataSetSerializer
```

Creates a new pipeline data set serializer instance from a descriptor.

Required

```
func makeTensor(descriptor: MTLTensorDescriptor) throws -> any MTLTensor
```

Creates a tensor by allocating new memory.

Required

```
func makeTextureViewPool(descriptor: MTLResourceViewPoolDescriptor) throws -> any MTLTextureViewPool
```

Creates a new texture view pool from a resource view pool descriptor.

Required

```
func queryTimestampFrequency() -> UInt64
```

Queries the frequency of the GPU timestamp in ticks per second.

Required

`func size(ofCounterHeapEntry: MTL4CounterHeapType) -> Int`

Returns the size, in bytes, of each entry in a counter heap of a specific counter heap type when your app resolves it into a usable format.

Required

`func tensorSizeAndAlign(descriptor: MTLTensorDescriptor) -> MTLSizeAndAlign`

Determines the size and alignment required to hold the data of a tensor you create with a descriptor in a buffer.

Required

Relationships

Inherits From

`NSObjectProtocol`, `Sendable`, `SendableMetatype`

See Also

Locating and inspecting a GPU device

Getting the default GPU

Select the system's default GPU device on which to run your Metal code.

Detecting GPU features and Metal software versions

Use the device object's properties to determine how you perform tasks in Metal.

`func MTLCreateSystemDefaultDevice() -> (any MTLDevice)?`

Returns the device instance Metal selects as the default.

Multi-GPU systems

Locate and work with internal and external GPUs and their displays, video memory, and performance tradeoffs.