

[Metal](#) / MTLComputeCommandEncoder

Protocol

MTLComputeCommandEncoder

An interface for dispatching commands to encode in a compute pass.

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

protocol MTLComputeCommandEncoder : MTLCommandEncoder

Mentioned in

- 📄 [Understanding the Metal 4 core API](#)
- 📄 [Improving CPU performance by using argument buffers](#)
- 📄 [Sampling GPU data into counter sample buffers](#)
- 📄 [Setting up a command structure](#)
- 📄 [Simplifying GPU resource management with residency sets](#)

Overview

You create compute command encoders by calling the `makeComputeCommandEncoder(dispatchType:)` method of the `MTLCommandBuffer` instance you're using to encode your compute pass. You can encode multiple commands to execute as part of a single pass of the encoder.

To encode kernel function calls:

1. Configure an `MTLComputePipelineState` instance with a kernel, using a method such as `makeComputePipelineState(function:)`. See [Creating Compute Pipeline States for all MTLDevice](#) methods that create a new pipeline state for your command encoder.

2. Set the pipeline state with the `setComputePipelineState(_:)` method on your command encoder.
3. Provide parameters for your compute kernel by binding information to kernel arguments. Examples of methods that bind data for access on the GPU are `setBuffer(_:offset:index:)` and `setTexture(_:index:)`.
4. Encode compute commands that call your kernel by either Dispatching Kernel Calls Directly or Dispatching from Indirect Command Buffers.
5. Call `endEncoding()` to finish encoding the kernel call of the compute pass.

Important

Call `endEncoding()` on any existing compute command encoder before releasing it or creating one.

After adding all commands to your compute command encoder, use the `commit()` method to submit work to the GPU.

Topics

Configuring the pipeline state

Configure a compute pipeline state to describe the runtime environment of an encoder.

```
func setComputePipelineState(any MTLComputePipelineState)
```

Configures the compute encoder with a pipeline state for subsequent kernel calls.

Required

```
var dispatchType: MTLDispatchType
```

The dispatch type to use when submitting compute work to the GPU.

Required

Encoding buffers

Encode buffers to provide their data on the GPU as kernel arguments.

```
func setBuffer((any MTLBuffer)?, offset: Int, index: Int)
```

Binds a buffer to the buffer argument table, allowing compute kernels to access its data on the GPU.

Required

```
func setBuffer(any MTLBuffer, offset: Int, attributeStride: Int, index: Int)
```

Binds a buffer with a stride to the buffer argument table, allowing compute kernels to access its data on the GPU.

Required

```
func setBuffers([(any MTLBuffer)?], offsets: [Int], range: Range<Int>)
```

Binds multiple buffers to the buffer argument table at once, allowing compute kernels to access their data on the GPU.

```
func setBuffers([(any MTLBuffer)?], offsets: [Int], attributeStrides: [Int], range: Range<Int>)
```

Binds multiple buffers with data in stride to the buffer argument table at once, allowing compute kernels to access their data on the GPU.

```
func setBufferOffset(Int, index: Int)
```

Changes where the data begins in a buffer already bound to the buffer argument table.

Required

```
func setBufferOffset(offset: Int, attributeStride: Int, index: Int)
```

Changes where the data begins and the distance between adjacent elements in a buffer already bound to the buffer argument table.

Required

Encoding raw bytes

Encode bytes directly from the CPU as a kernel argument, without creating an intermediate buffer.

```
func setBytes(UnsafeRawPointer, length: Int, index: Int)
```

Copies data directly to the GPU to populate an entry in the buffer argument table.

Required

```
func setBytes(UnsafeRawPointer, length: Int, attributeStride: Int, index: Int)
```

Copies data with a given stride directly to the GPU to populate an entry in the buffer argument table.

Required

Encoding textures

Encode textures to provide access on the GPU as kernel arguments.

```
func setTexture((any MTLTexture)?, index: Int)
```

Binds a texture to the texture argument table, allowing compute kernels to access its data on the GPU.

Required

```
func setTextures([(any MTLTexture)?], range: Range<Int>)
```

Binds multiple textures to the texture argument table, allowing compute functions to access their data on the GPU.

Encoding texture sampler states

Encode texture samplers to provide access on the GPU as kernel arguments.

```
func setSamplerState((any MTLSamplerState)?, index: Int)
```

Encodes a texture sampler, allowing compute kernels to use it for sampling textures on the GPU.

Required

```
func setSamplerState((any MTLSamplerState)?, lodMinClamp: Float, lodMaxClamp: Float, index: Int)
```

Encodes a texture sampler with a custom level of detail clamping, allowing compute kernels to use it for sampling textures on the GPU.

Required

```
func setSamplerStates([(any MTLSamplerState)?], range: Range<Int>)
```

Encodes multiple texture samplers to the sampler argument table, allowing compute kernels to use them for sampling textures on the GPU.

```
func setSamplerStates([(any MTLSamplerState)?], lodMinClamps: [Float], lodMaxClamps: [Float], range: Range<Int>)
```

Encodes multiple texture samplers for the compute function, specifying clamp values for the level of detail of each sampler.

Encoding function tables

Encode function information for use by a compute kernel, providing access to function pointers.

```
func setVisibleFunctionTable((any MTLVisibleFunctionTable)?, bufferIndex: Int)
```

Binds a visible function table to the buffer argument table, allowing you to call its functions on the GPU.

Required

```
func setVisibleFunctionTables([(any MTLVisibleFunctionTable)?], buffer
Range: Range<Int>)
```

Binds multiple visible function tables to the buffer argument table, allowing you to call their functions on the GPU.

```
func setIntersectionFunctionTables([(any MTLIntersectionFunctionTable
)?], bufferRange: Range<Int>)
```

Binds multiple intersection function tables to the buffer argument table, allowing you to call their functions on the GPU.

Encoding acceleration structures

Access acceleration structure instances in an intersection function.

```
func setAccelerationStructure((any MTLAccelerationStructure)?, buffer
Index: Int)
```

Binds an acceleration structure to the buffer argument table, allowing functions to access it on the GPU.

Required

```
func setIntersectionFunctionTable((any MTLIntersectionFunctionTable)?,
bufferIndex: Int)
```

Binds an intersection function table to the buffer argument table, making it callable in your Metal shaders.

Required

Encoding resident resources

Access resources that the CPU allocates on the GPU during your compute pass without a copy of the data.

```
func useResource(any MTLResource, usage: MTLResourceUsage)
```

Ensures kernel calls that the system encodes in subsequent commands have access to a resource.

Required

```
func useResources([any MTLResource], usage: MTLResourceUsage)
```

Ensures kernel calls that the system encodes in subsequent commands have access to multiple resources.

```
func useHeap(any MTLHeap)
```

Ensures the shaders in the render pass's subsequent draw commands have access to all of the resources you allocate from a heap.

Required

```
func useHeaps([any MTLHeap])
```

Ensures the shaders in the render pass's subsequent draw commands have access to all of the resources you allocate from multiple heaps.

Encoding tile memory usage

Reserve space in GPU tile memory for threadgroups and imageblocks.

```
func setThreadgroupMemoryLength(Int, index: Int)
```

Configures the size of a block of threadgroup memory.

Required

```
func setImageblockWidth(Int, height: Int)
```

Sets the size, in pixels, of imageblock data in tile memory.

Required

Encoding stage-in data

Set data in the stage-in region of a compute kernel for processing per-thread inputs.

```
func setStageInRegion(MTLRegion)
```

Sets the dimensions over the thread grid of how your compute kernel receives stage-in arguments.

Required

```
func setStageInRegionWithIndirectBuffer(any MTLBuffer, indirectBuffer  
Offset: Int)
```

Sets the region of the stage-in attributes to apply to a compute kernel using an indirect buffer.

Required

Dispatching kernel calls directly

Encode kernel function calls to run as part of your compute pass.

```
func dispatchThreads(MTLSize, threadsPerThreadgroup: MTLSize)
```

Encodes a compute command using an arbitrarily sized grid.

Required

```
func dispatchThreadgroups(MTLSize, threadsPerThreadgroup: MTLSize)
```

Encodes a compute dispatch command using a grid aligned to threadgroup boundaries.

Required

Dispatching from indirect command buffers

Encode commands within an indirect command buffer to run as part of your compute pass.

```
func dispatchThreadgroups(indirectBuffer: any MTLBuffer, indirectBuffer  
Offset: Int, threadsPerThreadgroup: MTLSize)
```

Encodes a dispatch call for a compute pass, using an indirect buffer that defines the size of a grid that aligns to threadgroup boundaries.

Required

```
func executeCommandsInBuffer(any MTLIndirectCommandBuffer, range: Range  
<Int>)
```

Encodes an instruction to run commands from an indirect buffer.

```
func executeCommandsInBuffer(any MTLIndirectCommandBuffer, indirect  
Buffer: any MTLBuffer, offset: Int)
```

Encodes an instruction to run commands from an indirect buffer, using another buffer to provide the command range.

```
func executeCommands(in: any MTLIndirectCommandBuffer, indirectBuffer:  
any MTLBuffer, indirectBufferOffset: Int)
```

Encodes an instruction to run commands from an indirect buffer, using another buffer to provide the command range.

```
func executeCommands(in: any MTLIndirectCommandBuffer, with: NSRange)
```

Encodes an instruction to run commands from an indirect buffer.

Synchronizing across command execution

Protect against hazards for untracked resources, using memory fences and barriers.

```
func waitForFence(any MTLFence)
```

Encodes a command that instructs the GPU to pause pass execution until a fence updates.

Required

```
func updateFence(any MTLFence)
```

Encodes a command that instructs the GPU to update a fence, allowing passes waiting on the fence to start or resume.

Required

```
func memoryBarrier(scope: MTLBarrierScope)
```

Creates a memory barrier that enforces the order of write and read operations for specific resource types.

Required

```
func memoryBarrier(resources: [any MTLResource])
```

Creates a memory barrier that enforces the order of write and read operations for specific resources.

Encoding sample counters

Sample real-time data on execution from the GPU's hardware as it runs your compute pass.

```
func sampleCounters(sampleBuffer: any MTLCounterSampleBuffer, sample  
Index: Int, barrier: Bool)
```

Encodes a command to sample hardware counters, providing performance information.

Required

Relationships

Inherits From

MTLCommandEncoder, NSObjectProtocol

See Also

Encoding a compute pass

 [Creating threads and threadgroups](#)

Learn how Metal organizes compute-processing workloads.

 [Calculating threadgroup and grid sizes](#)

Calculate the optimum sizes for threadgroups and grids when dispatching compute-processing workloads.

`protocol MTL4ComputeCommandEncoder`

Encodes a compute pass and other memory operations into a command buffer.