Protocol

MTLCommandEncoder

An encoder that writes GPU commands into a command buffer.

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

```
protocol MTLCommandEncoder : NSObjectProtocol
```

Mentioned in

- Setting up a command structure
- Understanding the Metal 4 core API

Overview

Don't implement this protocol yourself; instead you call methods on an `MTLCommandBuffer` instance to create command encoders. Command encoder instances are lightweight instances that you re-create every time you need to send commands to the GPU.

There are many different kinds of command encoders, each providing a different set of commands that can be encoded into the buffer. A command encoder implements the `MTLCommandEncoder` protocol and an additional protocol specific to the kind of encoder being created.

Protocol	Task
`MTLRenderCommandEncoder`	Graphics rendering
`MTLComputeCommandEncoder`	Computation
`MTLBlitCommandEncoder`	Memory management

Protocol	Task
MTLParallelRenderCommandEncoder	Multiple graphics rendering tasks encoded in parallel.

While a command encoder is active, it has the exclusive right to append commands to its command buffer. Once you finish encoding commands, call the `endEncoding()` method to finish encoding the commands. To write further commands into the same command buffer, create a new command encoder.

You can call the `insertDebugSignpost(_:)`, `pushDebugGroup(_:)`, and `popDebugGroup()` methods to put debug strings into the command buffer and to push or pop string labels used to identify groups of encoded commands. These methods don't change the rendering or compute behavior of your app; the Xcode debugger uses them to organize your app's rendering commands in a format that may provide insight into how your app works.

Topics

Ending command encoding

`func endEncoding()`

Declares that all command generation from the encoder is completed.

Required

Annotating the command buffer with debug information

`func insertDebugSignpost(String)`

Inserts a debug string into the captured frame data.

Required

`func pushDebugGroup(String)`

Pushes a specific string onto a stack of debug group strings for the command encoder.

Required

`func popDebugGroup()`

Pops the latest string off of a stack of debug group strings for the command encoder.

Required

Identifying the command encoder

`var device: any MTLDevice`

The Metal device from which the command encoder was created.

Required

`var label: String?`

A string that labels the command encoder.

Required

Instance Methods

`func barrier(afterQueueStages: MTLStages, beforeStages: MTLStages)`

Encodes a consumer barrier on work you commit to the same command queue.

Required

Relationships

Inherits From

`NSObjectProtocol`

Inherited By

`MTLAccelerationStructureCommandEncoder`
`MTLBlitCommandEncoder`
`MTLComputeCommandEncoder`
`MTLParallelRenderCommandEncoder`
`MTLRenderCommandEncoder`
`MTLResourceStateCommandEncoder`

See Also

Submitting work to a GPU with Metal

Setting up a command structure

Discover how Metal executes commands on a GPU.

`protocol MTLCommandQueue`

An instance you use to create, submit, and schedule command buffers to a specific GPU device to run the commands within those buffers.

`class MTLCommandQueueDescriptor`

A configuration that customizes the behavior for a new command queue.

`protocol MTLCommandBuffer`

A container that stores a sequence of GPU commands that you encode into it.

`class MTLCommandBufferDescriptor`

A configuration that customizes the behavior for a new command buffer.

`struct MTLCommandBufferError`

The command buffer error codes that indicate why the GPU doesn’t finish executing a command buffer.