

[Metal / MTLCommandBuffer](#)

Protocol

# MTLCommandBuffer

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

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

```
protocol MTLCommandBuffer : NSObjectProtocol
```

## Mentioned in

- 📄 Understanding the Metal 4 core API
- 📄 Setting up a command structure
- 📄 Simplifying GPU resource management with residency sets
- 📄 Tracking the resource residency of argument buffers

## Overview

A command buffer represents a chunk of work for the GPU that stores the commands you encode to it, as well as any resources those commands need. You primarily use a command buffer to:

- Create command encoders and call their methods to add commands to the buffer
- Optionally reserve a place for the command buffer in its command queue by *enqueueing* the command buffer, even before you encode any commands into it
- Submit, or *commit*, the contents of the command buffer to the command queue that creates it to run on the GPU device the queue represents

Create a command encoder from an [MTLCommandQueue](#) instance by calling its [makeCommandBuffer\(\)](#) method. Typically, you create one or more command queues when your app launches and then keep them throughout your app's lifetime.

To add commands to an [MTLCommandBuffer](#) instance, create an encoder from one of its factory methods, including:

- An [MTLRenderCommandEncoder](#) instance by calling [makeRenderCommandEncoder\(descriptor:\)](#)
- An [MTLComputeCommandEncoder](#) instance by calling [makeComputeCommandEncoder\(dispatchType:\)](#)
- An [MTLBlitCommandEncoder](#) instance by calling [makeBlitCommandEncoder\(\)](#) or [makeBlitCommandEncoder\(descriptor:\)](#)
- An [MTLParallelRenderCommandEncoder](#) instance by calling [makeParallelRenderCommandEncoder\(descriptor:\)](#)

#### Note

All encoders inherit additional methods from the [MTLCommandEncoder](#).

You can use only a single encoder at a time to add commands to a command buffer. To start using a different command encoder, first signal that you're done with the current encoder by calling its [endEncoding\(\)](#) method. Then create another command encoder from the command buffer and continue adding commands to the buffer with the new encoder.

Repeat the process until you finish encoding commands to the command buffer and are ready to run the buffer's contents on the GPU. Then submit the command buffer to the command queue that you used to create it by calling the command buffer's [commit\(\)](#) method. After an app commits a command buffer, you check its [status](#) property or block a thread by calling its [waitUntilScheduled\(\)](#) or [waitForCompletion\(\)](#) methods.

You also have the option to reserve a place for the command buffer in its command queue by calling the command buffer's [enqueue\(\)](#) method. You can call this method exactly once at any time before you commit the buffer to the queue. If you don't enqueue a command buffer, it implicitly enqueues itself when you commit it. Each command queue ensures the order that you enqueue its command buffers is the same order the queue schedules them to run on the GPU.

#### Tip

Establish an order of execution for multiple command buffers you encode in parallel by first calling their [enqueue\(\)](#) methods in that order.

For example, a multithreaded app might set the GPU's execution order for a sequence of related subtasks by:

1. Creating a command buffer for each subtask
  2. Enqueuing the command buffers in the proper order on a single thread
  3. Encoding commands to each buffer on a separate thread and then committing it
- 

# Topics

## Creating command encoders

Create a command encoder from a command buffer that encodes a series of GPU commands to it.

### Command encoder factory methods

A command encoder defines the actions of a single pass, such as GPU commands that draw, compute, or quickly copy resource data.

## Attaching residency sets

```
func useResidencySet(any MTLResidencySet)
```

Applies a residency set to a command buffer.

**Required**

```
func useResidencySets([any MTLResidencySet])
```

Applies multiple residency sets to a command buffer.

## Synchronizing passes with events

Instruct the GPU to wait for an event between two passes until a different command queue's command buffer signals the event.

```
func encodeWaitForEvent(any MTLEvent, value: UInt64)
```

Encodes a command into the command buffer that pauses the GPU from running the buffer's subsequent passes until the event equals or exceeds a value.

**Required**

```
func encodeSignalEvent(any MTLEvent, value: UInt64)
```

Encodes a command that updates an event's value, which can clear the GPU to run passes from other command buffers waiting for the event.

**Required**

## Presenting a drawable

Instruct the command buffer to call a drawable's presentation method for you at the best time with convenience methods.

```
func present(any MTLDrawable)
```

Presents a drawable as early as possible.

**Required** Default implementation provided.

```
func present(any MTLDrawable, atTime: CFTimeInterval)
```

Presents a drawable at a specific time.

**Required**

```
func present(any MTLDrawable, afterMinimumDuration: CFTimeInterval)
```

Presents a drawable after the system presents the previous drawable for an amount of time.

**Required**

## Registering state change handlers

Notify your app before and after a command buffer runs on the GPU.

```
func addScheduledHandler(MTLCommandBufferHandler)
```

Registers a completion handler the GPU device calls immediately after it schedules the command buffer to run on the GPU.

**Required**

```
func addCompletedHandler(MTLCommandBufferHandler)
```

Registers a completion handler the GPU device calls immediately after the GPU finishes running the commands in the command buffer.

**Required**

```
typealias MTLCommandBufferHandler
```

A completion handler signature a GPU device calls when it finishes scheduling a command buffer, or when the GPU finishes running it.

## Submitting a command buffer

Send a command buffer to run on the GPU, or reserve a place in the queue to arrange its relative order with other command buffers.

```
func enqueue()
```

Reserves the next available place for the command buffer in its command queue.

**Required**

```
func commit()
```

Submits the command buffer to run on the GPU.

Required

## Waiting for state changes

Pause your app's execution on the CPU until the command buffer changes its state.

```
func waitUntilScheduled()
```

Blocks the current thread until the command queue schedules the buffer.

Required

```
func waitUntilCompleted()
```

Blocks the current thread until the GPU finishes executing the command buffer and all of its completion handlers.

Required

## Troubleshooting a command buffer

```
var status: MTLCommandBufferStatus
```

The command buffer's current state.

Required

```
enum MTLCommandBufferStatus
```

The discrete states for a command buffer that represent its life cycle stages.

≡ Command buffer debugging

Properties and methods for programmatically debugging runtime issues with a command buffer.

## Instance Methods

```
func completed() async
```

```
func scheduled() async
```

---

## Relationships

## Inherits From

NSObjectProtocol

---

## 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.

`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.

`protocol MTLCommandEncoder`

An encoder that writes GPU commands into a command buffer.