

[Metal / Shader libraries](#)

[API Collection](#)

Shader libraries

Manage and load your app's Metal shaders.

Overview

A Metal library represents a collection of one or more shaders. Xcode creates a library from the shader source files in a project, a Metal intermediate representation (IR) file, or a binary archive file. You can also create IR files from Metal source code by running the Metal compiler in a command-line environment.

Apps create the default library instance by calling a Metal device's [`makeDefaultLibrary\(\)`](#) method. The default library contains all the shaders from a project's shader source files, which Xcode compiles at build time. Apps create additional libraries by passing an IR file to an [`MTLDevice`](#) instance's [`makeLibrary\(URL:\)`](#) method or one of its sibling methods. The device can also create a library directly from source code by passing it as a string to the [`makeLibrary\(source:options:\)`](#) method. See [Shader library and archive creation](#) for more information.

You can apply a shader from a library to a pipeline state's entry point, such as the [`computeFunction`](#) property for a compute pass. Start by retrieving an [`MTLFunction`](#) instance from a library, which is a reference to the library's shader, by calling its [`makeFunction\(name:\)`](#) method or a sibling method. Then set the function instance to the appropriate property of a pipeline descriptor. For example, an app can retrieve a vertex stage's entry point shader from a library and assign it to the [`vertexFunction`](#) property of an [`MTLRenderPipelineDescriptor`](#).

Dynamic libraries are a collection of other shaders, typically utility functions, that support the entry point shaders for a pipeline state. To create a dynamic library, pass an [`MTLLibrary`](#) instance to a device's [`makeDynamicLibrary\(library:\)`](#) method, or pass a file URL to [`makeDynamicLibrary\(url:\)`](#). Add a dynamic library to a pipeline state by including it in an array of a pipeline descriptor's preloaded libraries property. For example, if a vertex shader calls a shader in a dynamic library, directly or indirectly, add that dynamic library to the [`vertexPreloaded`](#)

[Libraries](#) property's array. You can also build dynamic libraries with the Metal compiler in Terminal.

Binary archives are precompiled static libraries for specific GPU architectures that allow you to avoid the cost of runtime shader compilation. Because Metal automatically builds and caches shaders on the device running an app, use binary archives as part of your distributed app, or deliver them through content updates. See [Creating binary archives from device-built pipeline state objects](#) for more information on how to build and distribute binary archives for any device that supports Metal.

Topics

Shader compilation

Compile and manipulate Metal shader libraries from the command line.

☰ Metal libraries

Compile and manage Metal libraries from the command line.

☰ Metal dynamic libraries

Create a single Metal library containing reusable code to reduce library size and avoid repeated shader compilation at runtime.

☰ Metal binary archives

Distribute precompiled GPU-specific binaries as part of your app to avoid runtime compilation of Metal shaders.

`protocol MTL4Compiler`

A abstraction for a pipeline state and shader function compiler.

`class MTL4CompilerDescriptor`

Groups together properties for creating a compiler context.

`class MTL4CompilerTaskOptions`

The configuration options that control the behavior of a compilation task for a Metal 4 compiler instance.

`enum MTL4CompilerTaskStatus`

Represents the status of a compiler task.

`protocol MTL4Archive`

A read-only container that stores pipeline states from a shader compiler.

```
protocol MTL4BinaryFunction
```

Represents a binary function.

```
class MTL4BinaryFunctionDescriptor
```

Base interface for other function-derived interfaces.

```
struct MTL4BinaryFunctionOptions
```

Options for configuring the creation of binary functions.

```
class MTL4PipelineStageDynamicLinkingDescriptor
```

Groups together properties to drive the dynamic linking process of a pipeline stage.

Pipeline compilation

```
enum MTL4BlendState
```

Enumeration for controlling the blend state of a pipeline state object.

```
class MTL4FunctionDescriptor
```

Base interface for describing a Metal 4 shader function.

```
enum MTL4IndirectCommandBufferSupportState
```

Enumeration for controlling support for [MTLIndirectCommandBuffer](#).

```
class MTL4LibraryDescriptor
```

Serves as the base descriptor for creating a Metal library.

```
class MTL4LibraryFunctionDescriptor
```

Describes a shader function from a Metal library.

```
enum MTL4LogicalToPhysicalColorAttachmentMappingState
```

Enumerates possible behaviors of how a pipeline maps its logical outputs to its color attachments.

```
typealias MTL4NewBinaryFunctionCompletionHandler
```

Provides a signature for a callback block that Metal calls when the compiler finishes a build task for a binary function.

```
typealias MTL4NewMachineLearningPipelineStateCompletionHandler
```

Provides a signature for a callback block that Metal calls when the compiler finishes a build task for a machine learning pipeline state.

```
struct MTL4ShaderReflection
```

Option mask for requesting reflection information at pipeline build time.

`class MTL4SpecializedFunctionDescriptor`

Groups together properties to configure and create a specialized function by passing it to a factory method.

`enum MTL4AlphaToCoverageState`

Enumeration for controlling alpha-to-coverage state of a pipeline state object.

`enum MTL4AlphaToOneState`

Enumeration for controlling alpha-to-one state of a pipeline state object.

`class MTL4StaticLinkingDescriptor`

Groups together properties to drive a static linking process.

`class MTL4StitchedFunctionDescriptor`

Groups together properties that describe a shader function suitable for stitching.

`class MTLFunctionReflection`

Represents a reflection object containing information about a function in a Metal library.

`typealias MTLNewDynamicLibraryCompletionHandler`

Pipeline harvesting

`protocol MTL4PipelineDataSetSerializer`

A fast-addition container for collecting data during pipeline state creation.

`struct MTL4PipelineDataSetSerializerConfiguration`

Configuration options for pipeline dataset serializer objects.

`class MTL4PipelineDataSetSerializerDescriptor`

Groups together properties to create a pipeline data set serializer.

`class MTL4PipelineDescriptor`

Base type for descriptors you use for building pipeline state objects.

`class MTL4PipelineOptions`

Provides options controlling how to compile a pipeline state.

Shader library management

```
protocol MTLLibrary
```

A collection of Metal shader functions.

```
protocol MTLDynamicLibrary
```

A dynamically linkable representation of compiled shader code for a specific Metal device object.

```
protocol MTLBinaryArchive
```

A container for pipeline state descriptors and their associated compiled shader code.

```
class MTLCompileOptions
```

Compilation settings for a Metal shader library.

```
enum MTLLibraryType
```

A set of options for Metal library types.

```
enum MTLLanguageVersion
```

Metal shading language versions.

```
enum MTLCompileSymbolVisibility
```

```
enum MTLLibraryOptimizationLevel
```

The optimization options for the Metal compiler.

Shader functions

```
class MTLFunctionDescriptor
```

A description of a function object to create.

```
protocol MTLFunction
```

A interface that represents a public shader function in a Metal library.

```
protocol MTLFunctionHandle
```

An object representing a function that you can add to a visible function table.

```
class MTLVisibleFunctionTableDescriptor
```

A specification of how to create a visible function table.

```
protocol MTLVisibleFunctionTable
```

A table of shader functions visible to your app that you can pass into compute commands to customize the behavior of a shader.

```
class MTLIntersectionFunctionDescriptor
```

A description of an intersection function that performs an intersection test.

```
class MTLIntersectionFunctionTableDescriptor
```

A specification of how to create an intersection function table.

```
protocol MTLIntersectionFunctionTable
```

A table of intersection functions that Metal calls to perform ray-tracing intersection tests.

Stitched function libraries

{} Customizing shaders using function pointers and stitching

Define custom shader behavior at runtime by creating functions from existing ones and preferentially linking to others in a dynamic library.

```
class MTLStitchedLibraryDescriptor
```

A description of a new library of procedurally generated functions.

```
class MTLFunctionStitchingGraph
```

A description of a new stitched function.

```
class MTLFunctionStitchingInputNode
```

A call graph node that describes an input to the call graph.

```
class MTLFunctionStitchingFunctionNode
```

A call graph node that describes a function call and its inputs.

```
protocol MTLFunctionStitchingNode
```

A protocol to identify call graph nodes.

```
class MTLFunctionStitchingAttributeAlwaysInline
```

An attribute to specify that Metal needs to inline all of the function calls when generating the stitched function.

```
protocol MTLFunctionStitchingAttribute
```

A protocol to identify types that customize how the Metal compiler stitches a function together.

Compile-time variant functions

```
class MTLFunctionConstant
```

A constant that specializes the behavior of a shader.

`class MTLFunctionConstantValues`

A set of constant values that specialize a graphics or compute GPU function.

Introspection data

`class MTLComputePipelineReflection`

Information about the arguments of a compute function.

`typealias MTLAutoreleasedComputePipelineReflection`

A convenience type alias for an autoreleased compute pipeline reflection object.

`class MTLRenderPipelineReflection`

Information about the arguments of a graphics function.

`typealias MTLAutoreleasedRenderPipelineReflection`

A convenience type alias for an autoreleased pipeline reflection instance.

`enum MTLBindingType`

`protocol MTLBinding`

`enum MTLBindingAccess`

`protocol MTLBufferBinding`

`protocol MTLTextureBinding`

`protocol MTLThreadgroupBinding`

`protocol MTLObjectPayloadBinding`

Function arguments

`class MTLAttribute`

An object that describes an attribute defined in the stage-in argument for a shader.

`class MTLVertexAttribute`

An instance that represents an attribute of a vertex function.

~~`class MTLArgument`~~

Information about an argument of a graphics or compute function.

Deprecated

~~typealias MTLAutoreleasedArgument~~

A convenience type alias for an autoreleased argument instance.

Deprecated

~~enum MTLArgumentType~~

The resource type for an argument of a function.

Deprecated

~~typealias MTLArgumentAccess~~

Function access restrictions to argument data in the shading language code.

Deprecated

Shader types

class MTLType

A description of a data type.

enum MTLDatatype

The types of GPU functions, including shaders and compute kernels.

class MTLArrayType

A description of an array.

class MTLStructType

A description of a structure.

class MTLStructMember

An instance that provides information about a field in a structure.

class MTLPointertype

A description of a pointer.

class MTLTextureReferenceType

A description of a texture.

Shader logging

class MTLLogStateDescriptor

An interface that represents a log state configuration.

```
protocol MTLLogState
```

A container for shader log messages.

Errors

```
struct MTLLibraryError
```

Metal errors related to libraries.

```
enum Code
```

Error codes for Metal library errors.

```
let MTLLibraryErrorDomain: String
```

The error domain for Metal libraries.

See Also

Shader compilation and libraries

 Using the Metal 4 compilation API

Control when and how you compile an app's shaders.

 Using function specialization to build pipeline variants

Create pipelines for different levels of detail from a common shader source.