Flexible Array Members

in

Swift

AudioBufferList

```
struct AudioBufferList {
    UInt32 mNumberBuffers;
    AudioBuffer mBuffers[1];
};

struct AudioBufferList {
    var mNumberBuffers: UInt32
    var mBuffers: (AudioBuffer)
}
```

In an Ideal World

```
let audioBuffers = [
    AudioBuffer(...),
    AudioBuffer(...)
]

var audioBufferList = AudioBufferList()
audioBufferList.mNumberBuffers = UInt32(audioBuffers.count)
audioBufferList.mBuffers = audioBuffers // problems start here
```

Here in Reality

```
audioBufferList.mBuffers = audioBuffers
```

results in:

Cannot assign a value of type '[AudioBuffer]'
to a value of type '(AudioBuffer)'

— Swift Compiler

A Tuple is not an Array

```
struct AudioBufferList {
    UInt32 mNumberBuffers;
    AudioBuffer mBuffers[1]; // C array
};
```

```
struct AudioBufferList {
    var mNumberBuffers: UInt32
    var mBuffers: (AudioBuffer) // (AudioBuffer) == AudioBuffer
}
```

Flexible Array Member

- "struct hack"[1] became formalized in C99[2]
- fixed length struct members must include a count
- the variable length struct member must be last
- create with `malloc` and pass by pointer
- be aware of memory alignment issues

---

[1] http://c-faq.com/struct/structhack.html

[2] http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf

One Buffer to Rule Them All

```
struct AudioBufferList {
    UInt32 mNumberBuffers;
    AudioBuffer mBuffers[1];
};
```

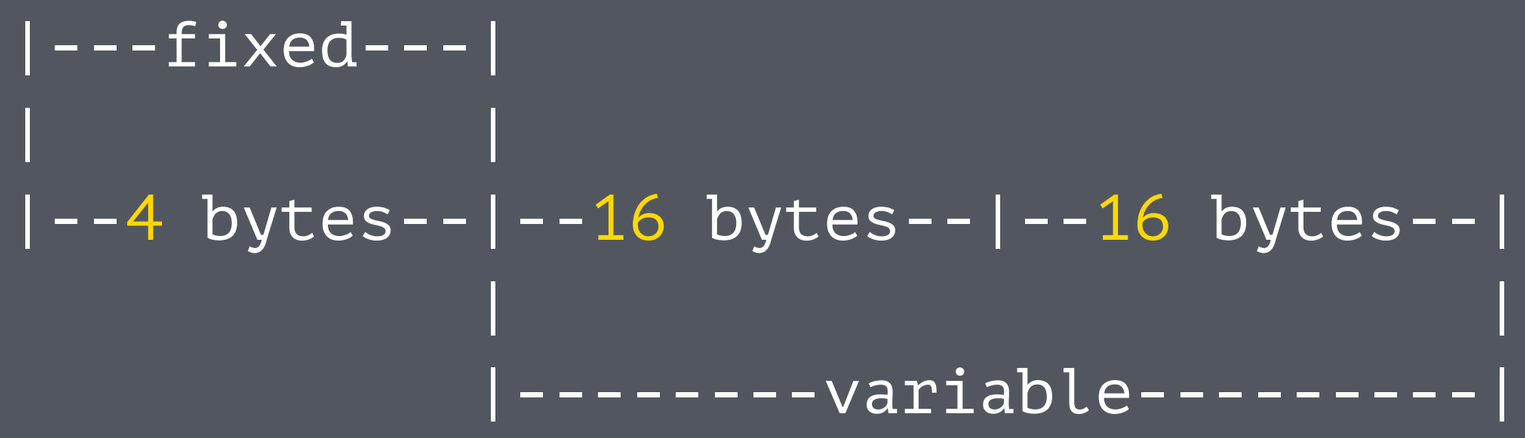
memory layout:

```
|---fixed---|
|           |
|--4 bytes--|--16 bytes--|
            |            |
            |--variable--|
```

Two Buffers

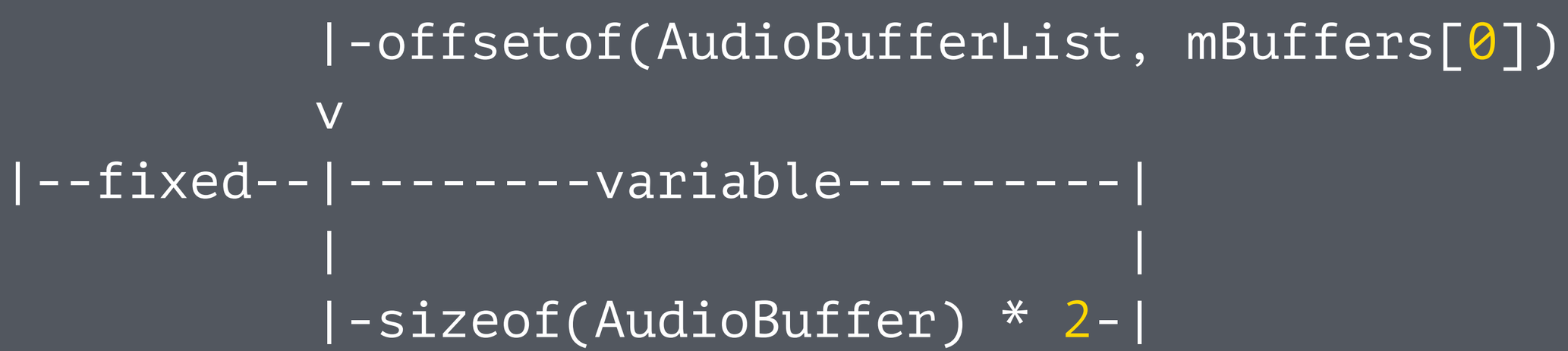
```
struct AudioBufferList {
    UInt32 mNumberBuffers;
    AudioBuffer mBuffers[1];
};
```

same type, more memory ???



Objective-C Allocation

```
size_t fixedLengthMemberSize = offsetof(AudioBufferList, mBuffers[0]);
size_t variableLengthMemberSize = sizeof(AudioBuffer) * 2;
size_t totalSize = fixedLengthMemberSize + variableLengthMemberSize;

AudioBufferList *audioBufferList = malloc(size);
```

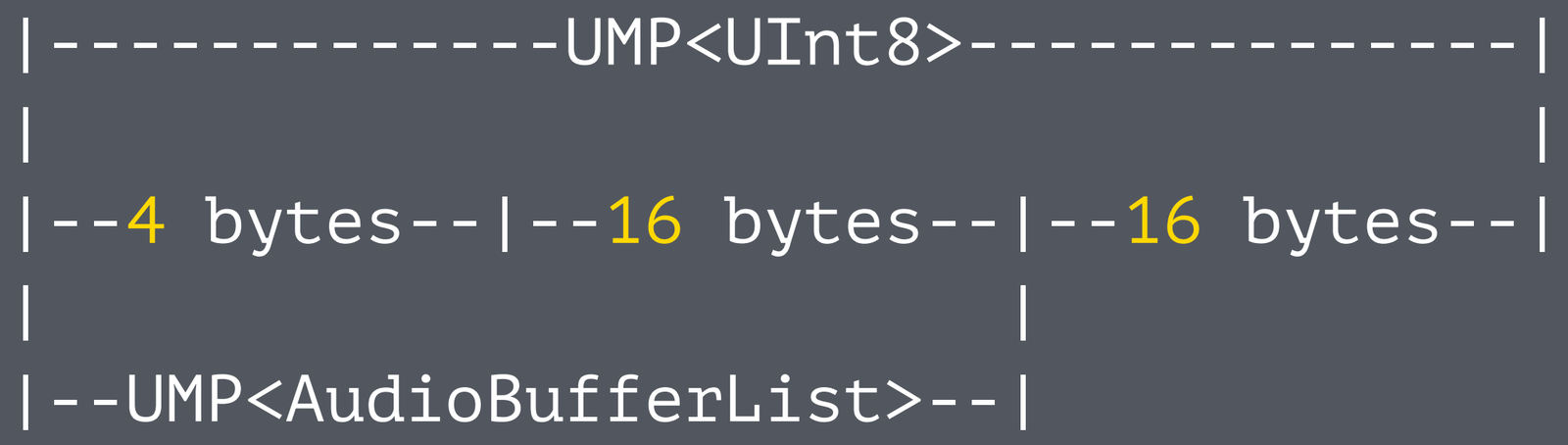


Objective-C Usage

```
audioBufferList->mNumberBuffers = 2;

audioBufferList->mBuffers[0].mNumberChannels = 1;
...
audioBufferList->mBuffers[1].mNumberChannels = 1;
...

// don't forget to free the memory at some point
free(audioBufferList);
```

UnsafeMutablePointer<Memory>[3]

"A pointer to an object of type Memory. This type provides no automated memory management, and therefore the user must take care to allocate and free memory appropriately."



[3] http://swiftdoc.org/type/UnsafeMutablePointer/

Swift Allocation[4]

In Swift, use `strideof` where you would use `sizeof` in ObjC[5]

```
var audioBuffers = [...]

let size = strideof(AudioBufferList) + strideof(AudioBuffer) * (audioBuffers.count - 1)
let memoryPointer = UnsafeMutablePointer<UInt8>.alloc(size)
```

---

[4] http://stackoverflow.com/questions/27724055/initializing-midimetaevent-structure

[5] https://devforums.apple.com/message/1086617#1086617

Swift Usage

In Swift we dereference with the `memory` property.

```
var audioBufferListPointer = UnsafeMutablePointer<AudioBufferList>(memoryPointer)

audioBufferListPointer.memory.mNumberBuffers = UInt32(audioBuffers.count)

// this doesn't work here either
audioBufferListPointer.memory.mBuffers = ...
```

Swift Usage

Memory layout is only guaranteed for structs declared in C.

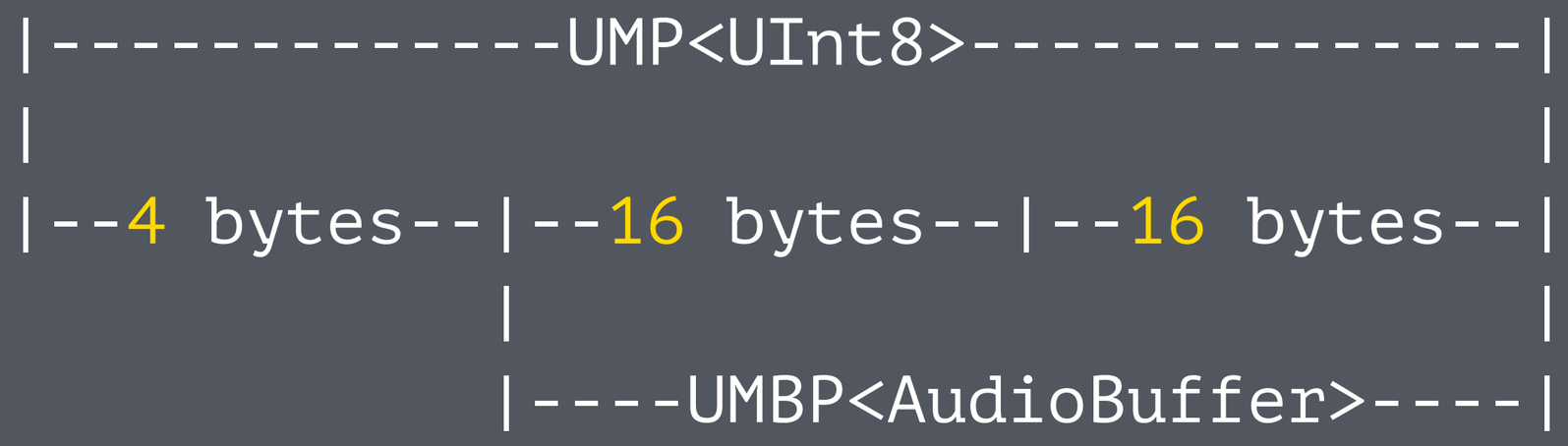
```
var audioBufferListPointer: UnsafeMutablePointer<AudioBufferList>(memoryPointer) = ...

memcpy(&audioBufferListPointer.memory.mBuffers, &audioBuffers, size)

// don't forget to release the memory when you're done
memoryPointer.dealloc(size)
```

Swift Wrapper

```
class AudioBufferListWrapper {

    let count: Int

    var audioBufferListPointer: UnsafeMutablePointer<AudioBufferList>

    private let size: Int
    private let memoryPointer: UnsafeMutablePointer<UInt8>

    init(count: Int) {
        self.count = count
        self.size = strideof(AudioBufferList) + strideof(AudioBuffer) * (count - 1)

        self.memoryPointer = UnsafeMutablePointer<UInt8>.alloc(size)
        audioBufferListPointer = UnsafeMutablePointer<AudioBufferList>(memoryPointer)

        audioBufferListPointer.memory.mNumberBuffers = UInt32(count)

        // we still need to populate the AudioBufferList
    }

    deinit {
        memoryPointer.dealloc(size)
    }

}
```

UnsafeMutableBufferPointer<Element>[6]

"A non-owning pointer to buffer of mutable Elements stored contiguously in memory, presenting a Collection interface to the underlying elements."



[6] http://swiftdoc.org/type/UnsafeMutableBufferPointer/

Swift Wrapper Redux

```
class AudioBufferListWrapper {

    let count: Int

    var audioBufferListPointer: UnsafeMutablePointer<AudioBufferList>

    private let size: Int
    private let memoryPointer: UnsafeMutablePointer<UInt8>
    private let audioBuffersPointer: UnsafeMutableBufferPointer<AudioBuffer> // our buffer pointer

    init(count: Int) {
        self.count = count
        self.size = strideof(AudioBufferList) + strideof(AudioBuffer) * (count - 1)

        self.memoryPointer = UnsafeMutablePointer<UInt8>.alloc(size)
        audioBufferListPointer = UnsafeMutablePointer<AudioBufferList>(memoryPointer)

        audioBufferListPointer.memory.mNumberBuffers = UInt32(count)

        // initialize our buffer pointer
        audioBuffersPointer = UnsafeMutableBufferPointer(start: &audioBufferListPointer.memory.mBuffers, count: count)
    }

    deinit {
        memoryPointer.dealloc(size)
    }

}
```

Adopt MutableCollectionType

```
extension AudioBufferListWrapper: MutableCollectionType {

    var startIndex: Int { return 0 }
    var endIndex: Int { return count }

    func generate() -> IndexingGenerator<AudioBufferListWrapper> {
        return IndexingGenerator(self)
    }

    subscript (index: Int) -> AudioBuffer {
        get {
            return audioBuffersPointer[index]
        }
        set {
            audioBuffersPointer[index] = newValue
        }
    }

}
```

Swift Wrapper Usage

```
let wrapper = AudioBufferListWrapper(count: 2)
wrapper[0] = AudioBuffer(...)
wrapper[1] = AudioBuffer(...)

let firstBuffer = wrapper[0]
let secondBuffer = wrapper[1]

SomeCFunction(wrapper.audioBufferListPointer)
```

Apple's Take

```
extension AudioBufferList {
    static func sizeInBytes(#maximumBuffers: Int) -> Int
    static func allocate(#maximumBuffers: Int) -> UnsafeMutableAudioBufferListPointer
}

extension AudioBuffer {
    init<T>(_ typedBuffer: UnsafeMutableBufferPointer<T>, numberOfChannels: Int)
}

struct UnsafeMutableAudioBufferListPointer {
    init(_ p: UnsafeMutablePointer<AudioBufferList>)
    var count: Int { get nonmutating set }
    var unsafePointer: UnsafePointer<AudioBufferList> { get }
    var unsafeMutablePointer: UnsafeMutablePointer<AudioBufferList>
}

extension UnsafeMutableAudioBufferListPointer : MutableCollectionType {
    func generate() -> IndexingGenerator<UnsafeMutableAudioBufferListPointer>
    var startIndex: Int { get }
    var endIndex: Int { get }
    subscript (index: Int) -> AudioBuffer { get nonmutating set }
}
```

Recommendations

- avoid `memcpy` approach
  - write only
  - depends on C struct memory layout
- use ObjC or Apple wrappers for light usage
- consider custom wrappers
  - for heavy usage
  - teams with less C experience

Frameworks with FAM Structs

- AudioToolbox
- AudioUnit
- CoreAudio
- CoreMIDI
- CoreText
- CoreVideo
- GLKit

Marc Schwieterman

@mschwieterman

marcschwieterman.com

initiative.fm

github.com/marcisme/talks/tree/master/FAM