

[Accelerate](#) / Decompressing and Parsing an Archived String

Article

Decompressing and Parsing an Archived String

Recreate a string from an archive file.



Overview

In this article, you'll learn how to use AppleArchive to decompress and parse a previously compressed string.

The code below decompresses and parses the file generated using the steps explained in [Compressing and saving a string to the file system](#). The operation obtains the contents of the DAT blob for the first file in the archive and creates a string from that data.

Specify the compressed file path

Create a `FilePath` structure that specifies the file name and location of the AppleArchive file that stores the compressed data. You must add read and write file access to the Downloads folder in the Signing and Capabilities pane. To learn more about configuring the App Sandbox, see [Configure App Sandbox](#).

The following code creates a file path to `lorem.aar`:

```
let archiveFileName = "lorem.aar"

let archiveFilePath: FilePath = {
    guard let downloadURL = FileManager.default.urls(for: .downloadsDirectory,
                                                       in: .userDomainMask).first,
        let archiveFilePath = FilePath(downloadURL.appendingPathComponent(archiveName))
    fatalError("Unable to create archive file path.")
}
```

```
    return archiveFilePath  
}()
```

Create the file stream to read the source archive

The [ArchiveByteStream](#) class provides static factory methods that create streams for different functions. In this case, use [`fileStream\(path:mode:options:permissions:\)`](#) to create a byte stream that reads the source file:

```
guard let readFileStream = ArchiveByteStream.fileStream(  
    path: archiveFilePath,  
    mode: .readOnly,  
    options: [ ],  
    permissions: FilePermissions(rawValue: 0o644)) else {  
    print("can't create reading file stream")  
    return  
}  
defer {  
    try? readFileStream.close()  
}
```

Create the decompression stream

Create the decompression stream. Specify the file-reading stream as the input stream that provides the compressed data:

```
guard let decompressStream = ArchiveByteStream.decompressionStream(  
    readingFrom: readFileStream) else {  
    return  
}  
defer {  
    try? decompressStream.close()  
}
```

Create the decoding stream

Create a decoding stream that provides archive elements from the raw, decompressed data:

```
guard let decodeStream = ArchiveStream.decodeStream(  
    readingFrom: decompressStream) else {  
    return  
}  
defer {  
    try? decodeStream.close()  
}
```

Derive the size of the uncompressed string

Use the size of the DAT blob field that you specified in [Compressing and saving a string to the file system](#) to create a buffer to receive the uncompressed data. To access the size, read the DAT field of the decode stream's header:

```
guard  
    let header = try? decodeStream.readHeader(),  
    let datField = header.field(forKey: ArchiveHeader.FieldKey("DAT")) else {  
    return  
}  
  
let byteCount: UInt64  
  
switch datField {  
    case .blob(_, let size, _):  
        byteCount = size  
    default:  
        byteCount = 0  
}  
  
guard byteCount != 0 else {  
    return  
}
```

Decompress the archived string

Create an [UnsafeMutableRawBufferPointer](#) structure and allocate to it the size of the original string to receive the decompressed data:

```
let rawBufferPtr = UnsafeMutableRawBufferPointer.allocate(  
    byteCount: Int(byteCount),
```

```
alignment: MemoryLayout<UTF8>.alignment)
```

```
defer {
    rawBufferPtr.deallocate()
}
```

Call `doc://com.apple.documentation/documentation/applearchive/archivestream/3589310-readblob` to read the decompressed data from the DAT field and write it to the raw buffer pointer. The decode stream parses its input from the decompression stream that, in turn, decompresses its input from the AppleArchive file supplied by the file stream.

```
do {
    try decodeStream.readBlob(key: ArchiveHeader.FieldKey("DAT"),
                               into: rawBufferPtr)
} catch {
    print("Unable to read from stream into `rawBufferPtr`.")
}
```

Create a string from the raw buffer pointer

Create a string from the raw buffer pointer by creating a typed pointer that's bound to `CChar`, and use `init(cString:)` to initialize the new string:

```
let typedPtr = rawBufferPtr.bindMemory(to: CChar.self)

let decompressedString = String(cString: typedPtr.baseAddress!)

print(decompressedString)

// prints:
// Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fermentum vestik
// [...]
```

See Also

Directories, Files, and Data Archives

 Compressing single files

Compress a single file and store the result on the file system.

 Decompressing single files

Recreate a single file from a compressed file.

 Compressing file system directories

Compress the contents of an entire directory and store the result on the file system.

 Decompressing and extracting an archived directory

Recreate an entire file system directory from an archive file.

 Compressing and saving a string to the file system

Compress the contents of a Unicode string and store the result on the file system.