Article

Compressing and decompressing data with buffer compression

Compress a string, write it to the file system, and decompress the same file using buffer compression.

Overview

The code in this article uses the Compression framework to encode (compress) and decode (decompress) a string. The code writes the encoded result to the temporary directory that the `NSTemporaryDirectory()` function returns.

The code in this sample is useful in applications that store or transmit text files where saving or sending smaller files can improve performance and reduce storage overhead. This sample app implements *buffer compression*, where it reads the contents of a source buffer in a single step to compress or decompress data.

Create the source data

Typically, your app would dynamically generate the source data that it compresses, but for this example, the source data is a hard-coded string.

```
import Compression

let sourceString = """
    Lorem ipsum dolor sit amet consectetur adipiscing elit mi
    nibh ornare proin blandit diam ridiculus, faucibus mus
    dui eu vehicula nam donec dictumst sed vivamus bibendum
    aliquet efficitur. Felis imperdiet sodales dictum morbi
    vivamus augue dis duis aliquet velit ullamcorper porttitor,
    lobortis dapibus hac purus aliquam natoque iaculis blandit
    montes nunc pretium.
    """

var sourceBuffer = Array(sourceString.utf8)
```

On return, `sourceBuffer` is an array of `UInt8` values that contains the UTF-8 representation of the source string.

Create the destination buffer

Create an `UnsafeMutablePointer` structure and allocate it with a capacity of the source string's `count` to receive the encoded data.

```
let destinationBuffer = UnsafeMutablePointer<UInt8>.allocate(capacity: sourceString.

defer {
    destinationBuffer.deallocate()
}
```

Select a compression algorithm

The code in this example uses the `COMPRESSION_LZFSE` algorithm, which provides the compression ratio of zlib level 5, but with much higher energy efficiency and speed (between 2x and 3x) for both encode and decode operations.

```
let algorithm = COMPRESSION_LZFSE
```

For apps that require interoperability with non-Apple devices, use `COMPRESSION_ZLIB` instead. For more information on other compression algorithms, see `compression_algorithm`.

Compress the data

The `compression_encode_buffer(_:_:_:_:_:_:)` function compresses the data, writes the result to the destination buffer, and returns the size of the encoded data.

```
let compressedSize = compression_encode_buffer(destinationBuffer, sourceString.count
                                               &sourceBuffer, sourceString.count,
                                               nil,
                                                algorithm)
```

When working with small files, the compression may fail and `compression_encode_buffer(_:_:_:_:_:_:)` returns `0`.

```
if compressedSize == 0 {
    fatalError("Encoding failed.")
}
```

You may elect to handle this situation differently, for example, by displaying a warning to the user that the compression failed.

Write the encoded data to a file

The code below writes the encoded data to a file in the the app's temporary directory on macOS:

```
let encodedFileName = "stringEncoded.LZFSE"

let tempDirURL = NSURL(fileURLWithPath: NSTemporaryDirectory())

guard
    let encodedFileURL = tempDirURL.appendingPathComponent(encodedFileName) else {
        return
}

FileManager.default.createFile(atPath: encodedFileURL.path,
                               contents: nil,
                               attributes: nil)

guard let destinationFileHandle = try? FileHandle(forWritingTo: encodedFileURL) else
    print("destinationFileHandle fail.")
    return
}

let encodedData = NSData(bytesNoCopy: destinationBuffer,
                         length: compressedSize,
                         freeWhenDone: false)

destinationFileHandle.write(encodedData as Data)
destinationFileHandle.closeFile()
```

Read the encoded data from a file

To read the encoded file, create a file handle for reading from the encoded file's URL.

```
guard
    let encodedFileHandle = try? FileHandle(forReadingFrom: encodedFileURL) else {
        print("encodedFileHandle fail.")
        return
}
```

Use the file handle to read the entire encoded file and populate `encodedSourceData`.

```
let encodedSourceData = encodedFileHandle.readDataToEndOfFile()
```

Decompress the data

Allocate memory to contain the decoded data. Typically, the encoded payload would be part of a larger structure containing additional metadata such as the uncompressed size, and you'd use that to define the buffer capacity. However, for this example, allocate 8 MB:

```
let decodedCapacity = 8_000_000
let decodedDestinationBuffer = UnsafeMutablePointer<UInt8>.allocate(capacity: decode
defer {
    decodedDestinationBuffer.deallocate()
}
```

Use `compression_decode_buffer(_:_:_:_:_:_:)` to decode the raw bytes of the encoded source data and write the result to `decodedDestinationBuffer`. You can create a string from the destination buffer using the `init(cString:)` initializer.

```
let decodedString: String = encodedSourceData.withUnsafeBytes { encodedSourceBuffer
    let typedPointer = encodedSourceBuffer.bindMemory(to: UInt8.self)
    let decodedCharCount = compression_decode_buffer(decodedDestinationBuffer, decod
                                                      typedPointer.baseAddress!, enco
                                                     nil,
                                                      algorithm)

    return String(cString: decodedDestinationBuffer)
}
```

The `compression_decode_buffer(_:_:_:_:_:_:)` function returns the size of the decoded data. If the decompression fails, the size returned is zero. This may indicate that the memory allocated to the destination buffer is insufficient and you should switch to the stream API or retry with a larger buffer.

See Also

Compression

{} Compressing and decompressing files with stream compression

Perform compression for all files and decompression for files with supported extension types.

Compressing and decompressing data with input and output filters

Compress and decompress streamed or from-memory data, using input and output filters.