Class

FileHandle

An object-oriented wrapper for a file descriptor.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class FileHandle
```

Mentioned in

About Apple File System

Overview

You use file handle objects to access data associated with files, sockets, pipes, and devices. For files, you can read, write, and seek within the file. For sockets, pipes, and devices, you can use a file handle object to monitor the device and process data asynchronously.

Most creation methods for `FileHandle` cause the file handle object to take ownership of the associated file descriptor. This means that the file handle object both creates the file descriptor and is responsible for closing it later, usually when the system deallocates the file handle object. If you want to use a file handle object with a file descriptor that you created, use the `init(fileDescriptor:)` method or use the `init(fileDescriptor:closeOnDealloc:)` method and pass `false` for the `flag` parameter.

Run Loop Considerations

When using a file handle object to communicate asynchronously with a socket, you must initiate the corresponding operations from a thread with an active run loop. Although the read, accept, and wait operations themselves are performed asynchronously on background threads, the file

handle uses a run loop source to monitor the operations and notify your code appropriately. Therefore, you must call those methods from your application's main thread or from any thread where you've configured a run loop and are using it to process events.

Topics

Creating a File Handle

`convenience init(fileDescriptor: Int32)`

Creates and returns a file handle object associated with the specified file descriptor.

`init(fileDescriptor: Int32, closeOnDealloc: Bool)`

Creates and returns a file handle object associated with the specified file descriptor and deallocation policy.

`convenience init?(forReadingAtPath: String)`

Returns a file handle initialized for reading the file, device, or named socket at the specified path.

`convenience init(forReadingFromURL: URL) throws`

Returns a file handle initialized for reading the file, device, or named socket at the specified URL.

`convenience init?(forWritingAtPath: String)`

Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

`convenience init(forWritingToURL: URL) throws`

Returns a file handle initialized for writing to the file, device, or named socket at the specified URL.

`convenience init?(forUpdatingAtPath: String)`

Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.

`convenience init(forUpdatingURL: URL) throws`

Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified URL.

`init?(coder: NSCoder)`

Returns a file handle initialized from data in an unarchiver.

Getting a File Handle

`class var standardError: FileHandle`

The file handle associated with the standard error file.

`class var standardInput: FileHandle`

The file handle associated with the standard input file.

`class var standardOutput: FileHandle`

The file handle associated with the standard output file.

`class var nullDevice: FileHandle`

The file handle associated with a null device.

Getting a File Descriptor

`var fileDescriptor: Int32`

The POSIX file descriptor associated with the receiver.

Reading from a File Handle Asynchronously

`var bytes: FileHandle.AsyncBytes`

The file's contents, as an asynchronous sequence of bytes.

`struct AsyncBytes`

An asynchronous sequence of bytes.

Reading from a File Handle Synchronously

`var availableData: Data`

The data currently available in the receiver.

`func readToEnd() throws -> Data?`

Reads the available data synchronously up to the end of file or maximum number of bytes.

`func read(upToCount: Int) throws -> Data?`

Reads data synchronously up to the specified number of bytes.

Reading Asynchronously with Notifications

`func acceptConnectionInBackgroundAndNotify()`

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the “near” (client) end of the communications channel.

`func acceptConnectionInBackgroundAndNotify(forModes: [RunLoop.Mode]?)`

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the “near” (client) end of the communications channel.

`func readInBackgroundAndNotify()`

Reads from the file or communications channel in the background and posts a notification when finished.

`func readInBackgroundAndNotify(forModes: [RunLoop.Mode]?)`

Reads from the file or communications channel in the background and posts a notification when finished.

`func readToEndOfFileInBackgroundAndNotify()`

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

`func readToEndOfFileInBackgroundAndNotify(forModes: [RunLoop.Mode]?)`

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

`func waitForDataInBackgroundAndNotify()`

Asynchronously checks to see if data is available.

`func waitForDataInBackgroundAndNotify(forModes: [RunLoop.Mode]?)`

Asynchronously checks to see if data is available.

Writing to a File Handle

`func write<T>(contentsOf: T) throws`

Writes the specified data synchronously to the file handle.

Seeking Within a File

`func offset() throws -> UInt64`

Gets the position of the file pointer within the file.

`func seekToEnd() throws -> UInt64`

Places the file pointer at the end of the file referenced by the file handle and returns the new file offset.

`func seek(toOffset: UInt64) throws`

Moves the file pointer to the specified offset within the file.

Operating on a File

`func close() throws`

Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.

`func synchronize() throws`

Causes all in-memory data and attributes of the file represented by the file handle to write to permanent storage.

`func truncate(atOffset: UInt64) throws`

Truncates or extends the file represented by the file handle to a specified offset within the file and puts the file pointer at that position.

Monitoring for Readability and Writability

Set these properties if you want to use a block that reads or writes based on the file handle's availability.

`var readabilityHandler: ((FileHandle) -> Void)?`

The block to use for reading the contents of the file handle asynchronously.

`var writeabilityHandler: ((FileHandle) -> Void)?`

The block to use for writing the contents of the file handle asynchronously.

Constants

Keys for Notification UserInfo Dictionary

Strings that the system uses as keys in a userinfo dictionary during a file handle notification.

Exception Names

Constant that defines the name of a file operation exception.

Notifications

`NSFileHandle` posts several notifications related to asynchronous background I/O operations. They are set to post when the run loop of the thread that started the asynchronous operation is idle.

```
static let NSFileHandleConnectionAccepted: NSNotification.Name
```

Posted when a file handle object establishes a socket connection between two processes, creates a file handle object for one end of the connection, and makes this object available to observers.

```
static let NSFileHandleDataAvailable: NSNotification.Name
```

Posted when the file handle determines that data is currently available for reading in a file or at a communications channel.

```
class let readCompletionNotification: NSNotification.Name
```

Posted when the file handle reads the data currently available in a file or at a communications channel.

```
static let NSFileHandleReadToEndOfFileCompletion: NSNotification.Name
```

Posted when the file handle reads all data in the file or, in a communications channel, until the other process signals the end of data.

Deprecated

~~`func readDataToEndOfFile() -> Data`~~

Reads the available data synchronously up to the end of file or maximum number of bytes.

Deprecated

~~`func readData(ofLength: Int) -> Data`~~

Reads data synchronously up to the specified number of bytes.

Deprecated

~~`func write(Data)`~~

Writes the specified data synchronously to the file handle.

Deprecated

~~`var offsetInFile: UInt64`~~

The position of the file pointer within the file represented by the file handle.

Deprecated

~~`func seekToEndOfFile() -> UInt64`~~

Places the file pointer at the end of the file referenced by the file handle and returns the new file offset.

Deprecated

~~`func seek(toFileOffset: UInt64)`~~

Moves the file pointer to the specified offset within the file represented by the receiver.

Deprecated

~~`func closeFile()`~~

Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.

Deprecated

~~`func synchronizeFile()`~~

Causes all in-memory data and attributes of the file represented by the handle to write to permanent storage.

Deprecated

~~`func truncateFile(atOffset: UInt64)`~~

Truncates or extends the file represented by the file handle to a specified offset within the file and puts the file pointer at that position.

Deprecated

~~`let NSFileHandleNotificationMonitorModes: String`~~

Currently unused.

Deprecated

Structures

`struct ConnectionAcceptedMessage`

`struct DataAvailableMessage`

`struct ReadCompletionMessage`

`struct ReadToEndOfFileCompletionMessage`

Relationships

Inherits From

NSObject

Conforms To

CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSCoding
NSObjectProtocol
NSSecureCoding
Sendable
SendableMetatype

See Also

Managed file access

class NSFileSecurity

A stub class that encapsulates security information about a file.

class NSFileVersion

A snapshot of a file at a specific point in time.

class FileWrapper

A representation of a node (a file, directory, or symbolic link) in the file system.