# NSPipe

| | |
|---|---|
| **Inherits From:** | NSObject |
| **Conforms To:** | NSObject (NSObject) |
| **Declared In:** | Foundation/NSFileHandle.h |

## Class Description

An NSPipe represents both ends of a pipe and enables communication through the pipe. A pipe is a one-way communications channel between related processes; one process writes data while the other process reads that data. The data that passes through the pipe is buffered; the size of the buffer is determined by the underlying operating system.

Each end point of the pipe is a file descriptor, represented by an NSFileHandle. You thus use NSFileHandle messages to read and write pipe data. A "parent" process creates the NSPipe and holds one end of it. It creates an NSTask for the other process and, before launching it, passes the other end of the pipe to that process; it does this by setting the NSTask's standard input, standard output, or standard error device to be the other NSFileHandle or the NSPipe itself (in the latter case, the type of NSFileHandle—reading or writing—is determined by the NSTask "set" method).

The following example illustrates the above procedure:

```
- (void)readTaskData:(id)sender
{
    NSTask *pipeTask = [[NSTask alloc] init];
    NSPipe *newPipe = [NSPipe pipe];
    NSFileHandle *readHandle = [newPipe fileHandleForReading];
    NSData *inData = nil;

    [pipeTask setStandardOutput:newPipe]; // write handle is closed to this process
    [pipeTask setLaunchPath:[NSHomeDirectory()
stringByAppendingPathComponent:@"PipeTask.app/PipeTask"]];
    [pipeTask launch];

    while ((inData = [readHandle availableData]) && [inData length]) {
        [inData processData];
   }
}
```

The launched process in this example must get data and write that data, using NSFileHandle's **writeData:**, to its standard output device (obtained NSFileHandle's **fileHandleWithStandardOutput**).

When the processes have no more data to communicate across the pipe, the writing process should simply send **closeFile** to its NSFileHandle end point. This causes the process with the “read” NSFileHandle to receive an empty NSData, signalling end of data. If the “parent” process created the NSPipe with the **init** method, it should then release it.

## Method Types

| | |
|---|---|
| Creating an NSPipe | - init<br>+ pipe |
| GettingNSFileHandles for pipe | – fileHandleForReading<br>– fileHandleForWriting |

## Class Methods

### pipe

+ (id)**pipe**

Returns an NSPipe that’s been sent **autorelease**. Because the returned object will be deallocated at the end of the current loop, retain the object or, better yet, use the **init** method if you intend to keep the object beyond that point. Returns **nil** if the method encounters errors while attempting to create the pipe or the NSFileHandle end points.

## Instance Methods

### fileHandleForReading

– (NSFileHandle *)**fileHandleForReading**

Returns an NSFileHandle that accepts messages that read pipe data: **availableData**, **readDataToEndOfFile**, **readDataOfLength:**. You don’t need to send **closeFile** to this object or explicitly release after you’re finished using it. This descriptor represented by this object is deleted and the object itself is automatically deallocated when the NSPipe is deallocated.

### fileHandleForWriting

– (NSFileHandle *)**fileHandleForWriting**

Returns an NSFileHandle for writing to the pipe using NSFileHandle’s **writeData:**. When you are finished writing data to this object, send it **closeFile** to delete the descriptor, which causes the reading process to

receive an end-of-data signal (an empty NSData). This object is automatically deallocated when the NSPipe is deallocated.

## init

– (id)**init**

Returns an NSPipe. Returns **nil** if the method encounters errors while attempting to create the pipe or the NSFileHandles that serve as end points of the pipe.

**See also:** + **pipe**