

---

# NSMutableData Class Reference

[Cocoa](#) > [Data Management](#)



2007-03-26



Apple Inc.  
© 2007 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, Cocoa, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY**

**DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

---

## **NSMutableData Class Reference 5**

---

Overview	5
Tasks	6
Creating and Initializing an NSMutableData Object	6
Adjusting Capacity	6
Accessing Data	6
Adding Data	6
Modifying Data	6
Class Methods	7
dataWithCapacity:	7
dataWithLength:	7
Instance Methods	8
appendBytes:length:	8
appendData:	8
increaseLengthBy:	9
initWithCapacity:	9
initWithLength:	10
mutableBytes	10
replaceBytesInRange:withBytes:	11
replaceBytesInRange:withBytes:length:	11
resetBytesInRange:	12
setData:	12
setLength:	13

---

## **Document Revision History 15**

---

---

## **Index 17**

---



# NSMutableData Class Reference

---

<b>Inherits from</b>	NSData : NSObject
<b>Conforms to</b>	NSCoding (NSData) NSCopying (NSData) NSMutableCopying (NSData) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/Foundation.framework
<b>Availability</b>	Available in Mac OS X v10.0 and later.
<b>Companion guide</b>	Binary Data Programming Guide for Cocoa
<b>Declared in</b>	NSData.h
<b>Related sample code</b>	CocoaHTTPServer CocoaSOAP GridCalendar ImageClient URL CacheInfo

## Overview

`NSMutableData` (and its superclass `NSData`) provide data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Foundation objects. They are typically used for data storage and are also useful in Distributed Objects applications, where data contained in data objects can be copied or moved between applications. `NSData` creates static data objects, and `NSMutableData` creates dynamic data objects. You can easily convert one type of data object to the other with the initializer that takes an `NSData` object or an `NSMutableData` object as an argument.

`NSMutableData` is “toll-free bridged” with its Core Foundation counterpart, `CFData`. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSMutableData *` parameter, you can pass a `CFDataRef`, and in a function where you see a `CFDataRef` parameter, you can pass an `NSMutableData` instance (you cast one type to the other to suppress compiler warnings). See [Interchangeable Data Types](#) for more information on toll-free bridging.

## Tasks

### Creating and Initializing an NSMutableData Object

- + [dataWithCapacity:](#) (page 7)  
Creates and returns an NSMutableData object capable of holding the specified number of bytes.
- + [dataWithLength:](#) (page 7)  
Creates and returns an NSMutableData object containing a given number of zeroed bytes.
- [initWithCapacity:](#) (page 9)  
Returns an initialized NSMutableData object capable of holding the specified number of bytes.
- [initWithLength:](#) (page 10)  
Initializes and returns an NSMutableData object containing a given number of zeroed bytes.

### Adjusting Capacity

- [increaseLengthBy:](#) (page 9)  
Increases the length of the receiver by a given number of bytes.
- [setLength:](#) (page 13)  
Extends or truncates a mutable data object to a given length.

### Accessing Data

- [mutableBytes](#) (page 10)  
Returns a pointer to the receiver's data.

### Adding Data

- [appendBytes:length:](#) (page 8)  
Appends to the receiver a given number of bytes from a given buffer.
- [appendData:](#) (page 8)  
Appends the content of another NSData object to the receiver.

### Modifying Data

- [replaceBytesInRange:withBytes:](#) (page 11)  
Replaces with a given set of bytes a given range within the contents of the receiver.
- [replaceBytesInRange:withBytes:length:](#) (page 11)  
Replaces with a given set of bytes a given range within the contents of the receiver.
- [resetBytesInRange:](#) (page 12)  
Replaces with zeroes the contents of the receiver in a given range.

- [setData:](#) (page 12)

Replaces the entire contents of the receiver with the contents of another data object.

## Class Methods

### dataWithCapacity:

Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

```
+ (id)dataWithCapacity:(NSUInteger)aNumItems
```

#### Parameters

*aNumItems*

The number of bytes the new data object can initially contain.

#### Return Value

A new `NSMutableData` object capable of holding *aNumItems* bytes.

#### Discussion

This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to 0.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- + [dataWithLength:](#) (page 7)
- [initWithCapacity:](#) (page 9)
- [initWithLength:](#) (page 10)

#### Declared In

`NSData.h`

### dataWithLength:

Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

```
+ (id)dataWithLength:(NSUInteger)length
```

#### Parameters

*length*

The number of bytes the new data object initially contains.

#### Return Value

A new `NSMutableData` object of *length* bytes, filled with zeros.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- + [dataWithCapacity:](#) (page 7)
- [initWithCapacity:](#) (page 9)
- [initWithLength:](#) (page 10)

**Declared In**

NSData.h

## Instance Methods

### appendBytes:length:

Appends to the receiver a given number of bytes from a given buffer.

```
- (void)appendBytes:(const void *)bytes length:(NSUInteger)length
```

**Parameters**

*bytes*

A buffer containing data to append to the receiver's content.

*length*

The number of bytes from *bytes* to append.

**Discussion**

A sample using this method can be found in [Working With Mutable Binary Data](#).

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [appendData:](#) (page 8)

**Related Sample Code**

Core Data HTML Store  
QTSSConnectionMonitor  
QTSSInspector

**Declared In**

NSData.h

### appendData:

Appends the content of another `NSData` object to the receiver.

```
- (void)appendData:(NSData *)otherData
```

**Parameters***otherData*

The data object whose content is to be appended to the contents of the receiver.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

– [appendBytes:length:](#) (page 8)

**Related Sample Code**

GridCalendar

**Declared In**

NSData.h

**increaseLengthBy:**

Increases the length of the receiver by a given number of bytes.

– (void)increaseLengthBy:(NSUInteger)*extraLength*

**Parameters***extraLength*

The number of bytes by which to increase the receiver's length.

**Discussion**

The additional bytes are all set to 0.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

– [setLength:](#) (page 13)

**Declared In**

NSData.h

**initWithCapacity:**

Returns an initialized NSMutableData object capable of holding the specified number of bytes.

– (id)initWithCapacity:(NSUInteger)*capacity*

**Parameters***capacity*

The number of bytes the data object can initially contain.

**Return Value**

An initialized NSMutableData object capable of holding *capacity* bytes.

**Discussion**

This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to 0.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

+ [dataWithCapacity:](#) (page 7)

- [initWithLength:](#) (page 10)

**Declared In**

NSData.h

**initWithLength:**

Initializes and returns an `NSMutableData` object containing a given number of zeroed bytes.

```
- (id) initWithLength:(NSUInteger) length
```

**Parameters**

*length*

The number of bytes the object initially contains.

**Return Value**

An initialized `NSMutableData` object containing *length* zeroed bytes.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

+ [dataWithCapacity:](#) (page 7)

+ [dataWithLength:](#) (page 7)

- [initWithCapacity:](#) (page 9)

**Declared In**

NSData.h

**mutableBytes**

Returns a pointer to the receiver's data.

```
- (void *) mutableBytes
```

**Return Value**

A pointer to the receiver's data.

**Discussion**

If the length of the receiver's data is not zero, this function is guaranteed to return a pointer to the object's internal bytes. If the length of receiver's data *is* zero, this function may or may not return `NULL` dependent upon many factors related to how the object was created (moreover, in this case the method result might change between different releases).

A sample using this method can be found in [Working With Mutable Binary Data](#).

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSData.h`

**replaceBytesInRange:withBytes:**

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)bytes
```

**Parameters**

*range*

The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

*bytes*

The data to insert into the receiver's contents.

**Discussion**

If the location of *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised. The receiver is resized to accommodate the new bytes, if necessary.

A sample using this method is given in [Working With Mutable Binary Data](#).

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [replaceBytesInRange:withBytes:length:](#) (page 11)
- [resetBytesInRange:](#) (page 12)

**Declared In**

`NSData.h`

**replaceBytesInRange:withBytes:length:**

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)replacementBytes
    length:(NSUInteger)replacementLength
```

**Parameters***range*

The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

*replacementBytes*

The data to insert into the receiver's contents.

*replacementLength*

The number of bytes to take from *replacementBytes*.

**Discussion**

If the length of *range* is not equal to *replacementLength*, the receiver is resized to accommodate the new bytes. Any bytes past *range* in the receiver are shifted to accommodate the new bytes. You can therefore pass `NULL` for *replacementBytes* and 0 for *replacementLength* to delete bytes in the receiver in the range *range*. You can also replace a range (which might be zero-length) with more bytes than the length of the range, which has the effect of insertion (or “replace some and insert more”).

**Availability**

Available in Mac OS X v10.2 and later.

**See Also**

– [replaceBytesInRange:withBytes:](#) (page 11)

**Declared In**

`NSData.h`

**resetBytesInRange:**

Replaces with zeroes the contents of the receiver in a given range.

– `(void)resetBytesInRange:(NSRange)range`

**Parameters***range*

The range within the contents of the receiver to be replaced by zeros. The range must not exceed the bounds of the receiver.

**Discussion**

If the location of *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised. The receiver is resized to accommodate the new bytes, if necessary.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

– [replaceBytesInRange:withBytes:](#) (page 11)

**Declared In**

`NSData.h`

**setData:**

Replaces the entire contents of the receiver with the contents of another data object.

```
- (void)setData:(NSData *)aData
```

#### Parameters

*aData*

The data object whose content replaces that of the receiver.

#### Discussion

As part of its implementation, this method calls [replaceBytesInRange:withBytes:](#) (page 11).

#### Availability

Available in Mac OS X v10.0 and later.

#### Declared In

NSData.h

### setLength:

Extends or truncates a mutable data object to a given length.

```
- (void)setLength:(NSUInteger)length
```

#### Parameters

*length*

The new length for the receiver.

#### Discussion

If the mutable data object is extended, the additional bytes are filled with zeros.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [increaseLengthBy:](#) (page 9)

#### Declared In

NSData.h



# Document Revision History

---

This table describes the changes to *NSMutableData Class Reference*.

Date	Notes
2007-03-26	Corrected minor typographical errors.
2007-04-03	Enhanced discussion of <code>replaceBytesInRange:withBytes:length:</code> .
2006-10-03	Corrected typographical errors.
2006-05-23	First publication of this content as a separate document.

## REVISION HISTORY

### Document Revision History

# Index

---

## A

---

appendBytes:length: **instance method** [8](#)  
appendData: **instance method** [8](#)

## D

---

dataWithCapacity: **class method** [7](#)  
dataWithLength: **class method** [7](#)

## I

---

increaseLengthBy: **instance method** [9](#)  
initWithCapacity: **instance method** [9](#)  
initWithLength: **instance method** [10](#)

## M

---

mutableBytes **instance method** [10](#)

## R

---

replaceBytesInRange:withBytes: **instance method** [11](#)  
replaceBytesInRange:withBytes:length: **instance method** [11](#)  
resetBytesInRange: **instance method** [12](#)

## S

---

setData: **instance method** [12](#)  
setLength: **instance method** [13](#)