NSData Class Cluster

lass Cluster Description

NSData objects provide an object-oriented wrapper for byte buffers. This enables simple allocated buffers (that is, data with no embedded pointers) to take on the behavior of Foundation Kit objects. NSData is typically used for data storage. It is also useful in Distributed Objects applications, where data contained in NSData objects can be copied or moved between applications.

NSData objects can be used to wrap data of any size. When the data size is more than a few memory pages, NSData uses virtual memory management. NSData can also be used to wrap pre-existing data, regardless of how the data was allocated. NSData contains no information about the data itself (such as its type); the responsibility for deciding how to use the data lies with the client. In particular, it will not handle byte-order swapping when distributed between big-endian and little-endian machines. For typed data, use NSValue.

NSData provides an operating system-independent way to benefit from copy-on-write memory. The copy-on-write technique means that when data is copied through a virtual memory copy, an actual copy of the data is not made until there is an attempt to modify it.

The cluster's two public classes, NSData and NSMutableData, declare the programmatic interface for static and dynamic NSData objects, respectively.

The objects you create using these classes are referred to as *data objects*. Because of the nature of class clusters, data objects are not actual instances of the NSData or NSMutableData classes but instead are instances of one of their private subclasses. Although a data object's class is private, its interface is public, as declared by these abstract superclasses, NSData and NSMutableData.

Generally, you instantiate a data object by sending one of the **data...** messages to either the NSData or NSMutableData class object. These methods return a data object containing the bytes you pass in as arguments. If you use one of the **data...** methods whose name *does not* include "NoCopy" (such as **dataWithBytes:length:**), the bytes to be contained by the data object are copied as part of the instantiation process, and the data object then contains the copied bytes. When you subsequently release a data object that has been instantiated in this manner, the bytes contained by the data object—those that were copied during instantiation—are automatically freed. If you instantiate a data object with one of the methods whose name includes "NoCopy," however, (such as **dataWithBytesNoCopy:length:**) the bytes are not copied and are freed when the data object is released.

The NSData classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert between efficient, read-only data objects and mutable data objects.

NSData

Inherits From:	NSObject
Conforms To:	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Declared In:	Foundation/NSData.h Foundation/NSSerialization.h

Class at a Glance

Purpose

An NSData object stores immutable data in the form of bytes.

Principal Attributes

- A count of the number of bytes in the data object.
- The sequence of bytes contained in the data object.

Creation

+ data	Returns an empty data object.
+ dataWithBytes:length:	Returns a data object that contains a copy of the specified bytes.
+ dataWithBytesNoCopy:length:	Returns a data object that contains the specified bytes (without copying them).
+ dataWithContentsOfFile:	Returns a data object initialized with the contents of a file.
+ dataWithContentsOfMappedFile:	Returns a data object initialized with the contents of a mapped file.
+ dataWithData:	Returns a data object initialized with the contents of another data object.

Commonly Used Methods

– bytes	Returns a pointer to the data object’s contents.
– getBytes:	Copies the data object’s contents into a buffer.
– length	Returns the number of bytes contained by the data object.

Primitive Methods

– bytes
– length

lass Description

The NSData class declares the programmatic interface to an object that contains immutable data in the form of bytes. NSData’s two primitive methods—**bytes** and **length**—provide the basis for all of the other methods in the interface. The **bytes** method returns a pointer to the bytes contained in the data object. **length** returns the number of bytes contained in the data object.

NSData provides access methods for copying bytes from a data object into a specified buffer. **getBytes** copies all of the bytes into a buffer, whereas **getBytes:length:** copies bytes into a buffer of a given length. **getBytes:range:** copies a range of bytes from a starting point within the bytes themselves. You can also obtain a data object that contains a subset of the bytes in another data object by using the

subdataWithRange: method. Or, you can use the **description** method to return an NSString representation of the bytes in a data object.

For determining if two data objects are equal, NSData provides the **isEqualToData:** method, which does a byte-for-byte comparison.

The **writeToFile:atomically:** method enables you to write the contents of a data object to a file.

dopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copyWithZone:
NSMutableCopying	– mutableCopyWithZone:

ethod Types

Creating data objects	+ allocWithZone: + data + dataWithBytes:length: + dataWithBytesNoCopy:length: + dataWithContentsOfFile: + dataWithContentsOfMappedFile + dataWithData: + dataWithStream: – initWithBytes:length: – initWithBytesNoCopy:length: – initWithContentsOfFile: – initWithContentsOfMappedFile: – initWithData: – initWithStream:
Accessing data	– bytes – description – getBytes: – getBytes:length: – getBytes:range: – subdataWithRange:

Deserializing data	– deserializeAlignedBytesLengthAtCursor: – deserializeBytes:length:atCursor: – deserializeDataAt:ofObjCType:atCursor:context: – deserializeIntAtCursor: – deserializeIntAtIndex: – deserializeInts:count:atCursor: – deserializeInts:count:atIndex:
Testing data	– isEqualToData: – length
Storing data	– writeToFile:atomically:

lass Methods

allocWithZone

+ (id)**allocWithZone:**(NSZone *)*zone*

Creates and returns an uninitialized data object in the specified zone. If the receiver is the NSData class object, an instance of an appropriate immutable subclass is returned; otherwise, an object of the receiver's class is returned.

Typically, you create temporary data objects using the **data...** class methods, not the **alloc...** and **init...** methods.

data

+ (id)data

Creates and returns an empty data object. This method is declared primarily for the use of mutable subclasses of NSData.

dataWithBytes:length:

+ (id)**dataWithBytes:**(const void *)*bytes* **length:**(unsigned int)*length*

Creates and returns a data object containing *length* bytes copied from the buffer *bytes*.

See also: **+ dataWithBytesNoCopy:length:**

dataWithBytesNoCopy:length:

+ (id)**dataWithBytesNoCopy:**(void *)*bytes* **length:**(unsigned int)*length*

Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

See also: + **dataWithBytes:length:**

dataWithContentsOfFile:

+ (id)**dataWithContentsOfFile:**(NSString *)*path*

Creates and returns a data object by reading every byte from the file specified by *path*.

The following code example creates a data object *myData* initialized with the contents of **myFile.txt**. The path must be absolute.

```
NSString *thePath = @"/u/smith/myFile.txt";
NSData *myData = [NSData dataWithContentsOfFile:thePath];
```

See also: + **dataWithContentsOfMappedFile:**

dataWithContentsOfMappedFile:

+ (id)**dataWithContentsOfMappedFile:**(NSString *)*path*

Creates and returns a data object from the mapped file specified by *path*. Because of file mapping restrictions, this method should only be used if the file is guaranteed to exist for the duration of the data object's existence. It is generally safer to use the **dataWithContentsOfFile:** method.

This methods assumes that mapped files are available from the underlying operating system. A mapped file uses virtual memory techniques to avoid copying pages of the file into memory until they are actually needed.

See also: + **dataWithContentsOfFile:**

dataWithData:

+ (id)**dataWithData:**(NSData *)*aData*

Creates and returns a data object containing the contents of another data object, *aData*.

dataWithStream:

+ (id)**dataWithStream:**(NXStream *)*stream*

Creates and returns a data object containing the contents of *stream*.

See also: **– initWithStream:**

nstance Methods

bytes

– (const void *)**bytes**

Returns a pointer to the data object's contents. This method returns read-only access to the data.

See also: **– description, – getBytes:, – getBytes:length:, – getBytes:range:**

description

@protocol NSObject
– (NSString *)**description**

Returns an NSString object that contains a hexadecimal representation of the receiver's contents in the property list format for NSData objects.

See also: – **bytes, – getBytes:, – getBytes:length:, – getBytes:range:**

deserializeAlignedBytesLengthAtCursor:

– (unsigned int)**deserializeAlignedBytesLengthAtCursor:**(unsigned *)*cursor*

Reads a sequence of bytes from the receiver beginning at location *cursor* and returns them formatted as an unsigned integer. On return, *cursor* is set to the location just past the bytes that were read.

Use this method to read an integer that was serialized using the **serializeAlignedBytesLength:** method of NSMutableData. This method ignores any filler bytes that were serialized by the **serializeAlignedBytesLength:** method.

See also: **– deserializeIntAtCursor:, – deserializeIntAtIndex:**

deserializeBytes:length:atCursor:

– (void)**deserializeBytes:**(void *)*buffer*
  length:(unsigned int)*bytes*
  atCursor:(unsigned int*)*cursor*

Reads a sequence of bytes from the receiver beginning at location *cursor* and places them in *buffer*. The *bytes* parameter specifies the number of bytes to be read. On return, *cursor* is set to the location just beyond the bytes that were read.

See also: **– getBytes:range:**

deserializeDataAt:ofObjCType:atCursor:context:

– (void)**deserializeDataAt:**(void *)*data*
  ofObjCType:(const char *)*type*
  atCursor:(unsigned int*)*cursor*
  context:(id <NSObjCTypeSerializationCallBack>)*callback*

Reads a sequence of bytes from the receiver beginning at location *cursor* and places them in *data*. The bytes are formatted according to the Objective-C type code given in *type*. If *type* specifies an object, *callback* is used to deserialize the object; in such a case, *callback* must itself be an object that conforms to the NSObjCTypeSerializationCallBack protocol. If *type* does not specify an object, *callback* can be **nil**.

For information on on creating an Objective-C type code suitable for *type*, see the description of the **@encode()** compiler directive in *Object-Oriented Programming and the Objective-C Language*.

deserializeIntAtCursor:

– (int)**deserializeIntAtCursor:**(unsigned int*)*cursor*

Reads a sequence of bytes from the receiver beginning at location *cursor* and returns them as an integer value. On return, *cursor* is set to the location just past the bytes that were read.

See also: **– deserializeIntAtIndex:, – serializeInt: (NSMutableData)**

deserializeIntAtIndex:

– (int)**deserializeIntAtIndex:**(unsigned int)*index*

Reads a sequence of bytes from the receiver starting at *index* and returns them as an integer value.

See also: **– getBytes:range:, – serializeInt: (NSMutableData)**

deserializeInts:count:atCursor:

– (void)**deserializeInts:**(int *)*intBuffer*
  count:(unsigned int)*numInts*
  atCursor:(unsigned int*)*cursor*

Reads *numInts* integers as a sequence of bytes from the receiver and copies them into *intBuffer*. The bytes are read from the receiver beginning at location *cursor*. On return, *cursor* is set to the location just past the integers that were read.

See also: **– getBytes:range:, – deserializeIntAtCursor:, – serializeInts:count:** (NSMutableData)

deserializeInts:count:atIndex:

– (void)**deserializeInts:**(int *)*intBuffer*
  count:(unsigned int)*numInts*
  atIndex:(unsigned int)*index*

Reads *numInts* integers as a sequence of bytes from the receiver and copies them into *intBuffer*. The bytes are read from the receiver starting at *index*.

See also: **– getBytes:range:, – deserializeIntAtIndex:, – serializeInts:count:** (NSMutableData)

getBytes:

– (void)**getBytes:**(void *)*buffer*

Copies a data object's contents into *buffer***.**

For example, the following code excerpt initializes a data object *myData* with the NSString *myString*. It then uses **getBytes:** to copy the contents of *myData* into *aBuffer*.

```
unsigned char aBuffer[20];
NSString *myString = @"Test string.";
NSData *myData = [NSData
    dataWithBytes:[myString cString]
   length:[myString cStringLength]];

[myData getBytes:aBuffer];
```

See also: – **bytes:**, – **description**, – **getBytes:length:**, – **getBytes:range:**

getBytes:length:

– (void)**getBytes:**(void *)*buffer* **length:**(unsigned int)*length*

Copies up to *length* bytes from the start of the receiver into *buffer*.

See also: – **bytes:**, – **description**, – **getBytes:,** – **getBytes:range:**

getBytes:range:

– (void)**getBytes:**(void *)*buffer* **range:**(NSRange)*range*

Copies the receiver's contents into *buffer*, from *range* that is within the bytes in the object. If *range* isn't within the receiver's range of bytes, an NSRangeException is raised.

See also: – **bytes:**, – **description**, – **getBytes:,** – **getBytes:length:**

hash

@protocol NSObject
– (unsigned int)**hash**

Returns an unsigned integer that can be used as a table address in a hash table structure. For a data object, **hash** returns the length of the data object. If two data objects are equal (as determined by the **isEqual:** method), they have the same hash value.

See also: – **isEqual:**

initWithBytes:length:

– (id)**initWithBytes:**(const void *)*bytes* **length:**(unsigned int)*length*

Initializes a newly allocated data object by adding to it length bytes of data copied from the buffer bytes. Returns self.

See also: + **dataWithBytes:length:**, – **initWithBytesNoCopy:length:**

initWithBytesNoCopy:length:

– (id)**initWithBytesNoCopy:**(void *)*bytes* **length:**(unsigned int)*length*

Initializes a newly allocated data object by adding to it length bytes of data from the buffer bytes. Returns self.

See also: + **dataWithBytes:length:**, – **initWithBytes:length:**

initWithContentsOfFile:

– (id)**initWithContentsOfFile:**(NSString *)*path*

Initializes a newly allocated data object by reading into it the data from the file specified by *path*. **Returns self.**

See also: + **dataWithContentsOfFile:**, – **initWithContentsOfMappedFile:**

initWithContentsOfMappedFile:

– (id)**initWithContentsOfMappedFile:**(NSString *)*path*

Initializes a newly allocated data object by reading into it the mapped file specified by *path*. **Returns self.**

See also: + **dataWithContentsOfMappedFile:**, – **initWithContentsOfFile:**

initWithData:

– (id)**initWithData:**(NSData *)*data*

Initializes a newly allocated data object by placing in it the contents of another data object, *data*. **Returns self.**

initWithStream:

– (id)**initWithStream:**(NXStream *)*stream*

Initializes a newly allocated data object by placing in it the contents of *stream*. **Returns self.**

isEqual:

@protocol NSObject
– (BOOL)**isEqual:**(id)*anObject*

Returns YES if the receiver and *anObject* are equal; otherwise returns NO. A YES return value indicates that the receiver and *anObject* are both instances of classes that inherit from NSData and that both contain the same data (as determined by the **isEqualToData:** method).

See also: – **isEqualToData:**

isEqualToData:

– (BOOL)**isEqualToData:**(NSData *)*otherData*

Compares the receiving data object to *otherData*. If the contents of *otherData* are equal to the contents of the receiver, this method returns YES. If not, it returns NO. Two data objects are equal if they hold the same number of bytes, and if the bytes at the same position in the objects are the same.

See also: – **isEqual:**

length

– (unsigned int)**length**

Returns the number of bytes contained in the receiver.

subdataWithRange:

– (NSData *)**subdataWithRange:**(NSRange)*range*

Returns a data object containing a copy of the receiver’s bytes that fall within the limits specified by *range*. If *range* isn’t within the receiver’s range of bytes, an NSRangeException is raised.

For example, the following code excerpt initializes a data object, *data2*, to contain a sub-range of *data1*:

```
NSString *myString = @"ABCDEFG";
NSRange range = {2, 4};
NSData *data1, *data2;

data1 = [NSData dataWithBytes:[myString cString]
   length:[myString cStringLength]];

data2 = [data1 subdataWithRange:range];
```

The result of this excerpt is that *data2* contains “CDEF**”.**

writeToFile:atomically:

– (BOOL)**writeToFile:**(NSString *)*path* **atomically:**(BOOL)*flag*

Writes the bytes in the receiver to the file specified by *path*. If *flag* is YES, the data is written to a backup file and then, assuming no errors occur, the backup file is renamed to the specified file name. Otherwise, the data is written directly to the specified file.

If *path* contains a tilde (~) character, you must expand it with **stringByExpandingTildeInPath:** before invoking this method.

YES is returned if the operation succeeded, otherwise NO is returned.

NSMutableData

Inherits From:	NSData : NSObject
Conforms To:	NSCoding NSCopying NSMutableCopying (NSData) NSObject (NSObject)
Declared In:	Foundation/NSData.h Foundation/NSSerialization.h

Class at a Glance

Purpose

An NSMutableData object stores mutable data in the form of bytes.

Principal Attributes

- A count of the number of bytes in the mutable data object.
- The sequence of bytes contained in the mutable data object.

Creation

+ dataWithCapacity:	Returns an NSMutableData with enough allocated memory to hold a specified number of bytes.
+ dataWithLength:	Returns an NSMutableData that contains a specified number of zero-filled bytes.

Commonly Used Methods

– mutableBytes	A pointer to the bytes in the NSMutableData object.
– replaceBytesInRange:withBytes:	Replaces a range of bytes in the NSMutableData object.

Primitive Methods

– mutableBytes
– setLength:

lass Description

The NSMutableData class declares the programmatic interface to an object that contains modifiable data in the form of bytes. NSMutableData's two primitive methods—**mutableBytes** and **setLength:**—provide the basis for all of the other methods in its interface. The **mutableBytes** method returns a pointer for writing into the bytes contained in the mutable data object. **setLength:** allows you to truncate or extend the length of a mutable data object.

increaseLengthBy: also allows you to change the length of a mutable data object.

The **appendBytes:length:** and **appendData:** methods let you append bytes or the contents of another data object to a mutable data object. You can replace a range of bytes in a mutable data object with zeros (using the **resetBytesInRange:** method), or with different bytes (using the **replaceBytesInRange:withBytes:** method).

lethod Types

Creating an NSMutableData	+ allocWithZone: + dataWithCapacity: + dataWithLength: – initWithCapacity: – initWithLength:
Adjusting capacity	– increaseLengthBy: – setLength:
Accessing data	– mutableBytes
Adding data	– appendBytes:length: – appendData:
Serializing data	– serializeAlignedBytesLength: – serializeDataAt:ofObjCType:context: – serializeInt: – serializeInt:atIndex: – serializeInts:count: – serializeInts:count:atIndex:
Modifying data	– replaceBytesInRange:withBytes: – resetBytesInRange: – setData:

lass Methods

allocWithZone

+ (id)**allocWithZone:**(NSZone *)*zone*

Creates and returns an uninitialized data object in the specified zone. If the receiver is the NSMutableData class object, an instance of an appropriate subclass is returned; otherwise, an object of the receiver's class is returned.

Typically, you create objects using the **data...** class methods, not the **alloc...** and **init...** methods. Note that it's your responsibility to release objects created with the **alloc...** methods.

dataWithCapacity:

+ (id)**dataWithCapacity:**(unsigned int)*aNumItems*

Creates and returns an NSMutableData object, initially allocating enough memory to hold *aNumItems* objects. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity.

Note: **dataWithCapacity:** doesn't necessarily allocate its memory at the time of method invocation. When it does allocate its memory, though, it initially allocates the specified amount.

See also: – **dataWithLength:**, – **initWithCapacity:**, – **initWithLength:**

dataWithLength:

+ (id)**dataWithLength:**(unsigned int)*length*

Creates an autoreleased, mutable data object of *length* bytes, filled with zeros.

See also: – **dataWithCapacity:**, – **initWithCapacity:**, – **initWithLength:**

nstance Methods

appendBytes:length:

– (void)**appendBytes:**(const void *)*bytes* **length:**(unsigned int)*length*

Appends *length* bytes to the receiver from the buffer *bytes*.

This excerpt copies the bytes in *data2* into *aBuffer*, and then appends *aBuffer* to *data1*.

```
NSMutableData *data1, *data2;
NSString *firstString =  @"ABCD";
NSString *secondString = @"EFGH";
```

```
unsigned char *aBuffer;
unsigned len;

data1 = [NSMutableData
    dataWithBytes:[firstString cString]
   length:[firstString cStringLength]];
data2 = [NSMutableData
    dataWithBytes:[secondString cString]
   length:[secondString cStringLength]];

len = [data2 length];
aBuffer = malloc(len);

[data2 getBytes:aBuffer];
[data1 appendBytes:aBuffer length:len];
```

The final value of *data1* is the series of ASCII characters “ABCDEFGH”.

See also: – **appendData:**

appendData:

– (void)**appendData:**(NSData *)*otherData*

Appends the contents of a data object *otherData* to the receiver.

See also: – **appendBytes:length:**

increaseLengthBy:

– (void)**increaseLengthBy:**(unsigned int)*extraLength*

Increases the length of the receiver by *extraLength*. The additional bytes are all set to zero.

See also: – **setLength:**

initWithCapacity:

– (id)**initWithCapacity:**(unsigned int)*capacity*

Initializes a newly allocated mutable data object, giving it enough memory to hold *capacity* bytes. Sets the length of the data object to 0. Returns **self**.

See also: – **dataWithCapacity:, – initWithLength:**

initWithLength:

– (id)**initWithLength:**(unsigned int)*length*

Initializes a newly allocated mutable data object, giving it enough memory to hold *length* bytes. Fills the object with zeros up to *length*. Returns **self**.

See also: – **dataWithCapacity:**, **– dataWithLength:**, **– initWithCapacity:**

mutableBytes

– (void *)**mutableBytes**

Returns a pointer to the receiver’s data.

In the following code example, **mutableBytes** is used to return a pointer to the bytes in *data2*. The bytes in *data2* are then overwritten with the contents of *data1*.

```
NSMutableData *data1, *data2;
NSString *myString = @"string for data1";
NSString *yourString = @"string for data2";
unsigned char *firstBuffer, secondBuffer[20];

/* initialize data1, data2, and secondBuffer... */
data1 = [NSMutableData dataWithBytes:[myString cString]
    length:[myString length]];
data2 = [NSMutableData dataWithBytes:[yourString cString]
    length:[yourString length]];

[data2 getBytes:secondBuffer];
NSLog(@"data2 before: \"%s\"\n", (char *)secondBuffer);
firstBuffer = [data2 mutableBytes];
[data1 getBytes:firstBuffer];
NSLog(@"data1: \"%s\"\n", (char *)firstBuffer);
[data2 getBytes:secondBuffer];
NSLog(@"data2 after: \"%s\"\n", (char *)secondBuffer);
```

This is the output from the above code example:

```
Oct  3 15:59:51 [1113] data2 before: "string for data2"
Oct  3 15:59:51 [1113] data1: "string for data1"
Oct  3 15:59:51 [1113] data2 after: "string for data1"
```

replaceBytesInRange:withBytes:

– (void)**replaceBytesInRange:**(NSRange)*range* **withBytes:**(const void *)*bytes*

Specifies a range within the contents of a mutable data object to be replaced by *bytes*. If the location of *range* isn’t within the receiver’s range of bytes, an NSRangeException is raised. The receiver is resized to accomodate the new bytes, if necessary.

In the following code excerpt, a range of bytes in *data1* is replaced by the bytes in *data2*.

```
NSMutableData *data1, *data2;
NSString *myString = @"Liz and John";
NSString *yourString = @"Larry";
unsigned len;
unsigned char *aBuffer;
NSRange range = {8, [yourString cStringLength]};

data1 = [NSMutableData
    dataWithBytes:[myString cString]
    length:[myString cStringLength]];

data2 = [NSMutableData
    dataWithBytes:[yourString cString]
    length:[yourString cStringLength]];

len = [data2 length];
aBuffer = malloc(len);
[data2 getBytes:aBuffer];
[data1 replaceBytesInRange:range withBytes:aBuffer];
```

The contents of *data1* changes from “Liz and John” to “Liz and Larry.”

See also: – **resetBytesInRange:**

resetBytesInRange:

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

Specifies a range within the contents of a mutable data object to be replaced by zeros. If the location of *range* isn’t within the receiver’s range of bytes, an NSRangeException is raised. The receiver is resized to accomodate the new bytes, if necessary.

See also: – **replaceBytesInRange:withBytes:**

serializeAlignedBytesLength:

– (void)**serializeAlignedBytesLength:**(unsigned int)*length*

Appends the bytes of *length* to the end of the receiver. This method may add extra filler bytes to increase the efficiency of deserializing subsequent data. Use of this method is optional; you can invoke **serializeInt:** instead. However, if you use this method, you should match its use by invoking **deserializeAlignedBytesLengthAtCursor:** to read the bytes of *length* later.

serializeDataAt:ofObjCType:context:

– (void)**serializeDataAt:**(const void *)*data*
    ofObjCType:(const char *)*type*
    context:(id <NSObjCTypeSerializationCallBack>)*callback*

Appends a sequence of bytes, specified by *data*, to the receiver. The bytes are formatted according to the Objective-C type code given in *type*. If *type* specifies an object, *callback* is used to serialize the object pointed to by *data*; in such a case, *callback* must itself be an object that conforms to the NSObjCTypeSerializationCallBack protocol. If *type* does not specify an object, *callback* can be **nil**.

For informationon on creating an Objective-C type code suitable for *type*, see the description of the **@encode()** compiler directive in *Object-Oriented Programming and the Objective-C Language*.

See also: **– deserializeDataAt:ofObjCType:atCursor:context:** (NSData)

serializeInt:

– (void)**serializeInt:**(int)*value*

Appends the bytes of *value* to the end of the receiver.

See also: – **serializeAlignedBytesLength:**

serializeInt:atIndex:

– (void)**serializeInt:**(int)*value* **atIndex:**(unsigned int)*index*

Replaces the bytes of an integer at location *index* in the receiver with the bytes of *value*.

See also: **– replaceBytesInRange:withBytes:**

serializeInts:count:

– (void)**serializeInts:**(int *)*intBuffer* **count:**(unsigned int)*numInts*

Appends the bytes of *numInts* integers in *intBuffer* to the receiver.

See also: **– serializeInt:**

serializeInts:count:atIndex:

– (void)**serializeInts:**(int *)*intBuffer*
    count:(unsigned int)*numInts*
    atIndex:(unsigned int)*index*

Replaces the bytes of *numInts* integers currently in the receiver with *numInts* integers in *intBuffer*.

See also: **– replaceBytesInRange:withBytes:**

setData:

– (void)**setData:**(NSData *)*aData*

Uses **replaceBytesInRange:withBytes:** to replace the entire contents of the receiver with the contents of *aData*.

setLength:

– (void)**setLength:**(unsigned int)*length*

Extends or truncates a mutable data object to *length*. If the mutable data object is extended, the additional bytes are filled with zero.

See also: **– increaseLengthBy:**