NSDictionary Class Cluster

Class Cluster Description

The NSDictionary and NSMutableDictionary classes declare the programmatic interface for objects that store associations of keys and values. You use these classes when you need a convenient and efficient way to retrieve data associated with an arbitrary key.

Because of the nature of class clusters, the objects you create with this interface are not actual instances of the NSDictionary or NSMutableDictionary classes. Rather, the instances belong to one of their private subclasses. (For convenience, we use the term *dictionary object* to refer to any one of these instances without specifying its exact class membership.) Although a dictionary object's class is private, its interface is public, as declared by these abstract superclasses, NSDictionary and NSMutableDictionary. (See "Class Clusters" in the introduction to the Foundation Kit for more information on class clusters and creating subclasses within a cluster.)

A key-value pair within a dictionary object is called an *entry*. Each entry consists of an NSString object that represents the key and another object (of any class) which is that key's value. You establish an immutable dictionary's entries when it's created, and thereafter the entries can't be modified. A mutable dictionary allows the addition and deletion of entries at any time, automatically allocating memory as needed.

Internally, a dictionary object uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key. However, the methods defined in this cluster insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys. The methods described below take key values directly, not their hashed form.

Generally, you instantiate a dictionary object by sending one of the **dictionary...** messages to either the NSDictionary or NSMutableDictionary class object. These methods return a dictionary object containing the associations specified as arguments to the method. Methods that add entries to dictionaries—whether as part of initialization (for all dictionary objects) or during modification (for mutable dictionary objects)—copy each key argument and add the copies to the dictionary. Each corresponding value object receives a **retain** message to ensure that it won't be deallocated before the dictionary is through with it.

The dictionary classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert a dictionary of one type to the other.

▶ NSDictionary

Inherits From:	NSObject
Conforms To:	NSCopying
	NSMutableCopying
	NSObject (NSObject)
Declared In:	foundation/NSDictionary.h

Class Description

The NSDictionary class declares the programmatic interface to objects that manage immutable associations of keys and values. NSDictionary's three primitive methods—**count**, **objectForKey:**, and **keyEnumerator**—provide the basis for all the other methods in its interface. The **count** method returns the number of entries in the dictionary object, **objectForKey:** returns the value associated with the given key, and **keyEnumerator** returns an object that lets you step through the keys in the dictionary.

The other methods declared here operate by invoking one or more of these primitives. The non-primitive methods provide convenient ways of accessing multiple entries at once. The **description...** and **writeToFile:atomically:** methods cause a dictionary object to write a representation of itself to a memory stream or a file, respectively.

Note: If you use the convenience methods **fileModificationDate**, **filePosixPermissions**, **fileOwner**, **fileSize**, and **fileType** (which are declared as a category on NSDictionary in NSFileManager.h), be sure to import **Foundation/NSFileManager.h**.

Instance Variables

None declared in this class.

Adopted Protocols

NSCopying	– copy
	– copyWithZone:

NSMutableCopying	– mutableCopy
	– mutableCopyWithZone:

Method Types

Allocating and initializing	+ allocWithZone:
	+ dictionary
	+ dictionaryWithObjects:forKeys:
	+ dictionaryWithObjects:forKeys:count:
	– initWithContentsOfFile:
	– initWithDictionary:
	– initWithObjects:forKeys:count:
Counting entries	– count
Accessing keys and values	– allKeys
	– allKeysForObject:
	– allValues
	– description
	– descriptionInStringsFileFormat
	– descriptionWithIndent:
	– keyEnumerator
	– objectEnumerator
	– objectForKey:
Comparing dictionaries	– isEqualToDictionary:
Storing dictionaries	– writeToFile:atomically:

Class Methods

allocWithZone:

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

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

Typically, you create dictionary objects using the **dictionary...** class methods, not the **alloc...** and **init...** methods. Note that it's your responsibility to release (with either **release** or **autorelease**) those objects created with the **alloc...** methods.

See also: + **dictionary**, + **dictionaryWithObjects:forKeys:count:**

dictionary

+ **dictionary**

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

See also: + **dictionaryWithObjects:forKeys:count:**

dictionaryWithObjects:forKeys:

+ **dictionaryWithObjects:**(NSArray *)*objects* **forKeys:**(NSArray *)*keys*

Creates and returns a dictionary object containing the keys and objects from the *keys* and *objects* arrays. The objects are associated with keys taken from the *keys* array.

See also: + **dictionary,** + **dictionaryWithObjects:forKeys:count:**

dictionaryWithObjects:forKeys:count:

+ **dictionaryWithObjects:**(id *)*objects*
    forKeys:(NSString **)*keys*
    count:(unsigned)*count*

Creates and returns a dictionary object containing *count* objects from the *objects* array. The objects are associated with keys taken from the *keys* array. For example, this code excerpt creates a dictionary that associates the alphabetic characters with their ASCII values:

```
static const int N_ENTRIES = 26;
NSDictionary *asciiDict;
NSString *keyArray[N_ENTRIES];
NSNumber *valueArray[N_ENTRIES];
int i;

for (i = 0; i < N_ENTRIES; i++) {
    char charValue = 'a' + i;
    keyArray[i] = [NSString stringWithFormat:@"%c", charValue];
    valueArray[i] = [NSNumber numberWithChar:charValue];
}
asciiDict = [NSDictionary dictionaryWithObjects:(id *)valueArray
    forKeys:keyArray count:N_ENTRIES];
```

See also: + **dictionary,** + **dictionaryWithObjects:forKeys:**

Instance Methods

allKeys

    – (NSArray *)**allKeys**

Returns an array containing the dictionary's keys or an empty array if the dictionary has no entries. The elements of the array are NSString objects, and their order isn't defined. This method invokes **keyEnumerator** as part of its implementation.

See also: – **allValues**, – **allKeysForObject:**, – **keyEnumerator**

allKeysForObject:

    – (NSArray *)**allKeysForObject:***anObject*

Finds all occurrences of the value *anObject* in the dictionary and returns an array with the corresponding keys. The array contains NSString objects representing these keys. Each object in the dictionary is sent an **isEqual:** message to determine if it's equal to *anObject*. If no object matching *anObject* is found, this method returns **nil**.

Since this method must traverse the entire dictionary, it is slow, with an execution time that's dependent on the number of records in the dictionary.

See also: – **allKeys**, – **keyEnumerator**

allValues

    – (NSArray *)**allValues**

Returns an array containing the dictionary's values, or an empty array if the dictionary has no entries. The order of the values in the array isn't defined. This method invokes **objectEnumerator** as part of its implementation.

See also: – **allKeys**, – **objectEnumerator**

count

    – (unsigned)**count**

Returns the number of entries in the dictionary.

description

   – (NSString *)**description**

Returns a string that represents the contents of the receiver.

descriptionInStringsFileFormat

   – (NSString *)**descriptionInStringsFileFormat**

Returns a string that represents the contents of the receiver. Key-value pairs are represented as appropriate for use in "**.strings**" files.

descriptionWithIndent:

   – (NSString *)**descriptionWithIndent:**(unsigned)*level*

Returns a string that represents the contents of the receiver. Key-value pairs are represented as appropriate for use in "**.strings**" files. Elements are indented from the left margin by *level* + 1 multiples of four spaces, to make the output more readable.

description

   – (NSString *)**description**

Returns a string that represents the contents of the receiver.

hash

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

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

See also: – **isEqual:**

initWithContentsOfFile:

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

Initializes a newly allocated dictionary object using the keys and values found in *path*. *path* can be a full or relative pathname; the file that it names must contain a string representation of a dictionary object, such as that produced by the **writeToFile:atomically:** method.

After initializing the receiver, this method returns **self**. However, if the new instance can't be initialized (either because of a file error or because the contents of the file is an invalid representation of a dictionary object), it's deallocated and **nil** is returned.

See also: – **descriptionInStringsFileFormat**, – **writeToFile:atomically:**

initWithDictionary:

   – **initWithDictionary:**(NSDictionary *)*otherDictionary*

Initializes a newly allocated dictionary object by placing in it the keys and values contained in *otherDictionary*. Returns **self**.

See also: – **initWithContentsOfFile:, – initWithObjects:forKeys:count:**

initWithObjects:forKeys:count:

   – **initWithObjects:**(id *)*objects*
        forKeys:(NSString **)*keys*
        count:(unsigned)*count*

Initializes a newly allocated dictionary object with *count* entries. This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. Each value object receives a **retain** message before being added to the dictionary. In contrast, each key object is copied, and the copy is added to the dictionary. An NSInvalidArgumentException error is raised if a key or value object is **nil**.

See also: – **initWithDictionary:**

isEqual:

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

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

See also: – **isEqualToDictionary:**

isEqualToDictionary:

– (BOOL)**isEqualToDictionary:**(NSDictionary *)*otherDictionary*

Compares the receiving dictionary object to ***otherDictionary***. If the contents of ***otherDictionary*** are equal to the contents of the receiver, this method returns YES. If not, it returns NO.

Two dictionaries have equal contents if they each hold the same number of entries and, for a given key, the corresponding value objects in each dictionary satisfy the **isEqual:** test.

See also: – **isEqual:** (NSObject protocol)

keyEnumerator

– (NSEnumerator *)**keyEnumerator**

Returns an enumerator object that lets you access each key in the dictionary:

```
NSEnumerator *enumerator = [myDictionary keyEnumerator];
id key;
while (key = [enumerator nextObject]) {
    /* code that uses the returned key */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the **allKeys** method to create a "snapshot" of the dictionary's keys. Then use this snapshot to traverse the entries, modifying them along the way.

Note that the **objectEnumerator** method provides a convenient way to access each value in the dictionary.

See also: – **objectEnumerator**, – **nextObject** (NSEnumerator protocol)

objectEnumerator

　　– (NSEnumerator *)**objectEnumerator**

Returns an enumerator object that lets you access each value in the dictionary:

```
NSEnumerator *enumerator = [myDictionary objectEnumerator];
id value;
while (value = [enumerator nextObject]) {
    /* code that acts on the dictionary's values */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the **allValues** method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

See also: – **keyEnumerator**, – **nextObject** (NSEnumerator protocol)

objectForKey:

　　– **objectForKey:**(NSString *)*aKey*

Returns an entry's value given its key, or nil if no value is associated with *aKey*. This method is fast, with an execution time that's independent of the number of entries in the dictionary.

See also: – **allKeys**, – **allValues**

writeToFile:atomically:

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

Writes a textual description of the contents of the dictionary to *path*. *filename* can be a full or relative path name.

If *flag* is YES, the dictionary is written to an auxiliary file having the name *path~*, and then the auxiliary file is renamed to *path*. If flag is NO, the dictionary is written directly to *path*. The YES option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

This method returns YES if the file is written successfully, and NO otherwise.

See also: – **descriptionInStringsFileFormat,** – **initWithContentsOfFile:**

▶ **NSMutableDictionary**

Inherits From:	NSDictionary : NSObject
Conforms To:	NSCopying
	NSMutableCopying
	NSObject (NSObject)
Declared In:	foundation/NSDictionary.h

Class Description

The NSMutableDictionary class declares the programmatic interface to objects that manage mutable associations of keys and values. With its two efficient primitive methods—**setObject:forKey:** and **removeObject:forKey:**—this class adds modification operations to the basic operations it inherits from NSDictionary.

The other methods declared here operate by invoking one or both of these primitives. The non-primitive methods provide convenient ways of adding or removing multiple entries at a time.

When an entry is removed from a mutable dictionary, the key and value objects that make up the entry receive **release** messages. If there are no further references to the objects, they're deallocated. Note that if your program keeps a reference to such an object, the reference will become invalid unless you remember to send the object a **retain** message before it's removed from the dictionary. For example, the third statement below would have result in a run-time error, if the **retain** message in the first statement had not been included:

```
id anObject = [[aDictionary objectForKey:theKey] retain];
[aDictionary removeObjectForKey:theKey];
[anObject someMessage];
```

Instance Variables

None declared in this class.

Method Types

Allocating and initializing	+ allocWithZone:
	+ dictionaryWithCapacity:
	– initWithCapacity:
Adding and removing entries	– addEntriesFromDictionary:
	– removeAllObjects
	– removeObjectForKey:
	– removeObjectsForKeys:
	– setObject:forKey:

Class Methods

allocWithZone:

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

Creates and returns an uninitialized mutable dictionary object in the specified zone. If the receiver is the NSMutableDictionary class object, an instance of a mutable private subclass is returned; otherwise, an object of the receiver's class is returned.

Typically, you create dictionary objects using the **dictionary...** class methods, not the **alloc...** and **init...** methods. Note that it's your responsibility to release (with either **release** or **autorelease**) those objects created with the **alloc...** methods.

See also: + **dictionary** (NSDictionary), + **dictionaryWithCapacity:**, + **dictionaryWithObjects:forKeys:count:** (NSDictionary)

dictionaryWithCapacity:

+ **dictionaryWithCapacity:**(unsigned)*numItems*

Creates and returns an mutable dictionary object, giving it enough allocated memory to hold *numEntries* entries. Mutable dictionary objects allocate additional memory as needed, so *numEntries* simply establishes the object's initial capacity.

See also: + **dictionary** (NSDictionary), + **dictionaryWithObjects:forKeys:count:** (NSDictionary), – **initWithCapacity:**

Instance Methods

addEntriesFromDictionary:

– (void)**addEntriesFromDictionary:**(NSDictionary *)*otherDictionary*

Adds the entries from *otherDictionary* to the receiver. Each value object from *otherDictionary* is sent a **retain** message before being added to the receiver. In contrast, each key object is copied, and the copy is added to the receiver.

If both dictionaries contain the same key, the receiver's previous value object for that key is sent a **release** message and the new value object takes its place.

This method invokes **setObject:forKey:** as part of its implementation.

See also: – **setObject:forKey:**

copyWithZone:

@protocol NSObject
– **copyWithZone:**(NSZone *)*aZone*

Creates and returns an immutable copy of the receiver. The new dictionary object contains copies of the receiver's keys and values.

Note that it's your responsibility to release a dictionary object created in this way.

See also: – **mutableCopyWithZone:** (NSObject protocol)

initWithCapacity:

– **initWithCapacity:**(unsigned)*numItems*

Initializes a newly allocated mutable dictionary object, giving it enough allocated memory to hold *numItems* entries. Mutable dictionary objects allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity. Returns **self**.

See also: + **dictionaryWithCapacity:**

removeAllObjects

– (void)**removeAllObjects**

Empties the dictionary of its entries. Each key and corresponding value object is sent a **release** message.

See also: – **removeObjectForKey:**, – **removeObjectsForKeys:**

removeObjectForKey:

– (void)**removeObjectForKey:**(NSString *)*aKey*

Removes *aKey* and its associated value object from the dictionary.

For example, assume you have a archived dictionary that records the call letters and associated frequencies of radio stations. To remove an entry of a defunct station, you could execute this code:

```
NSMutableDictionary *stations = nil;

stations = [[NSMutableDictionary alloc]
    initWithContentsOfFile:theArchiveFile];
if (stations) {
    [stations removeObjectForKey:@"KIKT"];
}
```

This method is fast, with an execution time that's independent of the number of entries in the dictionary.

See also: – **removeAllObjects**, – **removeObjectsForKeys:**

removeObjectsForKeys:

– (void)**removeObjectsForKeys:**(NSArray *)*keyArray*

Removes one or more entries from the receiving dictionary. The entries are identified by the keys in *keyArray*. This method is slow, with an execution time that's dependent on the number of entries in the dictionary.

See also: – **removeAllObjects**, – **removeObjectForKey:**

setObject:forKey:

– (void)**setObject:***anObject* **forKey**:(NSString *)*aKey*

Adds an entry to the receiver, consisting of a*Key* and its corresponding value object *anObject*. The value object receives a **retain** message before being added to the dictionary. In contrast, the key is copied, and the copy is added to the dictionary. An NSInvalidArgumentException error is raised if the key or value object is **nil**.

If *aKey* already exists in the receiving dictionary, the receiver's previous value object for that key is sent a **release** message and *anObject* takes its place.

This method is fast, with an execution time that's independent of the number of entries in the dictionary.

See also: – **removeObjectForKey:**