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* to refer to any one of these instances without specifying its exact class membership.) Although a dictionary's class is private, its interface is public, as declared by these abstract superclasses, NSDictionary and NSMutableDictionary.

A key-value pair within a dictionary is called an *entry.* Each entry consists of one object that represents the key, and a second object which is that key's value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by **isEqual:**).

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 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 keys directly, not their hashed form.

Generally, you create a temporary dictionary by sending one of the **dictionary...** messages to either the NSDictionary or NSMutableDictionary class object. These methods return a dictionary containing the associations specified as arguments to the method. Methods that add entries to dictionaries—whether as part of initialization (for all dictionaries) or during modification (for mutable dictionaries)—copy each key argument (keys must conform to the NSCopying protocol) 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:	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Declared In:	Foundation/NSDictionary.h

Class at a Glance

Purpose

An NSDictionary object stores an immutable set of entries.

Principal Attributes

- A count of the number of entries in the dictionary.
- The set of keys contained in the dictionary.
- The objects that correspond to the keys in the dictionary.

Creation

+ dictionary	Returns an empty dictionary.
+ dictionaryWithContentsOfFile:	Returns a dictionary initialized from the property list stored in the specified file.
+ dictionaryWithDictionary:	Returns a dictionary initialized from an existing dictionary.
+ dictionaryWithObject:forKey:	Returns a dictionary initialized with a single object and key.
+ dictionaryWithObjects:forKeys:	Returns a dictionary of the specified objects and their keys.
+ dictionaryWithObjects:forKeys:count:	Returns a dictionary of the specified objects and their keys.
+ dictionaryWithObjectsAndKeys:	Returns a dictionary of the specified objects and their keys.

Commonly Used Methods

– count	Returns the number of objects currently in the dictionary.
– objectForKey:	Returns the object that corresponds to the specified key.
– keyEnumerator	Returns an enumerator object that lets you access each key in the dictionary.

Primitive Methods

– count
– objectForKey:
– keyEnumerator

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 of the other methods in its interface. The **count** method returns the number of entries in the dictionary. **objectForKey:** returns the value associated with a given key. **keyEnumerator** returns an object that lets you iterate through each of 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 to write a representation of itself to a string or to a file, respectively.

Adopted Protocols

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

Method Types

Creating dictionaries	+ allocWithZone: + dictionary + dictionaryWithContentsOfFile: + dictionaryWithDictionary: + dictionaryWithObject:forKey: + dictionaryWithObjects:forKeys: + dictionaryWithObjects:forKeys:count: + dictionaryWithObjectsAndKeys: – initWithContentsOfFile: – initWithDictionary: – initWithObjects:forKeys: – initWithObjects:forKeys:count: – initWithObjectsAndKeys:
Counting entries	– count

Accessing keys and values	– allKeys – allKeysForObject: – allValues – description – descriptionInStringsFileFormat – descriptionWithLocale: – descriptionWithLocale:indent: – keyEnumerator – keysSortedByValueUsingSelector: – objectEnumerator – objectForKey: – objectsForKeys:notFoundMarker:
Comparing dictionaries	– isEqualToDictionary:
Storing dictionaries	– writeToFile:atomically:

Class Methods

allocWithZone:

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

Creates and returns an uninitialized dictionary 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 temporary dictionaries using the **dictionary...** class methods, not the **alloc...** and **init...** methods.

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

dictionary

+ (id)**dictionary**

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

See also: **+ dictionaryWithContentsOfFile:**, **+ dictionaryWithObjects:forKeys:**, **+ dictionaryWithObjects:forKeys:count:**, **+ dictionaryWithObjectsAndKeys:**

dictionaryWithContentsOfFile:

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

Creates and returns a dictionary containing 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, such as that produced by the **writeToFile:atomically:** method.

nil is returned if there's a file error or if the contents of the file is an invalid representation of a dictionary.

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

dictionaryWithDictionary:

+ (id)**dictionaryWithDictionary:**(NSDictionary *)*otherDictionary*

Creates and returns a dictionary containing the keys and values found in *otherDictionary.*

dictionaryWithObject:forKey:

+ (id)**dictionaryWithObjects:**(id)*anObject* **forKey:**(id)*aKey*

Creates and returns a dictionary containing a single object, *anObject*, for a single key, *aKey.*

dictionaryWithObjects:forKeys:

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

Creates and returns a dictionary containing entries constructed from the contents of *objects* and *keys*. 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 is copied (using **copyWithZone:**; keys must conform to the NSCopying protocol), and the copy is added to the dictionary. An NSInvalidArgumentException is raised if *objects* and *keys* don't have the same number of elements.

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

dictionaryWithObjects:forKeys:count:

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

Creates and returns a dictionary 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:(id *)keyArray count:N_ENTRIES];
```

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

dictionaryWithObjectsAndKeys:

+ (id)**dictionaryWithObjectsAndKeys:**(id)*object***,** (id)*key, ...*

Creates and returns a dictionary containing entries constructed from the specified set of objects and keys. **initWithObjectsAndKeys:** takes a variable number of arguments: a null-terminated list of alternating objects and keys. If any key is **nil**, an NSInvalidArgumentException is raised.

This method is similar to **initWithObjects:forKeys:**, differing only in the way key-value pairs are specified.

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

Instance Methods

allKeys

– (NSArray *)**allKeys**

Returns a new array containing the dictionary’s keys or an empty array if the dictionary has no entries. The order of the elements in the array isn’t defined.

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

allKeysForObject:

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

Finds all occurrences of the value *anObject* in the dictionary and returns a new array with the corresponding 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**.

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

allValues

– (NSArray *)**allValues**

Returns a new 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.

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

count

– (unsigned int)**count**

Returns the number of entries in the dictionary.

description

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

Returns a string that represents the contents of the receiver, formatted as a property list. If each key in the dictionary responds to **compare:**, the entries are listed in ascending order, by key. Otherwise, the order in which the entries are listed is undefined.

See also: – **descriptionWithLocale:**, – **descriptionWithLocale:indent:**

descriptionInStringsFileFormat

– (NSString *)**descriptionInStringsFileFormat**

Returns a string that represents the contents of the receiver, formatted in **.strings** file format. The order in which the entries are listed is undefined.

descriptionWithLocale:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*locale*

Returns a string object that represents the contents of the receiver, formatted as a property list. *locale* specifies options used for formatting each of the receiver's keys and values; specify **nil** if you don't want them formatted.

For a description of how *locale* is applied to each element in the receiver, see **descriptionWithLocale:indent:**.

If each key in the dictionary responds to **compare:**, the entries are listed in ascending order, by key. Otherwise, the order in which the entries are listed is undefined.

See also: – **description**, – **descriptionWithLocale:indent:**

descriptionWithLocale:indent:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*locale* **indent:**(unsigned int)*level*

Returns a string object that represents the contents of the receiver, formatted as a property list. Use *locale* to specify options to be passed to the methods that format each of the receiver's keys and values; specify **nil** if you don't want them formatted. *level* allows you to specify a level of indent, to make the output more readable: set level to 0 to use four spaces to indent, or 1 to indent the output with a tab character.

The returned NSString contains the string representations of each of the receiver's entries. **descriptionWithLocale:indent:** obtains the string representation of a given key or value as follows:

- If the object is an NSString, it is used as is.
- If the object responds to **descriptionWithLocale:indent:**, that method is invoked to obtain the object's string representation.
- If the object responds to **descriptionWithLocale:**, that method is invoked to obtain the object's string representation.
- If none of the above conditions are met, the object's string representation is obtained by invoking its **description** method.

If each key in the dictionary responds to **compare:**, the entries are listed in ascending order, by key. Otherwise, the order in which the entries are listed is undefined.

See also: – **description**, – **descriptionWithLocale:**

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 dictionary, **hash** returns the number of entries in the dictionary. If two dictionaries are equal (as determined by the **isEqual:** method), they will have the same hash value.

See also: – **isEqual:**

initWithContentsOfFile:

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

Initializes a newly allocated dictionary 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, 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), it's deallocated and **nil** is returned.

See also: + **dictionaryWithContentsOfFile:**, – **writeToFile:atomically:**

initWithDictionary:

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

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

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

initWithObjects:forKeys:

– (id)**initWithObjects:**(NSArray *)*objects* **forKeys:**(NSArray *)*keys*

Initializes a newly allocated dictionary with entries constructed from the contents of the *objects* and *keys* arrays. 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 (using **copyWithZone:**), and the copy is added to the dictionary. An NSInvalidArgumentException is raised if the *objects* and *keys* arrays do not have the same number of elements.

See also: + **dictionaryWithObjects:forKeys:**, – **initWithContentsOfFile:**, – **initWithDictionary:**, – **initWithObjects:forKeys:count:**, – **initWithObjectsAndKeys:**

initWithObjects:forKeys:count:

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

Initializes a newly allocated dictionary 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 (using **copyWithZone:**), and the copy is added to the dictionary. An NSInvalidArgumentException is raised if a key or value object is **nil**.

See also: + **dictionaryWithObjects:forKeys:count:**, – **initWithContentsOfFile:**, – **initWithDictionary:**, – **initWithObjects:forKeys:**, – **initWithObjectsAndKeys:**

initWithObjectsAndKeys:

– (id)**initWithObjectsAndKeys:**(id)*object,* (id)*key, ...*

Initializes a newly allocated dictionary with entries constructed from the specified set of objects and keys. **initWithObjectsAndKeys:** takes a variable number of arguments: a null-terminated list of alternating objects and keys. If a key is found to be **nil**, an NSInvalidArgumentException is raised.

This method is similar to **initWithObjects:forKeys:**, differing only in the way in which the key-value pairs are specified.

See also: + **dictionaryWithObjectsAndKeys:**, – **initWithContentsOfFile:**, – **initWithDictionary:**, – **initWithObjects:forKeys:**, – **initWithObjects:forKeys:count:**

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 NSDictionary and contain the same data (as determined by the **isEqualToDictionary:** method).

See also: – **isEqualToDictionary:**

isEqualToDictionary:

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

Compares the receiving dictionary 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)

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: – **allKeys**, – **allKeysForObject:**, – **objectEnumerator**, – **nextObject** (NSEnumerator protocol)

NS+

keysSortedByValueUsingSelector:

– (NSArray *)**keysSortedByValueUsingSelector:**(SEL)*comparator*

Returns an array of the dictionary’s keys, in the order they would be in if the dictionary was sorted by its values. Pairs of dictionary values are compared using the comparison method specified by *comparator*; the comparator message is sent to one of the values, and has as its single argument the other value from the dictionary. The comparator method should return NSOrderedAscending if the receiver is smaller than the argument, NSOrderedDescending if the receiver is larger than the argument, and NSOrderedSame if they are equal.

See also: – **allKeys**, **– sortedArrayUsingSelector:** (NSArray)

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:

– (id)**objectForKey:**(id)*aKey*

Returns an entry’s value given its key, or **nil** if no value is associated with *aKey.*

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

objectsForKeys:notFoundMarker:

– (NSArray *)**objectsForKeys:**(NSArray *)*keys* **notFoundMarker:**(id)*anObject*

Returns the set of objects from the receiver that correspond to the specified *keys* as an NSArray. The objects in the returned array and the *keys* array have a one-for-one correspondence, so that the *n*th object in the returned array corresponds to the *n*th key in *keys*. If an object isn’t found in the receiver to correspond to a given key, the marker object, specified by *anObject*, is placed in the corresponding element of the returned array.

writeToFile:atomically:

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

Writes a textual description of the contents of the dictionary to *path*. *path* must be an absolute path name.

If *flag* is YES, the dictionary is written to an auxiliary file, 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: – **initWithContentsOfFile:**

NSMutableDictionary

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

Class at a Glance

Purpose

An NSDictionary object stores a mutable set of entries.

Principal Attributes

- A count of the number of entries in the dictionary
- The set of keys contained in the dictionary
- The objects that correspond to the keys in the dictionary

Creation

+ dictionaryWithCapacity:	Returns an empty dictionary with enough allocated space to hold a specified number of objects

Commonly Used Methods

– removeObjectForKey:	Removes the specified entry from the dictionary.
– removeObjectForKeys:	Removes multiple entries from the dictionary.

Primitive Methods

– setObject:forKey:
– removeObjectForKey:

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 result in a run-time error if *anObject* was not retained before it was removed:

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

[aDictionary removeObjectForKey:theKey];
[anObject someMessage];
```

Method Types

Creating an NSMutableDictionary	+ allocWithZone: + dictionaryWithCapacity: – initWithCapacity:
Adding and removing entries	– addEntriesFromDictionary: – removeAllObjects – removeObjectForKey: – removeObjectsForKeys: – setDictionary: – setObject:forKey:

Class Methods

allocWithZone:

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

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

Typically, you create temporary dictionaries using the **dictionary...** class methods, not the **alloc...** and **init...** methods.

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

dictionaryWithCapacity:

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

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

See also: + **dictionary** (NSDictionary), + **dictionaryWithContentsOfFile:** (NSDictionary), + **dictionaryWithObjects:forKeys:** (NSDictionary), + **dictionaryWithObjects:forKeys:count:** (NSDictionary), + **dictionaryWithObjectsAndKeys:** (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 (using **copyWithZone:**; keys must conform to the NSCopying protocol), 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.

See also: – **setObject:forKey:**

initWithCapacity:

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

Initializes a newly allocated mutable dictionary, allocating enough memory to hold *numItems* entries. Mutable dictionaries 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:**(id)*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 write code similar to the following:

```
NSMutableDictionary *stations = nil;

stations = [[NSMutableDictionary alloc]
   initWithContentsOfFile:theArchiveFile];
[stations removeObjectForKey:@”KIKT”];
```

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

removeObjectsForKeys:

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

Removes one or more entries from the receiver. The entries are identified by the keys in *keyArray.*

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

setDictionary:

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

Sets the receiver to entries in *otherDictionary.* setDictionary does this by removing all entries from the receiver (with **removeAllObjects**) then adding each entry from *otherDictionary* into the receiver.

setObject:forKey:

– (void)**setObject:**(id)*anObject* **forKey**:(id)*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 (using **copyWithZone:**; keys must conform to the NSCopying protocol), and the copy is added to the dictionary. An NSInvalidArgumentException is raised if the key or value object is **nil**.

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

See also: – **removeObjectForKey:**