List

Inherits From:	Object
Declared In:	objc/List.h

Class Description

A List is a collection of objects. The class provides an interface that permits easy manipulation of the collection as a fixed or variable-sized list, a set, or an ordered collection. Lists are implemented as arrays to allow fast random access using an index. Indices start at 0.

A List array contains object **id**s. An object isn't copied when it's added to a List; only its **id** is. There are no empty slots within the array. **nil** objects can't be inserted in a List, and the collection is contracted to close the empty space when an object is removed.

A List grows dynamically when new objects are added. The default mechanism automatically doubles the capacity of the List when it becomes full, thus ensuring an average constant time for insertions, independent of the size of the List.

For manipulating sets of structures that aren't objects, see the Storage class.

Instance Variables

id ***dataPtr**;
unsigned int **numElements**;
unsigned int **maxElements**;

dataPtr	The data managed by the List object (the array of objects).
numElements	The actual number of objects in the array.
maxElements	The total number of objects that can fit in currently allocated memory.

Method Types

Initializing a new List object	– init
	– initCount:
Copying and freeing a List	– copyFromZone:
	– free
Manipulating objects by index	– insertObject:at:
	– addObject:
	– removeObjectAt:
	– removeLastObject
	– replaceObjectAt:with:
	– objectAt:
	– lastObject
	– count
Manipulating objects by **id**	– addObject:
	– addObjectIfAbsent:
	– removeObject:
	– replaceObject:with:
	– indexOf:
Comparing and combining Lists	
	– isEqual:
	– appendList:
Emptying a List	– empty
	– freeObjects
Sending messages to the objects	
	– makeObjectsPerform:
	– makeObjectsPerform:with:
Managing the storage capacity	– capacity
	– setAvailableCapacity:
Archiving	– read:
	– write:

Instance Methods

addObject:

– **addObject:***anObject*

Inserts *anObject* at the end of the List, and returns **self**. However, if *anObject* is **nil**, nothing is inserted and **nil** is returned.

See also: – **insertObject:at:**, – **appendList:**

addObjectIfAbsent:

– **addObjectIfAbsent:***anObject*

Inserts *anObject* at the end of the List and returns **self**, provided that *anObject* isn't already in the List. If *anObject* is in the List, it won't be inserted, but **self** is still returned.

If *anObject* is **nil**, nothing is inserted and **nil** is returned.

See also: – **insertObject:at:**

appendList:

– **appendList:**(List *)*otherList*

Inserts all the objects in *otherList* at the end of the receiving List, and returns **self**. The ordering of the objects is maintained.

See also: – **addObject:**

capacity

– (unsigned int)**capacity**

Returns the maximum number of objects that can be stored in the List without allocating more memory for it. When new memory is allocated, it's taken from the same zone that was specified when the List was created.

See also: – **count**, – **setAvailableCapacity:**

copyFromZone:

– **copyFromZone:**(NXZone *)*zone*

Returns a new List object with the same contents as the receiver. The objects in the List aren't copied; therefore, both Lists contain pointers to the same set of objects. Memory for the new List is allocated from *zone*.

See also: – **copy** (Object)

count

– (unsigned int)**count**

Returns the number of objects currently in the List.

See also: – **capacity**

empty

– **empty**

Empties the List of all its objects without freeing them, and returns **self**. The current capacity of the List isn't changed.

See also: – **freeObjects**

free

– **free**

Deallocates the List object and the memory it allocated for the array of object **id**s. However, the objects themselves aren't freed.

See also: – **freeObjects**

freeObjects

– **freeObjects**

Removes every object from the List, sends each one of them a **free** message, and returns **self**. The List object itself isn't freed and its current capacity isn't altered.

The methods that free the objects shouldn't have the side effect of modifying the List.

See also: – **empty**

indexOf:

– (unsigned int)**indexOf:***anObject*

Returns the index of the first occurrence of *anObject* in the List, or NX_NOT_IN_LIST if *anObject* isn't in the List.

init

– **init**

Initializes the receiver, a new List object, but doesn't allocate any memory for its array of object **id**s. It's initial capacity will be 0. Minimal amounts of memory will be allocated when objects are added to the List. Or an initial capacity can be set, before objects are added, using the **setAvailableCapacity:** method. Returns **self**.

See also: – **initCount:**, – **setAvailableCapacity:**

initCount:

– **initCount:**(unsigned int)*numSlots*

Initializes the receiver, a new List object, by allocating enough memory for it to hold *numSlots* objects. Returns **self**.

This method is the designated initializer for the class. It should be used immediately after memory for the List has been allocated and before any objects have been assigned to it; it shouldn't be used to reinitialize a List that's already in use.

See also: – **capacity**

insertObject:at:

– **insertObject:***anObject* **at:**(unsigned int)*index*

Inserts *anObject* into the List at *index*, moving objects down one slot to make room. If *index* equals the value returned by the **count** method, *anObject* is inserted at the end of the List. However, the insertion fails if *index* is greater than the value returned by **count** or *anObject* is **nil**.

If *anObject* is successfully inserted into the List, this method returns **self**. If not, it returns **nil**.

See also: – **count**, – **addObject:**

isEqual:

– (BOOL)**isEqual:***anObject*

Compares the receiving List to *anObject*. If *anObject* is a List with exactly the same contents as the receiver, this method returns YES. If not, it returns NO.

Two Lists have the same contents if they each hold the same number of objects and the **id**s in each List are identical and occur in the same order.

lastObject

– **lastObject**

Returns the last object in the List, or **nil** if there are no objects in the List. This method doesn't remove the object that's returned.

See also: – **removeLastObject**

makeObjectsPerform:

– **makeObjectsPerform:**(SEL)*aSelector*

Sends an *aSelector* message to each object in the List in reverse order (starting with the last object and continuing backwards through the List to the first object), and returns **self**. The *aSelector* method must be one that takes no arguments. It shouldn't have the side effect of modifying the List.

makeObjectsPerform:with:

– **makeObjectsPerform:**(SEL)*aSelector* **with:***anObject*

Sends an *aSelector* message to each object in the List in reverse order (starting with the last object and continuing backwards through the List to the first object), and returns **self**. The message is sent each time with *anObject* as an argument, so the *aSelector* method must be one that takes a single argument of type **id**. The *aSelector* method shouldn't, as a side effect, modify the List.

objectAt:

– **objectAt:**(unsigned int)*index*

Returns the **id** of the object located at slot *index*, or **nil** if *index* is beyond the end of the List.

See also: – **count**

read:

– **read:**(NXTypedStream *)*stream*

Reads the List and all the objects it contains from the typed stream *stream*.

See also: – **write:**

removeLastObject

– **removeLastObject**

Removes the object occupying the last position in the List and returns it. If there are no objects in the List, this method returns **nil**.

See also: – **lastObject**, – **removeObjectAt:**

removeObject:

– **removeObject:***anObject*

Removes the first occurrence of *anObject* from the List, and returns it. If *anObject* isn't in the List, this method returns **nil**.

The positions of the remaining objects in the List are adjusted so there's no gap.

See also: – **removeLastObject**, – **removeObjectAt:**

removeObjectAt:

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

Removes the object located at *index* and returns it. If there's no object at *index*, this method returns **nil**.

The positions of the remaining objects in the List are adjusted so there's no gap.

See also: – **removeLastObject**, – **removeObject:**

replaceObject:with:

– **replaceObject:***anObject* **with:***newObject*

Replaces the first occurrence of *anObject* in the List with *newObject*, and returns *anObject*. However, if *newObject* is **nil** or *anObject* isn’t in the List, nothing is replaced and **nil** is returned.

See also: – replaceObjectAt:with:

replaceObjectAt:with:

– **replaceObjectAt:**(unsigned int)*index* **with:***newObject*

Returns the object at *index* after replacing it with *newObject*. If there’s no object at *index* or *newObject* is **nil**, nothing is replaced and **nil** is returned.

See also: – replaceObject:with:

setAvailableCapacity:

– **setAvailableCapacity:**(unsigned int)*numSlots*

Sets the storage capacity of the List to at least *numSlots* objects and returns **self**. However, if the List already contains more than *numSlots* objects (if the **count** method returns a number greater than *numSlots*), its capacity is left unchanged and **nil** is returned.

See also: – capacity, **– count**

write:

– **write:**(NXTypedStream *)*stream*

Writes the List, including all the objects it contains, to the typed stream *stream*.

See also: – read: