MiscLinkedList

Inherits From: Object

Declared In: misckit/MiscLinkedList.h

Protocols: NXTransport

Class Description

Warning: This class is still untested. Sean's original methods are known to work, but Don's new methods and the changes haven't been tested yet. Use at your own risk!

The MiscLinkedList is an implementation of a doubly-linked list. This type of data structure is useful for storing sparse data that will be traversed often. Linked lists are very memory efficient when dealing with sparse data, unlike hashing and some array schemes. If fast random access is desired, a linked list may not be suitable for the application. The MiscLinkedList is much faster than the NeXT List object for insertion and deletion, but it suffers in performance when searching for a specific node (random access). Traversing the List from one end to the other is about the same, but slightly faster for the NeXT List object.

The MiscLinkedList can store any type of objectsÐor a mixture of different object types. A single object may be stored in the list in multiple places, as well. Using a MiscLinkedList is simple. First create a new list using the standard +**alloc** and ±**init** methods. Once created, the list has references to both the start and the end of the list as well as a pointer to the ªcurrentº node. The pointer to the current node is akin to a cursor: most operations on the MiscLinkedList will occur at this node or immediately before or after it.

To add objects to a MiscLinkedList, use the ±**insertObjectAfter:** or ±**insertObjectBefore** methods. They will insert the object either before or after the current node in the list. You can also insert a complete MiscLinkedList into another MiscLinkedList. To do this, use one of the ±**insertList:**, ±**insertListAtFront:**, and ±**appendList:** methods. To remove an

object, use ±**deleteObject**.

You can also change the ordering of nodes in the list. A node my be swapped with it's neighbor with either ±**swapWithNext** or ±**swapWithPrevious**. A node may be moved to the front or end of the list by using ±**moveToFirst** or ±**moveToLast**.

The current object may be replaced with a new object by using the ±**setObject:** method. To traverse the list, use ±**goFirst**, ±**goLast**, ±**goNext**, or ±**goPrevious**. Note that many of the methods which traverse the list might alter the current node; this is documented below for every method in which it occurs.

All objects in a MiscLinkedList may be broadcast a messgae with ±**makeObjectsPerform:** and ±**makeObjectsPerform:with:**. You can find out how many objects are in the MiscLinkedList via the ±**count** or ±**getLength** methods. (They are identical.)

There are also several methods which are similiar to the List object. You can compare lists with the ±**isEqual** method. You can replace and remove objects, given their id, with ±**replaceObject:with:** and ±**removeObject:** respectively. You can add objects uniquely with ±**addObjectIfAbsent:**.

For convenience, there are also methods to operate on the front and end of a MiscLinkedList: ±**firstObject**, ±**removeFirstObject**, ±**lastObject**, and ±**removeLastObject**. You can also use ±**addObject:** to append objects to a MiscLinkedList.

A MiscLinkedList implements the NXTransport protocol, allowing it to be copied over a Distributed Objects connection. It also understands ±**read:** and ±**write:** which allows it to be archived to a stream.

Internally, the MiscLinkedList uses a special class, the MiscLinkedListNode, to keep track of the actual list. Each node contains a pointer to the next and previous node and a pointer to the object it contains. Normally you won't need to worry about these nodes; their use allows any kind of object to be added to the list without it needing to know about linked lists and it allows an object to appear in the list multiple times. (This is better than requiring an object to have pointers to the next and previous object itself.) If you need to add special capabilities to a MiscLinkedList, however, you may wish to subclass the MiscLinkedListNode to facilitate this. See the documentation for that class if you need more information about this. The MiscLinkedList may be initialized to deal with a different node class by using the ±**initForClass:** method. You can access the current node object, rather than the object it contains, with the ±**getNode** method. Under normal circumstances, this shouldn't ever be necessary.

When you are finished with the MiscLinkedList, you can free it with ±**free**, which frees the MiscLinkedList and it's nodes, but not the objects it contains. If you wish to free the objects, first send a ±**freeObjects** message. The ±**empty** method will empty out a MiscLinkedList without freeing the objects it contains (but it will free the nodes).

Instance Variables

id **current_node**;
id **first_node**;
id **last_node**;
id **node_class**;
int **node_length**;

current_node	Pointer to the ªcurrentº node in the list. Most actions are performed in the vicinity of the node this points to.
first_node	Pointer to the last node of the list.
last_node	Pointer to the last node of the list.
node_class	The class object used to create new nodes; this should ideally be either a MiscLinkedListNode subclass or respond to all the same methods as the MiscLinkedListNode, since MiscLinkedList doesn't check ±**respondsTo:**.
node_length	The number of nodes (objects) in the MiscLinkedList

Method Types

Initializing and freeing	± empty ± freeNodes ± freeObjects ± init ± initForClass:
Getting size	± count ± getLength
Changing which is the current node	± firstObject ± goFirst ± goLast ± goNext ± goPrevious

	± lastObject
Manipulating the current node	± getNode ± getObject ± moveToFirst ± moveToLast ± setObject: ± swapWithNext ± swapWithPrevious
Adding and deleting objects	± addObject: ± addObjectIfAbsent: ± deleteObject ± insertObjectAfter: ± insertObjectBefore: ± removeLastObject ± removeFirstObject ± removeObject: ± replaceObject:
Adding an entire MiscLinkedList	± appendList: ± insertList: ± insertListAtFront:
Comparing MiscLinkedLists	± isEqual:
Sending messages to the objects	± makeObjectsPerform: ± makeObjectsPerform:with:
Archiving a MiscLinkedList	± read: ± write:

Instance Methods

addObject:
- addObject:*anObject*

Adds *anObject* to the end of the list. Returns *self*.

See also: **-addObjectIfAbsent:** and ±**removeObject:**

addObjectIfAbsent:
- addObjectIfAbsent:*anObject*

Adds *anObject* to the end of the list if *anObject* is not already anywhere in the list. Returns *self*.

See also: **-addObject:** and ±**removeObject:**

appendList:
- appendList:*this_list*

Insert the list *this_list* after the last node of the receiving list. The objects that were in *this_list* are left in the same order when inserted into the receiver. The list *this_list* is left unchanged. Returns *self*.

See also: **-insertList:** and ±**insertListAtFront:**

count
- (unsigned int)**count**

Returns the number of objects (nodes) in the list.

See also: **-getLength**

deleteObject
- deleteObject

Deletes the current node and object. Returns *self*.

See also: ±**removeFirstObject**, ±**removeLastObject**, and ±**removeObject:**

empty
- empty

Empties the list of all objects and frees all the nodes that made up the list. The objects in the list are not freed.

See also: ±**free**, ±**freeObjects**, and **-freeNodes**

firstObject
- firstObject

Returns the first object in the list.

See also: ±**getObject** and **-lastObject**

free
- free

Frees all the nodes in the list and the list itself.

See also: ±**empty**, ±**free**, and **-freeNodes**

freeObjects
- freeObjects

Frees all the objects in the list and the nodes associated with them. Returns *self*.

See also: ±**empty**, ±**free**, and **-freeNodes**

freeNodes
- freeNodes

Frees all the nodes in the receiver but does not free the objects that were in the nodes. Returns *self*.

See also: ±**empty**, ±**free**, and **-freeObjects**

getLength
- (int)**getLength**

Returns the number of objects (nodes) in the list.

See also: **-count**

getNode
- getNode

Returns the current node. Under most circumstances you shouldn't need to use this method.

See also: **-getObject**

getObject
- getObject

Returns the object stored in the current node.

See also: ±**firstObject**, **-getNode**, and ±**lastObject**

goFirst
- goFirst

Make the first node of the list the current node. Returns *nil* if there are no objects in the list, otherwise returns *self*.

See also: ±**goLast**, ±**goNext**, and ±**goPrevious**

goLast
- goLast

Make the last node of the list the current node. Returns *nil* if there are no objects in the list, otherwise returns *self*.

See also: **-goFirst**, ±**goNext**, and ±**goPrevious**

goNext
- goNext

Make the node after the current node become the current node. If already at the end of the list, *nil* is returned. Otherwise, returns *self*.

See also: **-goFirst**, ±**goLast**, and ±**goPrevious**

goPrevious
- goPrevious

Make the node before the current node become the current node. If already at the start of the list, *nil* is returned. Otherwise, returns *self*.

See also: **-goFirst**, ±**goLast**, and ±**goNext**

init
- init

Initialize the receiving instance of MiscLinkedList with the MiscLinkedListNode class as the class used to create new nodes in the list. Returns *self*.

See also: **-initForClass:**

initForClass:
- initForClass:*this_class*

Initialize the receiving instance of MiscLinkedList with the class *this_class* as the class used to create new nodes in the list. Returns *self*.

See also: **-init**

insertList:
- insertList:*this_list*

Insert the list *this_list* before the current node of the receiving list. The objects that were in *this_list* are left in the same order when inserted into the receiver. The list *this_list* is left unchanged. Returns *self*.

See also: **-appendList:** and ±**insertListAtFront:**

insertListAtFront:
- insertListAtFront:*this_list*

Insert the list *this_list* at the front of the receiving list. The objects that were in *this_list* are left in the same order when inserted into the receiver. The list *this_list* is left unchanged. Returns *self*.

See also: **-appendList:** and ±**insertList:**

insertObjectAfter:
- insertObjectAfter:*this_object*

Insert *this_object* into the list after the current object/node. Returns *this_object*.

See also: **-insertObjectBefore:**

insertObjectBefore:
- insertObjectBefore:*this_object*

Insert *this_object* into the list before the current object/node. Returns *this_object*.

See also: **-insertObjectAfter:**

isEqual:
- (BOOL)**isEqual:***anObject*

Returns YES if *anObject* is the same class as the receiver and contains the exact same set of objects as determined by sending ±**isEqual:** to each of the objects in the list. Returns NO otherwise.

lastObject
- lastObject

Returns the last object in the list.

See also: **-firstObject** and **-getObject**

makeObjectsPerform:
- makeObjectsPerform:(SEL)*aSelector*

Traverses the list from the first to the last node, sending the *aSelector* message to each object in the list. Returns *self*.

See also: **-makeObjectsPerform:with:**

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

Traverses the list from the first to the last node, sending the *aSelector* message with the argument *anObject* to each object in the list. Returns *self*.

See also: **-makeObjectsPerform:**

moveToFirst
- moveToFirst

Moves the current node to the front of the MiscLinkedList. Returns *self*. If there is no current node, then *nil* is returned.

See also: **-moveToLast**, **-swapWithPrevious**, and **-swapWithNext**

moveToLast
- moveToLast

Moves the current node to the end of the MiscLinkedList. Returns *self*. If there is no current node, then *nil* is returned.

See also: **-moveToFirst**, **-swapWithPrevious**, and **-swapWithNext**

read:
- read:(NXTypedStream *)*stream*

Unarchives the receiving MiscLinkedList and all the objects it contains from *stream*. Returns *self*.

See also: **-write:**

removeFirstObject
- removeFirstObject

Removes the first object from the list. The removed object is returned.

See also: ±**deleteObject**, ±**removeLastObject**, and ±**removeObject:**

removeLastObject
- removeLastObject

Removes the last object from the list. The removed object is returned.

See also: ±**deleteObject**, ±**removeFirstObject**, and ±**removeObject:**

removeObject:
- removeObject:*anObject*

Searches through the list for the first occurrence of *anObject* and removes it from the list. Returns *anObject*.

See also: **-addObject:**, **-addObjectIfAbsent:**, ±**deleteObject**, ±**removeFirstObject**, and ±**removeLastObject**

replaceObject:with:
- replaceObject:*anObject* **with:***newObject*

Searches from the start of the list for the first occurrence of *anObject* and replaces it with *newObject*. Returns *anObject*. This method will change the current node; the replaced object's node becomes the new current node.

See also: **-setObject:**

setObject:
- setObject:*this_object*

Sets the object in the current node to *this_object*. Returns the object which was replaced.

See also: **-replaceObject:with:**

swapWithNext
- swapWithNext

Swaps the current node with the node just after of it the MiscLinkedList. Returns *self*. If there is no current node or no next node, then *nil* is returned.

See also: **-moveToFirst**, **-moveToLast**, and **-swapWithPrevious**

swapWithPrevious
- swapWithPrevious

Swaps the current node with the node in front of it the MiscLinkedList. Returns *self*. If there is no current node or no previous node, then *nil* is returned.

See also: **-moveToFirst**, **-moveToLast**, and **-swapWithNext**

write:
- write:(NXTypedStream *)*stream*

Archives the receiving MiscLinkedList and all the objects it contains to *stream*. Returns *self*.

See also: **-read:**