NXData

Inherits From: Object

Conforms To: NXTransport (Distributed Objects)

Declared In: machkit/NXData.h

Class Description

NXData is an object-oriented wrapper for data. It's especially useful in Distributed Objects applications because of its conformance to the NXTransport protocol, allowing NXData objects to be copied or moved between applications. NXData can be used to wrap data of any size; it allocates small amounts of memory from its own zone using a malloc-related function, and allocates page-aligned data from the virtual memory system for requests of a page or larger. NXData can also be used to wrap preexisting data, regardless of how the data was allocated.

If data is to be moved between applications (rather than copied), you may find it necessary to override the **encodeRemotelyFor:...** method in a subclass of NXData to ensure that data is properly deallocated after it is passed across a connection; see the Distributed Objects introduction for more information on moving objects between applications.

Instance Variables

None declared in this class.

Adopted Protocols

NXTransport	– encodeRemotelyFor:freeAfterEncoding:isBycopy: – encodeUsing: – decodeUsing:

Method Types

Initializing and freeing instances	– initWithSize: – initWithData:size:dealloc: – free
Getting the object's data	– data
Getting the data's size	– size
Copying the object	– copyFromZone:

Instance Methods

copyFromZone:

– **copyFromZone:**(NXZone *)*zone*

Returns a newly allocated NXData instance containing a copy of the receiver's data. The new object's data will be deallocated when the new object gets freed.

data

– (void *)**data**

Returns a pointer to the data contained in the object.

encodeRemotelyFor: freeAfterEncoding:isBycopy:

– **encodeRemotelyFor:** (NXConnection *)*conn*
freeAfterEncoding:(BOOL *)*flagPointer*
isBycopy:(BOOL)*isBycopy*

Returns **self** to indicate that a copy of the NXData object (and not a proxy to it) is to be copied across a connection any time the object is vended to a remote object. The data for the remote copy will be freed when the copy is freed. If you want the local NXData to be freed after being sent across the connection, you will need to override this method to set the boolean indicated by *flagPointer* to YES.

free

– **free**

Deallocates the receiver's storage, including the data if it was initialized to do so, and returns **nil**.

See also: – **initWithData:size:dealloc:**, – **initWithSize:**

initWithData:size:dealloc:

– **initWithData:**(void *)*data*
size:(unsigned int)*size*
dealloc:(BOOL)*flag*

Initializes the receiver, a new NXData object, with *data*, which must be at most *size* bytes long. If *flag* is YES, then *data* will be deallocated when the NXData object is freed. *data* could have been allocated with **vm_allocate()** or a **malloc()** variant. Returns **self**.

See also: – **initWithSize:**, – **free**

initWithSize:

– **initWithSize:**(unsigned int)*size*

Initializes the receiver, a new NXData object, so that it can contain at most *size* bytes of data. The memory will be allocated directly from the virtual memory system if it is one page or greater in size (though applications shouldn't care where the memory came from); otherwise the data will be allocated from the object's zone. The data will be freed when the NXData object is freed. Returns **self**.

See also: – **initWithData:size:dealloc:**, – **free**

size

– (unsigned int)**size**

Returns the size of the data that the object holds.