NXProxy

Inherits From: none (*NXProxy is a root class*.)

Conforms To: NXReference (Mach Kit)
NXTransport

Declared In: remote/NXProxy.h

Class Description

The NXProxy class defines objects that are used to stand in for real objects (descendants of the Object class), where the real objects may exist within another process, even across a network. To the application, the NXProxy appears to be the real object, though the real object may not be directly accessible. The real object is known as the proxy's *correspondent*, indicating both that the objects are counterparts and that the real object is required to respond to messages sent to the proxy.

The NXProxy class defines very few methods, because proxies respond to very few messages directly. Instead, when an NXProxy receives a message that it doesn't respond to, it encodes the message, including the arguments, and forwards it to its remote correspondent (the "real" object). The actual communication details involved in forwarding the message are taken care of by an NXConnection object. The message is then acted upon by the real object, and any return values and parameters are encoded and sent back to the proxy.

An application never instantiates NXProxy objects directly; they are created for your application when you are given a reference to an object that doesn't exist in your address space. The proxies vended to your application are reference-counted, so only a single NXProxy per connection is instantiated for any real object. When you're done with a remote object, you should typically send it a **free** message to eliminate its remote proxy locally and its local proxy remotely. This will decrement the reference-count on the proxy, and free it if there are no outstanding references. The **free** message will also be forwarded to the proxy's correspondent, which will free it (or dereference it if the object conforms to the NXReference protocol). An application alternatively might free the proxy's NXConnection, which will free all the connection's resources, including all its proxies.

The methods defined in this class are the ones that the NXProxy class directly responds to. Unless otherwise noted, none of these methods are forwarded to the proxy's correspondent.

Instance Variables

None declared in this class.

Adopted Protocols

NXReference	– addReference – free – references
NXTransport	– encodeRemotelyFor: freeAfterEncoding:isBycopy: – encodeUsing: – decodeUsing:

Method Types

Returning the proxy's connection	– connectionForProxy
Freeing an NXProxy instance	– freeProxy
Determining if an object is a proxy	– isProxy
Specifying a protocol	– setProtocolForProxy:

Instance Methods

connectionForProxy

– **connectionForProxy**

Returns the local NXConnection instance used by the receiving NXProxy. A client might send messages to the returned NXConnection to be notified of invalidations (such as port deaths), or to instruct it to begin receiving messages with a variant of the **run** message.

See also: – **registerForInvalidationNotification** (NXInvalidationNotifier in Mach Kit), – **runFromAppKit** (NXConnection)

free

– **free**

Decrements the reference count on the proxy. If there are remaining references to the proxy, the **free** message isn't forwarded across the connection and this method returns **self**. If there are no remaining references, the proxy forwards the **free** message to its corresponding object, invokes the **freeProxy** method to free the proxy locally, and returns **nil**.

freeProxy

– **freeProxy**

Frees the receiving NXProxy instance. You generally shouldn't send this message; it isn't forwarded across the connection, so remote NXConnection objects may still have references to the freed NXProxy and it won't get removed from remote hashtables. If you want to free the local proxy and eliminate outstanding references, the real object should obey the NXReference protocol; then when you send the object a **free** message, the proper dereferencing (and perhaps freeing) will occur both locally and remotely.

isProxy

– (BOOL)**isProxy**

Returns YES to indicate that the receiver is an NXProxy rather than a normal object. This method is also implemented in a category of the Object class (where it returns NO), so you can send this message to any object to determine whether it is a real object or a proxy.

setProtocolForProxy:

– **setProtocolForProxy:**(Protocol *)*proto*

Formally establishes the messages and arguments that the proxy will forward to its corresponding object. It's a good idea to send this message to an NXProxy immediately after it is vended to your application.

If you don't send this message to a proxy (and therefore a protocol isn't established), at run-time the proxy doesn't know a message's argument types, and can't immediately encode the arguments. It must then send a remote message to its corresponding object to get the argument types. This round trip increases the cost of the message. You should therefore send the **setProtocolForProxy:** message to the proxy to cache the argument types, alleviating the need for the initial round trip.

If you send a message that isn't in the established protocol, the round trip to establish the argument types will still be performed. You must take care that the argument types in the given protocol *proto* accurately reflect the argument types of the methods in the proxy's corresponding object; otherwise the arguments will not be correctly encoded. Returns **self**.