NXTransport

Adopted By: List (common classes)
NXData (Mach Kit)
NXPort (Mach Kit)
NXProxy class (Distributed Objects)

Declared In: remote/transport.h

Protocol Description

The NXTransport protocol allows objects to be copied over a Distributed Objects connection. This protocol consists of three methods:

- `encodeRemotelyFor:freeAfterEncoding:isBycopy:`
- `encodeUsing:`
- `decodeUsing:`

When an object must be vended over a connection, the **encodeRemotelyFor:freeAfterEncoding:isBycopy:** method is invoked to determine what object is sent. The Object class implements a version of this method that returns an NXProxy; thus all objects may be sent over a connection in virtual form through the use of a proxy. Classes can override this method to specify another object (that conforms to the NXTransport protocol) to be sent over the connection. By sending a real object over the connection rather than a proxy, some applications can save the overhead of remote messaging (though if the object changes, keeping copies synchronized is an issue).

When an object is to be encoded, it is sent an **encodeUsing:** message. The *portal* argument for this message is an object that implements the NXEncoding protocol and thus knows how to encode various data types. The object to be encoded should send data to *portal* that allows a copy of itself to be decoded.

In order to create the copy of the object on the receiving side, the object is allocated and a **decodeUsing:** message is sent to it. The newly allocated object is not initialized, so the **decodeUsing:** implementation generally should invoke the object's designated initializer method.

Instance Methods

decodeUsing:

– **decodeUsing:**(id <NXDecoding>)*portal*

A newly allocated instance is sent this message in order to initialize itself when an object has been sent by copy over a connection. The instance is not initialized, so it should generally invoke the object's designated initializer. You must send messages (from the NXDecoding protocol) to the *portal* object to fetch any data that was encoded; these messages may be sent before or after initializing the new instance.

This method generally returns **self** to indicate that **self** is the object that is to be used as the local copy of the sent object. If it returns another object, that object is used as the local copy, and the instance that received this message is freed.

See also: – **encodeUsing:**

encodeRemotelyFor:freeAfterEncoding:isBycopy:

– **encodeRemotelyFor:**(NXConnection *)*connection*
freeAfterEncoding:(BOOL *)*flagp*
isBycopy:(BOOL)*isBycopy*

This method is responsible for returning the object that must be encoded to send the receiver over *connection*. The default implementation inherited from the Object class returns a local proxy to the receiver which, when encoded, yields a remote proxy that forwards all messages to the original object.

You can override this method to change how an object is transported. If you return another object (like **self**), that object will be encoded instead. The returned object must conform to the **NXTransport** protocol. You may wish to test the *isBycopy* flag and return **self** only if the object (rather than a proxy) is to be copied across the connection. If you want the receiving object to be freed after it is encoded, you can set the boolean pointed to by *flagp* to YES.

A typical implementation of this method simply ensures that the object or a proxy gets encoded, based on the value of *isBycopy*:

```
- encodeRemotelyFor:(NXConnection *)connection
    freeAfterEncoding:(BOOL *)flagp isBycopy:(BOOL)isBycopy
{
    if (isBycopy) return self;
    return [super encodeRemotelyFor:connection
        freeAfterEncoding:flagp isBycopy:isBycopy];
}
```

encodeUsing:

– **encodeUsing:**(id <NXEncoding>)*portal*

This method must send enough data to *portal* (an object that conforms to the NXEncoding protocol) that a copy of the object can be created on the other side of a connection using the **decodeUsing:** method. See the introduction to Distributed Objects for an example implementation of this method.