Indexing Kit

Q: I need to customize the two methods **source:***aSource* **willWriteRecord:** (unsigned)*record* and **source:***aSource* **didReadRecord:**(unsigned)*record* from the IXRecordTranscription protocol to do a fast archiving of my custom data. However, I don't understand why *aSource* doesn't respond to the IXRecordManager API. How can I work around this problem?

A: The source inside the IXRecordTranscription protocol methods can either be an IXRecordManager or an IXDataRepository object. In the latter case, the record manager is its delegate. For example, in order to find the record manager, your code should look like this:

```
- source:aSource willWriteRecord:(unsigned)record
{

        /* Check whether aSource is an IXRecordManager */
        if (![aSource isMemberOf:[IXRecordManager class]])
                aSource = [aSource delegate];

        /* aSource is now an IXRecordManager */
        /* Process your custom data here */
        return self;
```

```
}
```

QA896

Valid for 3.0, 3.1, 3.2

Q: When I archive and unarchive an IXStore object, I am able to write it to a typedstream, but reading it back in gives me a memory protection failure with the following backtrace.

```
Program generated(1): Memory access exception on address 0xe
(protection failure).
0xa025f46 in -[IXStore read:] ()
(gdb) where
#0  0xa025f46 in -[IXStore read:] ()
#1  0x500c9be in InternalReadObject ()
#2  0x500f0f8 in NXReadObject ()
```

Why?

A: The memory smasher occurs in InternalReadObject() (the archiving code) when that method tries to send an **awake** message to the unarchived store, which has been freed.

The store was freed because it didn't have transactions enabled and hence was in a partially updated state. To fix this problem, you can, for example, call **commitTransaction** in the **write:** method of your store object to finish all outstanding transactions before archiving. Note that this is only necessary if transactions are not enabled. If transactions are enabled, the store can be archived with incomplete transactions pending, and reading it back drops the uncommitted changes. See the code snippet below:

```
/* Make a new storage object with a brand new IXStore */
- init
{
        [super init];
        storage = [[IXStore alloc] init];
        return self;
}

/* Archiving myself */
- read:(NXTypedStream *)stream
{
        [super read:stream];
        storage = NXReadObject(stream);
        return self;
}

- write:(NXTypedStream *)stream
```

```
{
        [super write:stream];
                /* A convenient place to finish outstanding
                transactions. Not needed if transactions are
                enabled.
                */
        [storage commitTransaction];
        NXWriteObject(stream, storage);
        return self;
}
```

Please note that this program crasher has been fixed in Release 3.2, and an exception error is raised instead. However, you still need to follow the above guideline to properly archive an IXStore object.

QA901

Valid for 3.1, 3.2