```
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  FILENAME:  eText.Pasteboard.m
//  SUMMARY:   Implementation of interactions between eText and Pasteboards
//  CATEGORY:  Pasteboard
//  PROTOCOLS: None
//  INTERFACE: None
//  AUTHOR:    Rohit Khare
//  COPYRIGHT: ©1993,94 California Institute of Technology, eText Project
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Implementation Comments
//      All actions should get routed to read/writeSelectionTo/FromPasteboard.
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  History
//  02/12/95:  Changed Pasting policy to prefer creation of annotiations.
//  11/20/94:  Realized that most apps can't handle filename && ASCII, stupidly.
//  11/20/94:  Extended to use writeASCII on copying selections to PBoard.
//  10/31/94:  Added detectors for smartPaste.
//  10/17/94:  Cleaned up for eText5.
//  08/05/94:  Completely Rearchitected for 5.0. RK
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Imported Interfaces
//
    #import "eText.Pasteboard.h"

#define eTETFPboardType  NXUniqueString("eText ETF pasteboard type")

@implementation eText(Pasteboard)
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Centralized Selection Management
//
- readSelectionFromPasteboard:pboard {
    return [self readSelectionFromPasteboard:pboard linked:NO];}
- readSelectionFromPasteboard:pboard linked:(BOOL)isLinked {
    NXSelPt     a,b;
    NXStream    *memstream;
    NXAtom      supportedTypes[9];
    NXAtom      foundType;

// 1) is there a selection? what types do we have?
    [self getSel:&a :&b];
    if (!((a.cp >= 0) && [self isEditable]))
        return nil;

    // All possible "Pasteable" types:
    supportedTypes[0] = eTETFPboardType;
    supportedTypes[1] = NXRTFDPboardType;
    supportedTypes[2] = NXRTFPboardType;
```

```
    supportedTypes[3] = NXTabularTextPboardType;
    supportedTypes[4] = NXAsciiPboardType;
    supportedTypes[5] = NXColorPboardType;
    supportedTypes[6] = NXFontPboardType;
    supportedTypes[7] = NXRulerPboardType;
    supportedTypes[8] = NXFilenamePboardType;
    supportedTypes[9] = NULL;
    foundType = [pboard findAvailableTypeFrom:supportedTypes num:2];

// 2) If we found nothing, try creating an Annotation for it
// 2a) NEW POLICY: If we can create an annotation, && no etf, do it
    if (!foundType && [etApp annotationByPboard:pboard]) {
        [self insertAnnotation: [[[etApp annotationByPboard:pboard] alloc]
              initFromPboard:pboard inDoc:etDoc linked:isLinked]];
        return self;
    }

// Now, check the COMPLETE Pasteboard Types List
    foundType = [pboard findAvailableTypeFrom:supportedTypes num:8];

// b) If we have filenames, parse them out and instantiate each one.
// NOTE THAT THIS DOESN'T WORK YET!!!!!! FIX ME!
    if (foundType == NXFilenamePboardType) {
        NXLogError("eText tried to read an NXFilenamePboardType.");
        [self insertAnnotation: [[[etApp annotationByPboard:pboard] alloc]
              initFromPboard:pboard inDoc:etDoc linked:isLinked]];
// 3) do we have RTFD? This is a _very_ special case, and is mutex to ETF
    } else if (foundType == NXRTFDPboardType) {
        // this is really gruesome code. Don't look.
        Text       *rtfd;
        NXRect     rect;
        Class      oldHandler;

        // unload the old \NeXTGraphic handler
        oldHandler = [eText classForDirective:"NeXTGraphic"];
        // And load in NeXT's crap. All cause they serialize to a private form.
        [eText  registerDirective:"NeXTGraphic"
               forClass:[Text graphicCellClass]];
        rect.origin.x = 0; rect.origin.y = 0;
        rect.size.height = MAXFLOAT; rect.size.width = MAXFLOAT;
        rtfd = [[Text alloc] initFrame:&rect]; // NeXT's Text class
        [rtfd setMonoFont:NO];
        [rtfd setGraphicsImportEnabled:YES];
        [rtfd setSel:0 :[rtfd textLength]];
        [rtfd readSelectionFromPasteboard: pboard]; // Text groks RTFD
        if ([rtfd textLength]) {
            NXRTFDError ret;
```

```
        ret=[rtfd saveRTFDTo:"/tmp/eTextRTFDConversionStoreFile.rtfd"
                 removeBackup:YES errorHandler:nil];
        // get that crappy NXGraphicCell code out of my kernel NOW!!!
        rtfd = [rtfd free];
        [eText  registerDirective:"NeXTGraphic"
                forClass:oldHandler];

        if (ret == NX_RTFDErrorNone) {
            NXAtom oldP,oldT; int start;

            // now parse in the good stuff
            // temporarily set this doc's docInfoPath to tempName
            oldP = [[etDoc docInfo] docPath];
            oldT = [[etDoc docInfo] docTitle];
            start = sp0.cp;
            [[etDoc docInfo]
                setDocPath:"/tmp/eTextRTFDConversionStoreFile.rtfd"];

            // issue a read for the "RTF" part of the data
            [self undoSelChange:"Paste RTFD"];
            [super readSelectionFromPasteboard:pboard];
            [self undoAffectedRange:start to:spN.cp];

            // preserves the traditional semantics of pasting
            // since readSelection doesn't move the cursor
            [self setSel:sp0.cp :sp0.cp];

            [[etDoc docInfo] setDocPath:oldP];
            [[etDoc docInfo] setDocTitle:oldT];
        }
        // unfortunately, the objects created may want persistent
        // access to the files on disk, so we can't nuke them.
        // There's possibly a bug here relating to the use of a single,
        // shared ConversionStoreFile to hold all this shit.
        // Oh, for want of a OOfilesystem with autorelease!
    }
    // get that crappy NXGraphicCell code out of my kernel NOW!!!
    [eText  registerDirective:"NeXTGraphic"
            forClass:oldHandler];
// 4) do we have ETF or RTF (in that preference order?)
  } else if ((foundType == eTETFPboardType) ||
            (foundType == NXRTFPboardType)){
    // OK, just paste in the ETF/RTF code
    memstream = [pboard readTypeToStream:foundType];
    if (memstream) {
        char buf[3]; int smart=0; // Default is to assume dumbpaste
```

```
            int start = sp0.cp;

            if ([self getSubstring:buf start:spN.cp length:1] != -1){
                smart = (strchr(postSelSmartTable, buf[0]) ? 0 : 1);
            }
            [self undoSelChange:"Paste ETF/RTF"];
            [self replaceSelWithRichText:memstream]; // subsumes ETF as well.
            if (!(smart &&
                ([self getSubstring:buf start:spN.cp length:1] != -1) &&
                (buf[0] == ' ')))
                smart = 0;
            [self undoAffectedRange:start to:(spN.cp + smart)];
            NXCloseMemory(memstream, NX_FREEBUFFER);
        }
// 5) do we have ASCII or Tabular Text?
    }else if ((foundType == NXAsciiPboardType) ||
              (foundType == NXTabularTextPboardType)){
        // OK, just paste in the ASCII code
        char *data; int len, start=sp0.cp;

        [self undoSelChange:"Paste PlainText"];
        [pboard readType:foundType data:&data length:&len];
        [self replaceSel:data length:len]; // subsumes TabText as well.
        [pboard deallocatePasteboardData:data length:len];
        [self undoAffectedRange:start to:spN.cp];
        return self;
// 6) do we have Color data?
    } else if (foundType == NXColorPboardType) {
        [self setSelColor:NXReadColorFromPasteboard(pboard)];
// 7) do we have Font data?
// 8) do we have Ruler data?
    } else if ((foundType == NXFontPboardType) ||
               (foundType == NXRulerPboardType)) {
        // this is kind of funky: we pull a little switcheroo on Text
        id     targetPboard;
        char   *saveData,*newData;
        int    saveLen, newLen;
        NXAtom targetType[1];

        // determine the targetPboard
        if (foundType == NXRulerPboardType) {
            targetPboard = [Pasteboard newName:NXRulerPboard];
            *targetType = NXRulerPboardType;
        } else {
            targetPboard = [Pasteboard newName:NXFontPboard];
            *targetType = NXFontPboardType;
        }
```

```
        // save data from targetPboard
        [targetPboard  readType:*targetType
                       data:&saveData length:&saveLen];

        // copy data from pboard to targetPboard
        [pboard        readType:*targetType
                       data:&newData length:&newLen];
        [targetPboard declareTypes:targetType num:1 owner:nil];
        [targetPboard writeType:*targetType data:newData length:newLen];

        // perform operation
        if (foundType == NXRulerPboardType) {
            [self pasteRuler:self];
        } else {
            [self pasteFont:self];
        }

        // restore data to targetPboard
        [targetPboard declareTypes:targetType num:1 owner:nil];
        [targetPboard writeType:*targetType data:saveData length:saveLen];

        // free both datas
        [targetPboard deallocatePasteboardData:saveData length:saveLen];
        [pboard deallocatePasteboardData:newData length:newLen];
    }
    return self;
}
- writeSelectionToPasteboard:pboard {
    NXSelPt     a,b;
    int         pos,k,N;
    NXRun       *curr;
    NXStream    *memstream;
    NXAtom      supportedTypes[5];

// 1) is there a selection of nonzero length?
    [self getSel:&a :&b];
    if (!((a.cp >=0) && [self isEditable] && (b.cp != a.cp))) return nil;

// 2) Initialize our pboard
    // after some extensive experimentation, calling writeSelection does
    // seem to work after all. The issue is that a real "copy" can have a
    // "NeXT smart paste pasteboard type" which wstopb cannot. Tests in
    // Edit reveal that this has no effect. Some keen user may find one though.
    supportedTypes[0] = eTETFPboardType;
    supportedTypes[1] = NXRTFPboardType;
    supportedTypes[2] = NXAsciiPboardType;
```

```
    supportedTypes[3] = NXFilenamePboardType;
    supportedTypes[3] = NULL;
    supportedTypes[4] = NULL;
    [pboard declareTypes:supportedTypes num:3 owner:nil];

    memstream = NXOpenMemory(NULL,0,NX_READWRITE);
    if (memstream) {
        [self writeETF:memstream from:sp0.cp to:spN.cp];
        [pboard writeType:eTETFPboardType fromStream:memstream];
        NXCloseMemory(memstream, NX_FREEBUFFER);
    }

    memstream = NXOpenMemory(NULL,0,NX_READWRITE);
    if (memstream) {
        [self writeRTF:memstream from:sp0.cp to:spN.cp];
        [pboard writeType:NXRTFPboardType fromStream:memstream];
        NXCloseMemory(memstream, NX_FREEBUFFER);
    }

    memstream = NXOpenMemory(NULL,0,NX_READWRITE);
    if (memstream) {
        [self writeASCII:memstream from:sp0.cp to:spN.cp];
        [pboard writeType:NXAsciiPboardType fromStream:memstream];
        NXCloseMemory(memstream, NX_FREEBUFFER);
    }

//  [pboard writeType:NXFilenamePboardType data:[[etDoc docInfo] docPath]
//          length:strlen([[etDoc docInfo] docPath])+1];

// 3) are there Annotations in our selection?
    pos=0;
    N = theRuns->chunk.used/sizeof(NXRun);
    curr = theRuns->runs;
    for (k=0; ((k < N) && (b.cp > pos)); k++) {
        if ((curr->info) && (a.cp <= pos) &&
            [curr->info respondsTo:@selector(addToPboard:)])
            [curr->info addToPboard:pboard];
            // this is kosher because a second addType of the same type fails.
            // but a second writeType:of a different type might!
        pos += curr->chars;
        curr++;
    }
    return self;
}
- (BOOL)writeSelectionToPasteboard:pboard types:(NXAtom *)types
{
    id retVal;
```

```
    int numTypes = 0;

    retVal=[self writeSelectionToPasteboard:pboard];
    // now check to see that one of _types_ is on the pboard to return YES
    while(types && types[numTypes]){
        if (types[numTypes] == NXFilenamePboardType) {
            if (retVal)
                [pboard addTypes:&(types[numTypes]) num:1 owner:nil];
            else
                [pboard declareTypes:&(types[numTypes]) num:1 owner:nil];
            [pboard writeType:NXFilenamePboardType data:[[etDoc docInfo] docPath]
                 length:strlen([[etDoc docInfo] docPath])+1];
        }
        numTypes++;                    // Counter
    }
    return ([pboard findAvailableTypeFrom:types num:numTypes] ? YES : NO);
}

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Dropping
//
- (NXDragOperation)draggingEntered:(id <NXDraggingInfo>)sender {
    return [self draggingUpdated:sender];}
- (NXDragOperation)draggingUpdated:(id <NXDraggingInfo>)sender {
    NXSelPt         a,b;
    NXPoint         p;
    NXCoord         l,r,t,bot;
    NXDragOperation sourceMask = [sender draggingSourceOperationMask];

    // Check for self-drags
    [self getMarginLeft:&l right:&r top:&t bottom:&bot];
    p = [sender draggingLocation];
    if ((p.x < (l + SCROLLWIDTH)) && ([sender draggingSource] == self))
        return NX_DragOperationNone;

    // Check that the selection exists and isEditable
    [self getSel:&a :&b];
    if (!((a.cp >=0) && [self isEditable]))
        return NX_DragOperationNone;

    // It would be really nice to check whether the dragging Source is an
    // Annotation and is *within* the current selection (auto-destruct!)
    // Instead, it is the responsibility of each Annotation (see eTImage)
    // Also, consider laying the image in the window at the insertionPoint

    if (sourceMask & NX_DragOperationLink) {
        return NX_DragOperationLink;
```

```
    } else if (sourceMask & NX_DragOperationCopy) {
        return NX_DragOperationCopy;
    } else if (sourceMask & NX_DragOperationGeneric) {
        return NX_DragOperationGeneric;
    }
    return NX_DragOperationNone;
}
- (BOOL)prepareForDragOperation:(id <NXDraggingInfo>)sender {
    // Future Feature: slide to the insertion point? NXSelPt sp0?
    // Edit implementation: remove the temp image of *entered/exit
    return YES;
}
- (BOOL)performDragOperation:(id <NXDraggingInfo>)sender {
    [self readSelectionFromPasteboard:[sender draggingPasteboard]
    linked:([sender draggingSourceOperationMask] & NX_DragOperationLink)];
    return YES;
}
- concludeDragOperation:(id <NXDraggingInfo>)sender {
    [[self window] makeKeyAndOrderFront:self];
    return self;
}
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Changing Fonts
//
- acceptFont:(Font *)fontA atPoint:(NXPoint *)p {
    id font;

    if ([fontA matrix] != NX_FLIPPEDMATRIX)
        font = [Font newFont:[fontA name]
                       size:[fontA pointSize]
                     matrix:NX_FLIPPEDMATRIX];
    else font = fontA;

    if (sp0.cp == spN.cp)      // "insert" a font at a flashing caret.
        [self setSel:sp0.cp :(sp0.cp+1)];
    [self setSelFont:font];    // undo is in setSelFont:
    //[self setSel:sp0.cp :spN.cp]; Remind Text object where the selection is!
    return self;
}
- replaceFont:(Font *)oldFontA with:(Font *)newFontA {
    id      oldFont, newFont;
    int     k,N;
    NXRun   *curr;

    // First, check that the font matrices are in order
    if ([oldFontA matrix] != NX_FLIPPEDMATRIX)
        oldFont = [Font newFont:[oldFontA name]
```

```
                      size:[oldFontA pointSize]
                    matrix:NX_FLIPPEDMATRIX];
    else oldFont = oldFontA;

    if ([newFontA matrix] != NX_FLIPPEDMATRIX)
        newFont = [Font newFont:[newFontA name]
                      size:[newFontA pointSize]
                    matrix:NX_FLIPPEDMATRIX];
    else newFont = newFontA;

    // Freeze the display, Loop over all the NXRuns, recalc
    [[self window] disableFlushWindow];
    N = theRuns->chunk.used/sizeof(NXRun);
    curr = theRuns->runs;
    for (k=0; k < N; k++) {
        if (curr->font == oldFont)
            curr->font = newFont;
        curr++;
    }
    [self calcLine];
    [[[[self window] display] reenableFlushWindow] flushWindowIfNeeded];
    return self;
}

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Edit Menu
//
- copy:sender {
    [self writeSelectionToPasteboard:[Pasteboard newName:NXGeneralPboard]];
    return self;
}
- paste:sender {
    return [self readSelectionFromPasteboard:[Pasteboard newName:NXGeneralPboard]];
}
- clear:sender {
    return [self delete:sender];}
- delete:sender {
    id retval;
    if (sp0.cp == spN.cp) return nil;
    [self undoSelChange:"Delete Selection"];
    retval = [super delete:sender];
    [self undoAffectedRange:sp0.cp to:sp0.cp];
    return retval;
}

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Services Menu
```

```
//
- validRequestorForSendType:(NXAtom)typeSent
             andReturnType:(NXAtom)typeReturned
{
    // OK, so sue me, we can't advertise that we send images and audio
    // by actually confirming it dynamically -- that's too damn expensive
    // Originally, we only checked if the sent type was one of:
    // NXFilenamePboardType,NXAsciiPboardType,NXRTFPboardType
    // Now, we just say what the hell and return self.
    // If the requestor actually wanted audio data and the selection is
    // text, tough luck. But this way, EqB's Edit Equation works...
    if (typeReturned && (![self isEditable])) return nil;
    return self;
}

@end
```