```
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  FILENAME:  eText.PlainText.m
//  SUMMARY:   Implementation of PlainText markup formats of eText (ASCII, C)
//  CATEGORY:  PlainText
//  PROTOCOLS: <ASCIISupport>, <CSupport>
//  INTERFACE: None
//  AUTHOR:    Rohit Khare
//  COPYRIGHT: ©1993,94 California Institure of Technology, eText Project
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Implementation Comments
//      These methods are pretty much syntactic sugar.
//      I can't decide whether 'C' should unravel 8-bit characters into ascii
//  equivalents. OTOH, who programs with ligatures? -- and if they do, leave
//  it alone!
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  History
//  10/17/94:  Cleaned up for eText5.
//  08/05/94:  Completely Rearchitected for 5.0. RK
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Imported Interfaces
//
    #import "eText.PlainText.h"

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Encoder API
//
void ASCIIEncoder(NXStream *s, unsigned char *item, int len) {
    int i; char *out;

    if(!len) len=strlen((unsigned char*)item);
    for (i=0; i<len; i++) {
        out = NXToAscii(item[i]);
        NXWrite(s, out, strlen(out));
    }
}

@implementation eText(PlainText)
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Path Operators
//
- readASCIIfromPath:(const char *)path {
    NXStream *memstream;

    memstream = NXMapFile(path, NX_READONLY);
    if (memstream){
        [self readASCII: memstream];
        NXCloseMemory(memstream, NX_FREEBUFFER);
```

```
    } else
        return nil;
    return self;
}
- writeASCIItoPath:(const char *)path {
    NXStream *memstream;

    memstream = NXOpenMemory(NULL, 0, NX_WRITEONLY);
    if (memstream) {
        [self writeASCII:memstream];
        NXSaveToFile(memstream, path);
        NXCloseMemory(memstream, NX_FREEBUFFER);
    } else
        return nil;
    return self;
}

- readCfromPath:(const char *)path {
    NXStream *memstream;

    memstream = NXMapFile(path, NX_READONLY);
    if (memstream){
        [self readC: memstream];
        NXCloseMemory(memstream, NX_FREEBUFFER);
    } else
        return nil;
    return self;
}
- writeCtoPath:(const char *)path {
    NXStream *memstream;

    memstream = NXOpenMemory(NULL, 0, NX_WRITEONLY);
    if (memstream) {
        [self writeC:memstream];
        NXSaveToFile(memstream, path);
        NXCloseMemory(memstream, NX_FREEBUFFER);
    } else
        return nil;
    return self;
}

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Stream Operators
//
- readASCII: (NXStream *)s {
    return [self readText:s];}
- writeASCII: (NXStream *)s {
```

```
   return [self writeASCII:s from:0 to:[self textLength]];}
- writeASCII: (NXStream *)s from:(int) start to:(int) end {
   int                  k,N,cnt=0;
   NXRun                *curr;
   NXTextBlock          *currBlock;
   int                  currentOffset,targetOffset;
   id                   compList;

   N = theRuns->chunk.used/sizeof(NXRun);
   curr = theRuns->runs;
   currBlock = [self firstTextBlock];
   compList = [[List alloc] init];
   currentOffset = 0;
   for(k=0; (k < N) && (currentOffset < end); k++) {

       targetOffset = currentOffset + curr->chars;

       if ((targetOffset > start) && (curr->info)) {
           if ([curr->info respondsTo:@selector(writeASCIIRef:forView:)]) {
               NXPrintf(s,"[%d]", cnt++);
               [compList addObject:(curr->info)];
           }
           if ([curr->info respondsTo:@selector(writeASCII:forView:)]){
               [curr->info writeASCII:s forView:self];
           }
       }

       // consume full blocks
       while (currBlock && (targetOffset >= currBlock->chars)) {

           if ((targetOffset > start) &&
               (curr->info == nil) &&
               (targetOffset > currentOffset))

                   ASCIIEncoder(s, currBlock->text + MAX(currentOffset,start),
                       MIN(end,currBlock->chars) - MAX(currentOffset,start));

           targetOffset -= currBlock->chars;
           start -= currBlock->chars;
           end -= currBlock->chars;
           currentOffset=0;
           currBlock = currBlock->next;
       }

       // consume partial block
       if (currBlock)
```

```
        if ((targetOffset > start) &&
            (curr->info == nil) &&
            (targetOffset > currentOffset))

            ASCIIEncoder(s, currBlock->text + MAX(currentOffset,start),
                        MIN(end,targetOffset) - MAX(currentOffset,start));

        currentOffset=targetOffset;
        curr++;
    }

    // compList
    N=[compList count];
    if (N) {
        NXPrintf(s,"\n\nREFERENCES\n\n");
        for (k=0; k < N; k++) {
            NXPrintf(s,"[%d]\t", k);
            [[compList objectAt:k] writeASCIIRef:s forView:self];
            NXPutc(s,'\n');
        }
    }
    [compList free];
    return self;
}

- readC: (NXStream *)s {
    return [self readText:s];}
- writeC: (NXStream *)s {
    return [self writeC:s from:0 to:[self textLength]];}
- writeC: (NXStream *)s from:(int) start to:(int) end {
    int                 k,N;
    NXRun               *curr;
    NXTextBlock         *currBlock;
    int                 currentOffset,targetOffset;

    N = theRuns->chunk.used/sizeof(NXRun);
    curr = theRuns->runs;
    currBlock = [self firstTextBlock];
    currentOffset = 0;
    for (k=0; k < N; k++) {
        if (curr->info) {
            if ([curr->info respondsTo:@selector(writeC:forView:)]){
                [curr->info writeC:s forView:self];
            }
        }
        // encode the text corresponding to the run
        // misson is to write (cumulative) curr->chars chars beginning
```

```
    // at currentCount. boundaries may map onto > 1 block
    targetOffset = currentOffset + curr->chars;
    // consume full blocks
    while ((currBlock) && (targetOffset >= (currBlock->chars))) {
        if (!(curr->info))                // throw annotated bits in bucket
            if(targetOffset>currentOffset) // don't pass len=0 to encoder
                NXWrite(s, currBlock->text+currentOffset,
                           currBlock->chars - currentOffset);
        targetOffset -= currBlock->chars;
        currBlock = currBlock->next;
        currentOffset=0;
    }
    // consume partial block
    if (currBlock && (! curr->info))   // throw annotated bits in bucket
        if(targetOffset>currentOffset) // don't pass len=0 to encoder
            NXWrite(s, currBlock->text + currentOffset,
                       targetOffset-currentOffset);
    currentOffset=targetOffset;
    curr++;
  }
  return self;
}

@end
```