```
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  FILENAME:  eText.XText.m
//  SUMMARY:   Implementation of the XText keybinding subsystem of eText
//  CATEGORY:  XText
//  PROTOCOLS: Uses XTActions
//  INTERFACE: None
//  AUTHOR:    Rohit Khare, portions by Mike Dixon
//  COPYRIGHT: ©1993,94 California Institure of Technology, eText Project
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Description
//      There is code related to XText initialization in eText.Class.m.
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  History
//  10/18/94:  Cleaned up for eText5.
//  08/05/94:  Completely Rearchitected for 5.0. RK
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  Imported Interfaces
//
    #import "eText.XText.h"

//  A (not very elegant) table format for storing the initial emacs bindings.
//  Unused args are indicated by the magic value 99.
//typedef struct {
//  const SEL  *sel;
//  short      arg1;
//  short      arg2;
//  keyCode    key;
//} tbl_entry;

//  For these and other key codes, refer to
//  /NextLibrary/Documentation/NextDev/Summaries/06_KeyInfo/KeyInfo.rtfd
/* tbl_entry emacs_base[] = {
{&@selector(moveChar:mode:), -1,  0, 0x351},// ctrl-b   move back char
{&@selector(moveChar:mode:), -1,  1, 0x401},// ctrl-h   delete back char
{&@selector(moveChar:mode:), -1,  3, 0x353},// ctrl-B   select back char
{&@selector(moveChar:mode:),  1,  0, 0x3c1},// ctrl-f   move fwd char
{&@selector(moveChar:mode:),  1,  1, 0x3b1},// ctrl-d   delete fwd char
{&@selector(moveChar:mode:),  1,  3, 0x3c3},// ctrl-f   select fwd char
{&@selector(moveWord:mode:), -1,  0, 0x354},// alt-b    move back word
{&@selector(moveWord:mode:), -1,  1, 0x1b4},// alt-del  delete back word
{0,                           0,  0, 0x404}, // alt-h    (ditto)
{&@selector(moveWord:mode:), -1,  3, 0x356},// alt-B    select back word
{&@selector(moveWord:mode:),  1,  0, 0x3c4},// alt-f    move fwd word
{&@selector(moveWord:mode:),  1,  1, 0x3b4},// alt-d    delete fwd word
{&@selector(moveWord:mode:),  1,  3, 0x3c6},// alt-F    select fwd word
{&@selector(moveLine:mode:), -1,  0, 0x081},// ctrl-p   move back line
{&@selector(moveLine:mode:), -1,  3, 0x083},// ctrl-P   select back line
```

```
{&@selector(moveLine:mode:),  1,  0, 0x371},// ctrl-n   move fwd line
{&@selector(moveLine:mode:),  1,  3, 0x373},// ctrl-N   select fwd line
{&@selector(lineBegin:),     0, 99, 0x391}, // ctrl-a   move to line begin
{&@selector(lineBegin:),     3, 99, 0x393}, // ctrl-A   select to line bgn
{&@selector(lineEnd:),       0, 99, 0x441}, // ctrl-e   move to line end
{&@selector(lineEnd:),       1, 99, 0x3e1}, // ctrl-k   delete to line end
{&@selector(lineEnd:),       3, 99, 0x443}, // ctrl-E   select to line end
{&@selector(docBegin:),      0, 99, 0x2e6}, // alt-<    move to doc begin
{&@selector(docEnd:),        0, 99, 0x2f6}, // alt->    move to doc begin
{&@selector(collapseSel:),   0, 99, 0x381}, // ctrl-spc collapse selection
{&@selector(transChars),    99, 99, 0x481}, // ctrl-t   transpose chars
{&@selector(setNextAction:), 0, 99, 0x421}, // ctrl-q   quote next key
{&@selector(insertNextChar), 99, 99, 0x425},// ctrl-alt-q really quote key
{&@selector(openLine),     99, 99, 0x071},  // ctrl-o   open line
{&@selector(scroll::),      1, -1, 0x341},  // ctrl-v   scroll fwd page
{0,                         0,  0, 0x0f6},  // alt-shft-down (ditto)
{&@selector(scroll::),     -1,  1, 0x344},  // alt-v    scroll back page
{0,                         0,  0, 0x166},  // alt-shft-up(ditto)
{&@selector(scroll::),      0,  4, 0x343},  // ctrl-V   scroll fwd 4 lines
{&@selector(scroll::),      0, -4, 0x346},  // alt-V    scroll back 4 lines
{&@selector(scroll::),  -9999,  0, 0x165},  // alt-ctrl-up scroll to start
{&@selector(scroll::),   9999,  0, 0x0f5},  // alt-ctrl-down  scroll to end
{&@selector(scrollIfRO::),   1, -1, 0x380}, // space    scroll fwd pg if RO
{&@selector(scrollIfRO::),  -1,  1, 0x1b0}, // del      scroll back pg if RO
{&@selector(scrollIfRO::),   0,  4, 0x382}, // shift-sp scroll fwd 4 lines
{&@selector(scrollIfRO::),   0, -4, 0x1b2}, // shft-del scroll back 4 lines
{&@selector(scrollSelToVisible),
                           99, 99, 0x2d1},  // ctrl-l   scroll to selection
{0, 0, 0}
};
*/
/*void initbase_emacs(actionTbl actions, NXZone *zone)
{
   keyCode i;
   tbl_entry *e;
   XTAction *a = [XTAction undefinedAction];

   // make all non-command control & alt combinations invoke "unboundKey"
   for (i=0; i<KEY_CODES; i+=16) {
      actions[i+1] = actions[i+3] = actions[i+4] = actions[i+5]
         = actions[i+6] = actions[i+7] = a;
   }

   // ... except for ctrl-i (a handy substitute for tab)
   actions[6*16 + 1] = nil;

   // and then install the emacs key bindings
```

```
    for (e=emacs_base; (e->key != 0); ++e) {
        if (e->sel == 0) {}
            // same action as previous binding
        else if (e->arg1 == 99)
            a = [[XTMsg0Action allocFromZone:zone] initSel:*(e->sel)];
        else if (e->arg2 == 99)
            a = [[XTMsg1Action allocFromZone:zone]
                    initSel:*(e->sel) arg:e->arg1];
        else
            a = [[XTMsg2Action allocFromZone:zone]
                    initSel:*(e->sel) arg:e->arg1 arg:e->arg2];
        actions[e->key] = a;
    }
}
*/
unsigned char GetPrevious(NXStream *s)
{
     int pos, ch;

     pos = NXTell(s);
     if (pos <= 0) return EOF;
     NXSeek(s, --pos, NX_FROMSTART);
     ch = NXGetc(s);
     NXUngetc(s);
     return ch;
}

@implementation eText(XText)
//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  XText0 Management
//

- doesNotRecognize:(SEL)sel {
    char msg[256];

    sprintf(msg, "No method for %.48s on this text object", sel_getName(sel));
    [errorStream report: msg];
    return self;
}

- setErrorStream:errs {
    // Egregious paranoia
    if ([errs respondsTo: @selector(report:)]) errorStream = errs;
    else [errorStream report:"Invalid argument to setErrorStream:"];
    return self;
}
- errorStream {
    return errorStream;}

- setInitialAction:action {
    initialAction = nextAction = action; return self;}
- initialAction {
    return initialAction;}
- setNextAction:action {
    nextAction = action; return self;}

- unboundKey {
    NXBeep(); return self;}
- keyDown:(NXEvent *)event {
    id temp;

    temp = nextAction;
    nextAction = initialAction;
    if (temp) {
        temp = [temp applyTo:self event:event]; // this could turn autoD off...
        if (vFlags.disableAutodisplay) {      // this turns it back on...
            [self setAutodisplay:YES];
            [[self superview] display];
        }
        if (temp && (sp0.cp == spN.cp))
        // RK: Added temp check to prevent setSel from lousing up the typingRun's
font selection
            [self setSel:sp0.cp :sp0.cp];       // hack to make caret reappear
    }
    return temp ? self : [super keyDown:event];
}

//ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
//  XText Operations (emacs)
//
- goto:(int)pos end:(int)end mode:(int)mode {
    int start;

    switch(mode) {

    case 0:    // move
        [self setSel:pos :pos];
        [self scrollSelToVisible];
        posHint = -1;
        break;

    case 1:    // delete
    case 2:    // cut
        if (pos != end) {
            start = pos;
            if (start > end)
                { start = end; end = pos; }
            [self setSel:start :end];
            if (mode == 1)
                [self delete:self];
            else
                [self cut:self];
        }
        posHint = -1;
        break;

    case 3:     // select
        start = pos;
        if (start > end)
            { start = end; end = pos; }
        // The Text object can't even extend the selection without flashing,
        // unless we disable autodisplay
        //if (sp0.cp != spN.cp)
        //  [self disableAutodisplay];
        [self setSel:start :end];
        posHint = pos;
        break;
    }
    xHintPos = -1;
    return self;
}

- moveChar:(int)cnt mode:(int)mode {
    int pos, end;
    int max = [self textLength];

    if (sp0.cp == posHint) {
        pos = sp0.cp + cnt;
        end = spN.cp;
    } else {
        pos = spN.cp + cnt;
        end = sp0.cp;
    }
    if (pos < 0)
        pos = 0;
    else if (pos > max)
        pos = max;
    return [self goto:pos end:end mode:mode];
}

- moveWord:(int)cnt mode:(int)mode {
    NXStream *s = [self stream];
    char c;
    int i, pos, end;
    unsigned char digit_cat = charCategoryTable['0'];
    unsigned char alpha_cat = charCategoryTable['a'];
    unsigned char c_cat;
    BOOL inWord = NO;

    if (cnt == 0)
        return self;
    if (sp0.cp == posHint) {
        pos = sp0.cp;
        end = spN.cp;
    } else {
        pos = spN.cp;
        end = sp0.cp;
    }
    NXSeek(s, pos, NX_FROMSTART);
    i = (cnt<0 ? -cnt : cnt);
    while (1) {
        c = (cnt<0 ? NXBGetc(s) : NXGetc(s));
        if (c == EOF) break;
        c_cat = charCategoryTable[c];
        if (c_cat==alpha_cat || c_cat==digit_cat)
            inWord = YES;
        else if (inWord) {
            --i;
            if (i > 0)
                inWord = NO;
            else
                break;
        }
    }
    pos = NXTell(s);
    if (c != EOF)
        pos += (cnt<0 ? 1 : -1);
    return [self goto:pos end:end mode:mode];
}

- moveLine:(int)cnt mode:(int)mode {
    int pos, end, x, dir;

    if (sp0.cp == posHint) {
        pos = sp0.cp;
        end = spN.cp;
    } else {
        pos = spN.cp;
        end = sp0.cp;
    }
    //if (mode != 0)
    //  [self disableAutodisplay];
    // collapse and normalize the selection
    [self setSel:pos :pos];
    x = (sp0.cp == xHintPos ? xHint : (sp0.cp - sp0.c1st));

    if (cnt < 0) {
        dir = NX_UP;
        cnt = -cnt;
    } else {
        dir = NX_DOWN;
    }
    for (; cnt > 0; --cnt)
        [self moveCaret: dir];

    pos = LINE_LENGTH(sp0.line)-1;
    if (x < pos)
        pos = x;
    pos += sp0.c1st;
    [self goto:pos end:end mode:mode];
    xHintPos = pos;
    xHint = x;
    return self;
}

- lineBegin:(int)mode {
    int pos, end;

    if (sp0.cp == posHint) {
        pos = sp0.c1st;
        end = spN.cp;
    } else {
        pos = spN.c1st;
        // Text is inconsistent about what line it thinks we're on
        if (spN.cp == (spN.c1st + LINE_LENGTH(spN.line)))
            pos = spN.cp;
        end = sp0.cp;
    }
    return [self goto:pos end:end mode:mode];
}

- lineEnd:(int)mode {
    NXSelPt *sp;
    int pos, end;
    if (sp0.cp == posHint) {
        sp = &sp0;
        end = spN.cp;
    } else {
        // need to correct for TBD
        sp = &spN;
        end = sp0.cp;
    }
    pos = sp->c1st + LINE_LENGTH(sp->line) - 1;
    if (pos < sp->cp) {
        // Text is being flakey again; we really want to be on the next line
        // this is pretty gross
        pos = sp->line;
        if (theBreaks->breaks[pos/sizeof(NXLineDesc)] < 0)
            pos += sizeof(NXHeightChange);
        else
            pos += sizeof(NXLineDesc);
        pos = sp->cp + LINE_LENGTH(pos) - 1;
    }
    if ((pos == sp->cp) && (mode != 0))
        ++pos;
    return [self goto:pos end:end mode:mode];
}

- docBegin:(int)mode {
    return [self   goto:0
                    end:(sp0.cp == posHint ? spN.cp : sp0.cp)
                   mode:mode];
}

- docEnd:(int)mode {
    return [self   goto:[self textLength]
                    end:(sp0.cp == posHint ? spN.cp : sp0.cp)
                   mode:mode];
}

- collapseSel:(int)dir {
    int pos;

    if ((dir < 0) || ((dir == 0) && (sp0.cp == posHint)))
        pos = sp0.cp;
    else
        pos = spN.cp;
    return [self goto:pos end:pos mode:0];
}

- transChars {
    int pos = sp0.cp;
    char buf[2], temp;

    if (pos == spN.cp) {
        if (pos == (sp0.c1st + LINE_LENGTH(sp0.line) - 1))
            --pos;
        if (pos > 0)
            if ([self getSubstring:buf start:pos-1 length:2] == 2) {
                temp = buf[1]; buf[1] = buf[0]; buf[0] = temp;
                //[self disableAutodisplay];
                [self setSel:pos-1 :pos+1];
                [self replaceSel:buf length:2];
                return self;
            }
    }
    NXBeep();
    return self;
}

- openLine {
    int pos = sp0.cp;

    // don't do anything if there's a non-empty selection
    if (pos == spN.cp) {
        [self replaceSel:"\n"];
        [self setSel:pos :pos];
    } else
        NXBeep();
    return self;
}

- scroll:(int)pages :(int)lines {
    NXRect r;

    // if our superview isn't a ClipView, we can't scroll
    if ([superview respondsTo:@selector(rawScroll:)]) {
        [superview getBounds:&r];
        r.origin.y += pages*r.size.height + lines*[self lineHeight];
        // Added by RK to keep one line of context on pgdowns.(insurance)
        if (pages) r.origin.y -= [self lineHeight];
        [superview _scrollTo:&r.origin];
    } else
        NXBeep();
    return self;
}

- scrollIfRO:(int)pages :(int)lines {
    if (![self isEditable])
        return [self scroll:pages :lines];
    else
        return nil;
}

- insertChar:(NXEvent *)event {
    char c;

    c = event->data.key.charCode;
    [self replaceSel:&c length:1];
    return self;
}

- insertNextChar {
    static id action = nil;

    if (!action)
        action = [[XTEventMsgAction allocFromZone:[NXApp zone]]
                            initSel:@selector(insertChar:)];
    nextAction = action;
    return self;
}

- autoIndent
{
    int pos, end;
    unsigned char buf[2];

    // don't do anything if there's a non-empty selection
    if (sp0.cp != spN.cp) {
        NXBeep();
        return self;
    }

    if (sp0.cp == posHint) {
        pos = sp0.c1st;
        end = spN.cp;
    } else {
        pos = spN.c1st;
        // Text is inconsistent about what line it thinks we're on
        if (spN.cp == (spN.c1st + LINE_LENGTH(spN.line))){
            pos = spN.cp;
        }
        end = sp0.cp;
    }
    [[self hideCaret] setAutodisplay:NO];  // no need to display yet
    [self replaceSel:"\n" length:1];

    while ([self getSubstring:buf start:pos++ length:1] != -1){
        if(buf[0] == ' ' || buf[0] == '\t') [self replaceSel:buf length:1];
        else if(pos == end) break;
        else break;
    }

    [[self setAutodisplay:YES] displayIfNeeded];

    /* scroll down to the correct line */
    if ([superview respondsTo:@selector(rawScroll:)]) {
        [self scroll:0 :1];
        [self calcLine];
    }

    return self;
}

- match:(unsigned char *)LR
{
    NXRect oldRect, newRect;
    unsigned char buf[2];
    int count, left_pos, right_pos, utime;

    right_pos = sp0.cp;
    left_pos = right_pos-1;
    count = 1;
    utime = 100000;

    /*  don't do anything if there's a non-empty selection
     *  or not two character */

    if (sp0.cp != spN.cp || strlen(LR) != 2) return self;

    /* at the beginning of file ? */
    if (left_pos < 0){
        [self replaceSel:&LR[1] length:1];
        return self;
    }

    /* search for the left character */
    while([self getSubstring:buf start:left_pos length:1] != -1){
        if(buf[0] == LR[0]) count--;
        else if (buf[0] == LR[1])  count++;
        if(count == 0) break;
        if(left_pos-- == 0) break;
    }

    if(count != 0)  {
        [self replaceSel:&LR[1] length:1];
        return self;
    }


    [self goto:left_pos end:left_pos+1 mode:3];

    /* if our superview isn't a ClipView, no scrolling */
    if ([superview respondsTo:@selector(rawScroll:)]) {
        [superview getBounds:&oldRect];

        /* scroll to selection */
        [self scrollSelToVisible];
        [superview getBounds:&newRect];

        /* add some time for viewing if the text is scrolled */
        if(newRect.origin.y != oldRect.origin.y) utime +=300000;
    }

    [[self window] display];

    usleep(utime);

    [self goto:right_pos end:right_pos mode:0];

    /* scrollBack */
    if ([superview respondsTo:@selector(rawScroll:)]) {
        [self scrollTo:&oldRect.origin];
    }

    [self replaceSel:&LR[1] length:1];

    return self;
}

- insertKeyCombination:(NXEvent *)event
{
    char code[9];
    int code_len = 0;
    int cc = event->data.key.charCode;
    int f_digit = event->data.key.charCode >> 4;
    int s_digit = event->data.key.charCode & 0xf;
    if ((event->flags & NX_ALPHASHIFTMASK) &&
        !(event->flags & NX_SHIFTMASK))      code[code_len++] = 'l';
    if (event->flags & NX_SHIFTMASK)          code[code_len++] = 's';
    if (event->flags & NX_CONTROLMASK)        code[code_len++] = 'c';
    if (event->flags & NX_ALTERNATEMASK)      code[code_len++] = 'a';
    if (event->flags & NX_COMMANDMASK)        code[code_len++] = 'm';
    if (event->flags & NX_NUMERICPADMASK)     code[code_len++] = 'n';
    if (event->flags & NX_HELPMASK)           code[code_len++] = 'h';

    if(NXIsPrint(cc) && !NXIsSpace(cc) && !NXIsCntrl(cc)){
        /* should be able to print this character */
        code[code_len++] = 0x27; // '
        code[code_len++] = cc;
    }
    else if(NXIsCntrl(cc) && (event->flags & NX_CONTROLMASK)
            && !NXIsSpace(cc) && (cc <= 0x1F)){
        /* ordinary control character */
        code[code_len++] = 0x27; // '
        code[code_len++] = (event->flags & NX_SHIFTMASK) ? cc + 0x40: cc+ 0x60;
    }
    else{
        /* cannot print, replace with hex code */
        code[code_len++] = (f_digit < 10) ? '0' + f_digit: 'A'+ f_digit-10;
        code[code_len++] = (s_digit < 10) ? '0'+ s_digit : 'A'+ s_digit-10;
    }

    [self replaceSel:code length:code_len];
    return self;
}

- insertKeyCombOfNextKey
{
    static id action = nil;

    if (!action)
        action = [[XTEventMsgAction alloc]
                       initSel:@selector(insertKeyCombination:)];
    nextAction = action;
    return self;
}

@end
@implementation eText(private)

- scrollTo:(const NXPoint *)newOrigin
{
    // superview = ClipView
    // [superview superview] = ScrollView

    if([superview respondsTo:@selector(rawScroll:)]){
        [[superview constrainScroll:(NXPoint *)newOrigin] rawScroll:newOrigin];
        if ([[superview superview] respondsTo:@selector(reflectScroll:)]){
            [[superview superview] reflectScroll:superview];// romeo romeo
        }
    }
    return self;
}
@end
```