

```
//
// FILENAME: eText.Undo.m
// SUMMARY: Implementation of the Undo subsystem of eText
// CATEGORY: Undo
// PROTOCOLS: Uses UndoManager
// INTERFACE: None
// AUTHOR: Rohit Khare, portions by Jeff Martin of Bozell.
// COPYRIGHT: ©1993,94 California Institute of Technology, eText Project
//
// Implementation Comments
// These methods deal with undoManager. We may have some real problems
// intercepting the data flow between "user" methods and internal ones
// (e.g. paste vs replaceSelWithRTF:)
// Unlike Jeff's UndoText, we do ours with undo record grouping and
// and without writing new replace... 'Action' methods. At least, we'll try...
//
// History
// 10/27/94: Changed handling of typing undos to attempt coalescing.
// 10/17/94: Cleaned up for eText5.
// 08/05/94: Completely Rearchitected for 5.0. RK
//
// Imported Interfaces
//
// #import "eText.Undo.h"
//
static int undoLastPos = -1;
static int undoFirstPos = -1;
static Stream *undoStream = NULL; // RK: Assumes only 1 Text is being edited
static BOOL isTyping=NO;
#define DELETE_KEY (0x08)

NXTextFilterFunc oldTextFilter; // keep the old text filter around

@implementation eText(Undo)
//
// eText Undo API
//
- undoSelChange:(const char *)actionName { // Call before transaction
    if(sp0.cp < 0) return nil;
    [self registerLastTypingUndo]; // Clear the queue...

    if(sp0.cp != spN.cp) {
        undoStream = [[Stream alloc] init] openMemory];
        [self writeETF:[undoStream stream] from:sp0.cp to:spN.cp];
    } else {
        if(undoStream) [undoStream free];
        undoStream = NULL;
    }
}
```

[illegible]

[illegible]

```

    [self undoSelChange:"Subscript"];
    retval = [super subscript:sender];
    [self undoAffectedRange:sp0.cp to:spN.cp];
    return retval;
}
- superscript:sender {
    id retval;

    [self undoSelChange:"Superscript"];
    retval = [super superscript:sender];
    [self undoAffectedRange:sp0.cp to:spN.cp];
    return retval;
}
- unscript:sender {
    id retval;
    [self undoSelChange:"Unscript"];
    retval = [super unscript:sender];
    [self undoAffectedRange:sp0.cp to:spN.cp];
    return retval;
}

- setSelFont:font {
    id retval,newF;

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

    if (sp0.cp == spN.cp) // If the font is at a caret, just go on.
        return [super setSelFont:newF];

    [self undoSelChange:"Set Font"];
    retval = [super setSelFont:newF];
    [self undoAffectedRange:sp0.cp to:spN.cp];
    return retval;
}
- setSelFontSize:(float)size {
    id retval;

    [self undoSelChange:"Set Font Size"];
    retval = [super setSelFontSize:size];
    [self undoAffectedRange:sp0.cp to:spN.cp];
    return retval;
}

```

[illegible]

[illegible]

```

// If we're not typing yet, get cranking
if (!isTyping) {
    isTyping = YES;
    [undoManager setActionName:@"Typing"];
    [undoManager beginUndoRecordGrouping];
}

// This should never get called, since nuking one character already moves sp
//if ((start == end) && (text && (*text == DELETE_KEY))) {
//    [self setSel:start-1 :end];
//    start = self->sp0.cp, end = self->spN.cp;
//}
// No matter what, we're about to nuke a section of text.
// Note that we generate full RTF code for individual characters.
// Jeff's old code avoided that by snapshotting the _entire_ document...
if (start != end) {
    id stream = [[[Stream alloc] init] openMemory];
    [self writeETF:[stream stream] from:start to:end];
    [[[undoManager setUndoTarget:self] freeUndoArgs]
        replaceSelWith:stream from:start to:start];

    //[self replaceSel:@""]; // could use objc_msgSendSuper() here
    undoLastPos = start;
    undoFirstPos = start;
}

// if it was a delete key, we're done; otherwise we have to undo
// the subsequent addition of keystrokes
if (text && (*text != DELETE_KEY)) {
    [[undoManager setUndoTarget:self]
        replaceSel:@"" from:position to:(position + *len)];
    undoLastPos = position + *len;
    if (undoFirstPos == -1)
        undoFirstPos = position;
}
return ((char *)text);
}

```