```
/*
 *  Copyright (c) 1993 Christopher J. Kane.  All rights reserved.
 *
 *  This software is subject to the terms of the MiscKit license
 *  agreement.  Refer to the license document included with the
 *  MiscKit distribution for these terms.
 *
 *  Version: 1.2 (22 March 1994)
 */

#import "MiscSearchText.h"
#import "Kludges.subproj/regexpr.h"
#import "Kludges.subproj/MiscTBMK.h"

@implementation eText (SearchText)

- (oneway void)makeSelectionVisible
{
  [self scrollSelToVisible];
}

- (int)replaceAll:(const char *)pattern with:(const char *)replacement mode:
(SearchMode)mode regexpr:(BOOL)regexpr cases:(BOOL)cases
{
  unsigned char fm[256], tr[256];
  struct re_pattern_buffer rpat;
  Misc_TBMKpattern lpat=NULL;
  NXStream *in_strm=NULL, *out_strm=NULL;
  int s1=0, e1=0, s2=0, e2=0, ret_val, plen, rlen, pos, searchTextMaxSize;

  if (sp0.cp<0 && mode!=TextEdgeToTextEdge)
    return SEARCH_NO_SELECTION;
  if (![self isEditable])
    return SEARCH_CANNOT_WRITE;
  switch (mode)
    {
      case TextEdgeToTextEdge:
      case SelStartToSelStart:
      case SelEndToSelEnd: s1 = 0; e1 = textLength; break;
      case TextEdgeToSelStart: s1 = 0; e1 = sp0.cp; break;
      case TextEdgeToSelEnd: s1 = 0; e1 = spN.cp; break;
      case SelStartToSelEnd: s1 = sp0.cp; e1 = spN.cp-sp0.cp; break;
      case SelStartToTextEdge: s1 = sp0.cp; e1 = textLength-sp0.cp; break;
      case SelEndToTextEdge: s1 = spN.cp; e1 = textLength-spN.cp; break;
      case SelEndToSelStart: s1 = 0; e1 = sp0.cp; s2 = spN.cp; e2 = textLength-
spN.cp; break;
      default: return SEARCH_INVALID_ARGUMENT;
```

```
    }
  searchTextMaxSize = s1+e1;
  plen = strlen(pattern);
  rlen = strlen(replacement);
  if (regexpr)
    {
      char *str;
      int i;
      memset(&rpat, 0, sizeof(rpat));
      for(i=256; i--;)
        tr[i] = i;
      if (!cases)
        for(i='A'; i<='Z'; i++) tr[i] = i-'A'+'a';
      rpat.translate = tr;
      rpat.fastmap = fm;
      str = re_compile_pattern((char *)pattern, plen, &rpat);
      if (str!=NULL)
        return (strcmp(str, "Out of memory")?
SEARCH_INVALID_REGEXPR:SEARCH_INTERNAL_ERROR);
    }
  else
    {
      lpat = Misc_TBMKpattern_alloc(pattern, plen, 0, !cases);
      if (lpat==NULL)
        return SEARCH_INTERNAL_ERROR;
    }
  out_strm = NXOpenMemory(NULL, 0, NX_READWRITE);
  in_strm = NXOpenMemory(NULL, 0, NX_READWRITE);
  if (out_strm==NULL || in_strm==NULL)
    {
      ret_val = SEARCH_INTERNAL_ERROR;
      goto exit;
    }
  [self writeText:in_strm];
  NXSeek(in_strm, 0, NX_FROMSTART);
  ret_val = 0;
  NXWrite(out_strm, in_strm->buf_base, s1); /* get up to start of searching */
  /* invariant for searching: in_strm->buf_base+s1+e1 is the position _after_ the
last one searched */
 start_searching:
  if (regexpr)
    {
      while (!NXUserAborted() && e1>0 && (pos = re_search_pattern(&rpat, in_strm-
>buf_base, searchTextMaxSize, s1, e1, 0))>=0)
        {
          int len = re_match_pattern(&rpat, in_strm->buf_base, searchTextMaxSize,
pos, 0);
```

```
        if (len<0)
          {
            ret_val = SEARCH_INTERNAL_ERROR;
            goto exit;
          }
        NXWrite(out_strm, in_strm->buf_base+s1, pos-s1);
        NXWrite(out_strm, replacement, rlen);
        e1 -= (pos-s1+len);
        s1 = (pos+len);
        ret_val++;
      }
    if (pos==-2)
      {
        ret_val = SEARCH_INTERNAL_ERROR;
        goto exit;
      }
  }
else
  {
    while (!NXUserAborted() && e1>0 && Misc_TBMKsearch_memory(lpat, in_strm-
>buf_base+s1, e1, 0, &pos)>0)
      {
        NXWrite(out_strm, in_strm->buf_base+s1, pos);
        NXWrite(out_strm, replacement, rlen);
        s1 += (pos+plen);
        e1 -= (pos+plen);
        ret_val++;
      }
  }
if (NXUserAborted())
  {
    ret_val = SEARCH_ABORTED;
    goto exit;
  }
if (e1>0)
  NXWrite(out_strm, in_strm->buf_base+s1, e1); /* get remainder of searched
material */
if (e2!=0)
  {
    NXWrite(out_strm, in_strm->buf_base+s1+e1, s2-(s1+e1)); /* get gap */
    s1 = s2;
    e1 = e2;
    s2 = e2 = 0;
    goto start_searching;
  }
NXWrite(out_strm, in_strm->buf_base+s1+e1, textLength-(s1+e1)); /* after searched
to end */
```

```
  NXSeek(out_strm, 0, NX_FROMSTART);
  [self readText:out_strm];
  [self setSel:0 :0];
 exit:
  Misc_TBMKpattern_free(&lpat);
  if (in_strm!=NULL)
    NXCloseMemory(in_strm, NX_FREEBUFFER);
  if (out_strm!=NULL)
    NXCloseMemory(out_strm, NX_FREEBUFFER);
  return ret_val;
}

- (oneway void)replaceSelection:(const char *)replacement
{
  if ([self isEditable])
    [self replaceSel:replacement];
}

- (int)searchFor:(const char *)pattern mode:(SearchMode)mode reverse:(BOOL)rev
regexpr:(BOOL)regexpr cases:(BOOL)cases position:(out int *)pos size:(out int *)size
{
  unsigned char fm[256], tr[256];
  struct re_pattern_buffer rpat;
  Misc_TBMKpattern lpat=NULL;
  NXStream *in_strm=NULL;
  int s1=0, e1=0, s2=0, e2=0, ret_val, plen, p, position;

  if (sp0.cp<0 && mode!=TextEdgeToTextEdge)
    return SEARCH_NO_SELECTION;
  if (rev)
    switch (mode)
      {
        case TextEdgeToSelStart: s1 = textLength-1; e1 = sp0.cp-textLength; break;
        case TextEdgeToSelEnd: s1 = textLength-1; e1 = spN.cp-textLength; break;
        case TextEdgeToTextEdge: s1 = textLength-1; e1 = -textLength; break;
        case SelStartToSelEnd: s1 = sp0.cp-1; e1 = -sp0.cp+1; s2 = textLength-1; e2
= spN.cp-textLength+1; break;
        case SelStartToTextEdge: s1 = sp0.cp-1; e1 = -sp0.cp+1; break;
        case SelStartToSelStart: s1 = sp0.cp-1; e1 = -sp0.cp+1; s2 = textLength-1;
e2 = sp0.cp-textLength+1; break;
        case SelEndToTextEdge: s1 = spN.cp-1; e1 = -spN.cp+1; break;
        case SelEndToSelStart: s1 = spN.cp-1; e1 = sp0.cp-spN.cp+1; break;
        case SelEndToSelEnd: s1 = spN.cp-1; e1 = -spN.cp+1; s2 = textLength-1; e2 =
spN.cp-textLength+1; break;
        default: return SEARCH_INVALID_ARGUMENT;
      }
  else
```

```
    switch (mode)
      {
        case TextEdgeToSelStart: s1 = 0; e1 = sp0.cp; break;
        case TextEdgeToSelEnd: s1 = 0; e1 = spN.cp; break;
        case TextEdgeToTextEdge: s1 = 0; e1 = textLength; break;
        case SelStartToSelEnd: s1 = sp0.cp; e1 = spN.cp-sp0.cp; break;
        case SelStartToTextEdge: s1 = sp0.cp; e1 = textLength-sp0.cp; break;
        case SelStartToSelStart: s1 = sp0.cp; e1 = textLength-sp0.cp; s2 = 0; e2 =
sp0.cp; break;
        case SelEndToTextEdge: s1 = spN.cp; e1 = textLength-spN.cp;  break;
        case SelEndToSelStart: s1 = spN.cp; e1 = textLength-spN.cp; s2 = 0; e2 =
sp0.cp; break;
        case SelEndToSelEnd: s1 = spN.cp; e1 = textLength-spN.cp; s2 = 0; e2 =
spN.cp; break;
        default: return SEARCH_INVALID_ARGUMENT;
      }
  plen = strlen(pattern);
  if (regexpr)
    {
      char *str;
      int i;
      memset(&rpat, 0, sizeof(rpat));
      for(i=256; i--;)
        tr[i] = i;
      if (!cases)
        for(i='A'; i<='Z'; i++) tr[i] = i-'A'+'a';
      rpat.translate = tr;
      rpat.fastmap = fm;
      str = re_compile_pattern((char *)pattern, plen, &rpat);
      if (str!=NULL)
        return (strcmp(str, "Out of memory")?
SEARCH_INVALID_REGEXPR:SEARCH_INTERNAL_ERROR);
    }
  else
    {
      lpat = Misc_TBMKpattern_alloc(pattern, plen, rev, !cases);
      if (lpat==NULL)
        return SEARCH_INTERNAL_ERROR;
    }
  in_strm = NXOpenMemory(NULL, 0, NX_READWRITE);
  if (in_strm==NULL)
    {
      ret_val = SEARCH_INTERNAL_ERROR;
      goto exit;
    }
  [self writeText:in_strm];
  NXSeek(in_strm, 0, NX_FROMSTART);
```

```
 ret_val = 0;
start_searching:
 if (NXUserAborted())
   {
     ret_val = SEARCH_ABORTED;
     goto exit;
   }
 if (regexpr)
   {
     p = -1;
     if (s1>=0)
       p = re_search_pattern(&rpat, in_strm->buf_base, textLength, s1, e1, 0);
     if (p==-1 && e2!=0)
       {
         s1 = s2;
         e1 = e2;
         s2 = e2 = 0;
         goto start_searching;
       }
     if (p==-2)
       {
         ret_val = SEARCH_INTERNAL_ERROR;
         goto exit;
       }
     if (p>-1)
       {
         *pos = p;
         *size = re_match_pattern(&rpat, in_strm->buf_base, textLength, p, 0);
         if (*size<0)
           {
             ret_val = SEARCH_INTERNAL_ERROR;
             goto exit;
           }
         ret_val = 1;
       }
   }
 else
   {
     p = 0;
     if (s1>=0)
       p = Misc_TBMKsearch_memory(lpat, in_strm->buf_base+s1, e1, 0, &position);
     if (p<0)
       {
         ret_val = SEARCH_INTERNAL_ERROR;
         goto exit;
       }
     if (p==0 && e2!=0)
```

```
        {
          s1 = s2;
          e1 = e2;
          s2 = e2 = 0;
          goto start_searching;
        }
      if (p>0)
        {
          *pos = position+s1;
          *size = plen;
          ret_val = 1;
        }
    }
 exit:
  Misc_TBMKpattern_free(&lpat);
  if (in_strm!=NULL)
    NXCloseMemory(in_strm, NX_FREEBUFFER);
  return ret_val;
}

- (oneway void)selectTextFrom:(int)start to:(int)end
{
  if ([self isSelectable] && start<=end && 0<=start)
    [self setSel:start :end];
}

- (int)setRegExprSyntax:(int)syntax
{
  return re_set_syntax(syntax);
}

////////////////////////////////////////////////////////////////////////
// RK, 10/22/94 Changed protocol name to use writeSearchableSelection....//
////////////////////////////////////////////////////////////////////////
- (void)writeSearchableSelectionToPasteboard:(in Pasteboard *)pboard asType:(in
NXAtom)type
{
  char text[spN.cp-sp0.cp+1];
  [self getSubstring:text start:sp0.cp length:spN.cp-sp0.cp];
  text[spN.cp-sp0.cp] = '\0';
  if (*text!='\0')
    {
      [pboard declareTypes:&type num:1 owner:NULL];
      [pboard writeType:type data:text length:spN.cp-sp0.cp];
    }
}
```

```
- (BOOL)findText:(const char *)string ignoreCase:(BOOL)ignoreCaseflag backwards:
(BOOL)backwardsflag wrap:(BOOL)wrapflag
{
        Misc_TBMKpattern pat;
        int s1, e1, s2, e2, mp, pos, r;

        s1 = s2 = e1 = e2 = 0;
        if (backwardsflag && sp0.cp<0)
                {s1 = textLength; e1 = -textLength;}
        else if (backwardsflag)
                {s1 = sp0.cp; e1 = -sp0.cp; s2 = textLength; e2 = spN.cp-
textLength;}
        else if (sp0.cp<0)
                {s1 = 0; e1 = textLength;}
        else
                {s1 = spN.cp; e1 = textLength-spN.cp; s2 = 0; e2 = sp0.cp;}
        [self stream];
        if (textStream==0)
                return NO;
        pat = Misc_TBMKpattern_alloc(string, strlen(string), backwardsflag,
ignoreCaseflag);
        if (pat==0)
                return NO;
        NXSeek(textStream, s1, NX_FROMSTART);
        r = Misc_TBMKsearch_stream(pat, textStream, e1, 0, &mp);
        pos = mp+s1;
        if (!r && e2>0 && wrapflag) {
                NXSeek(textStream, s2, NX_FROMSTART);
                r = Misc_TBMKsearch_stream(pat, textStream, e2, 0, &mp);
                pos = mp+s2;
        }
        Misc_TBMKpattern_free(&pat);
        if (!r)
                return NO;
        [self setSel:pos :pos+strlen(string)];
        [self scrollSelToVisible];
        return YES;
}
@end
```