```
/*
 *  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: 1.1.1 (14 December 1993)
 *
 *  Incorporates a bug-fix by Annard Brouwer <annard@stack.urc.tue.nl>.
 *  The nextText outlet of the findTextField was remaining set to the
 *  replaceTextField.   When the replace controls were hidden, this
 *  caused the Tab key in the findTextField to appear to do nothing.
 *  The text in the field is now correctly selected when the Tab key
 *  is pressed.
 *
 *  Rohit Khare: 10/29/94:  rangePanel added
 *  Added a line/character count panel as in Edit. This is a local hack that
 *  relies on talking to a Text, not a searchableText, so I don't know how
 *  generic it is. The code is almost completely borrowed from:
        Find Manager

        copyright: NOT! Feel free to use any of the source to Find Manager.

        David Holscher
        Box #727
        Rose-Hulman Institute of Technology
        Terre Haute, IN 47803

        e-mail: holschdm@next-work.rose-hulman.edu
 */

#import "MiscFindPanel.h"
#import "SearchableText.h"

#define DURING                  NX_DURING
#define HANDLER                 NX_HANDLER switch (NXLocalHandler.code) {
#define ENDHANDLER              default: NXLogError("Uncaught exception: %d," \
                                " in %s:%s.\n", NXLocalHandler.code, __FILE__, \
                                sel_getName(_cmd)); abort();} NX_ENDHANDLER
#define IGNORE(E)   case E: NXLogError("%s exception in %s:%s. " \
                                "Ignored.\n", #E, __FILE__, sel_getName(_cmd)); break

#define LocalString(K)          NXLoadLocalizedStringFromTableInBundle(NULL, \
                                [NXBundle bundleForClass:[self class]], K, "")

static MiscFindPanel *_sharedFindPanel=nil;
static Pasteboard *_findPb=nil;
static Object *_firstConformer=nil;
static int _pbChangeCount=0;
static BOOL _replEnabled=YES;
static BOOL _useFindPb=YES;
static BOOL _allocationOK=NO;
static BOOL _resizeOK=NO;
static BOOL _fpLoaded=NO;
```

```
@implementation MiscFindPanel

- _calcFindPanelTarget
{
  id i;
  if (_firstConformer!=nil)
    return _firstConformer;
  if (self!=[NXApp keyWindow])
    {
      for(i=[[NXApp keyWindow] firstResponder]; i; i = [i nextResponder])
        if ([i conformsTo:@protocol(SearchableText)])
          return i;
      i = [[NXApp keyWindow] delegate];
      if ([i conformsTo:@protocol(SearchableText)])
        return i;
    }
  if ([NXApp keyWindow]!=[NXApp mainWindow])
    {
      for(i=[[NXApp mainWindow] firstResponder]; i; i = [i nextResponder])
        if ([i conformsTo:@protocol(SearchableText)])
          return i;
      i = [[NXApp mainWindow] delegate];
      if ([i conformsTo:@protocol(SearchableText)])
        return i;
    }
  i = NXApp;
  if ([i conformsTo:@protocol(SearchableText)])
    return i;
  i = [NXApp delegate];
  if ([i conformsTo:@protocol(SearchableText)])
    return i;
  return nil;
}

- (void)_getFindPbData
{
  if (_pbChangeCount==[_findPb changeCount])
    return;
  DURING
    if ([_findPb findAvailableTypeFrom:&NXAsciiPboardType num:1])
      {
        char *text;
        int len;
        if ([_findPb readType:NXAsciiPboardType data:&text length:&len])
          {
            [findTextField setStringValue:text];
            [_findPb deallocatePasteboardData:text length:len];
          }
      }
  HANDLER
    IGNORE(NX_pasteboardComm);
    IGNORE(NX_appkitVMError);
  ENDHANDLER
  _pbChangeCount = [_findPb changeCount];
}
```

```
- (void)_modifyFindPanel
{
   NXRect r;
   [contentView getBounds:&r];
   [self disableFlushWindow];
   if (_replEnabled)
      {
         _resizeOK = YES;
         [super sizeWindow:NX_WIDTH(&r) :NX_HEIGHT(&r)+25.0];
         _resizeOK = NO;
         [[[findTextField superview] superview] moveBy:0.0 :25.0];
         [contentView addSubview:_replBox :NX_BELOW relativeTo:nil];
         [findTextField setNextText:replaceTextField];       /* added by Annard */
      }
   else
      {
         [_replBox removeFromSuperview];
         [[[findTextField superview] superview] moveBy:0.0 :-25.0];
         _resizeOK = YES;
         [super sizeWindow:NX_WIDTH(&r) :NX_HEIGHT(&r)-25.0];
         _resizeOK = NO;
         [findTextField setNextText:findTextField];            /* added by Annard */
      }
   [self display];
   [[self reenableFlushWindow] flushWindowIfNeeded];
}

- (void)_resetFindPanel
{
   [messageTextField setStringValue:""];
   [findTextField selectText:nil];
}

- (void)_setFindPbData
{
   const char *text;
   if (_pbChangeCount==[_findPb changeCount])
      return;
   text = [findTextField stringValue];
   if (*text=='\0')
      return;
   DURING
      _pbChangeCount = [_findPb declareTypes:&NXAsciiPboardType num:1 owner:NULL];
      [_findPb writeType:NXAsciiPboardType data:text length:(int)strlen(text)];
   HANDLER
      IGNORE(NX_pasteboardComm);
   ENDHANDLER
}

+ setFindPbEnabled:(BOOL)aBool
{
   _useFindPb = aBool;
   return self;
}

+ setFirstConformer:aConformer
```

```
{
  id formerConformer = _firstConformer;
  if (aConformer==nil || [aConformer conformsTo:@protocol(SearchableText)])
    _firstConformer = aConformer;
  return formerConformer;
}

+ setReplacementEnabled:(BOOL)aBool
{
  if (aBool!=_replEnabled)
    {
      _replEnabled = aBool;
      [_sharedFindPanel _modifyFindPanel];
    }
  return self;
}

+ sharedInstance
{
  if (_sharedFindPanel==nil && !_fpLoaded)
    {
      char path[MAXPATHLEN+16];
      _fpLoaded = YES;
      _findPb = [Pasteboard newName:NXFindPboard];
      _allocationOK = YES;
      if ([[NXBundle bundleForClass:self] getPath:path forResource:"FindPanel.nib" ofType:NULL])
       {
            //NXLogError("About to loadNibFile %s", path);
            [NXApp loadNibFile:path owner:NXApp withNames:YES];
             _sharedFindPanel = self = NXGetNamedObject("sharedFindPanel", NXApp);
            // The above is undocumented behavior and may go haywire.
         }
      _allocationOK = NO;
      if (_sharedFindPanel==nil) {
            //NXLogError("About to run alert");
      //NXRunAlertPanel(LocalString("LOAD_ERR"), LocalString("LOAD_ERR_MSG"), NULL, NULL, NULL);
        NXRunAlertPanel("LOAD_ERR", "LOAD_ERR_MSG", NULL, NULL, NULL);
      } else {
          if (!_replEnabled)
            [_sharedFindPanel _modifyFindPanel];
              [_sharedFindPanel setExcludedFromWindowsMenu:YES];
              [rangePanel setExcludedFromWindowsMenu:YES];
              [_sharedFindPanel useOptimizedDrawing:YES];
        [_sharedFindPanel setFrameAutosaveName:"etFindPanel"];
            [_sharedFindPanel setFrameUsingName:"etFindPanel"];
        [rangePanel setFrameAutosaveName:"etRangePanel"];
            [rangePanel setFrameUsingName:"etRangePanel"];
           [self _resetFindPanel];
      }
    }
  return _sharedFindPanel;
}
```

```
- enterSelection:sender
{
   [[self _calcFindPanelTarget] writeSearchableSelectionToPasteboard:_findPb asType:NXAsciiPboardType];
   [self _getFindPbData];
   [self _resetFindPanel];
   return self;
}

- findBackward:sender
{
   [self _resetFindPanel];
   if (*[findTextField stringValue]=='\0' && _useFindPb)
      [self _getFindPbData];
   if (*[findTextField stringValue]=='\0')
      [self makeKeyAndOrderFront:nil];
   else
      {
         int pos, size, result;
         id target = [self _calcFindPanelTarget];
         if (_useFindPb)
            [self _setFindPbData];
         result = [target searchFor:[findTextField stringValue] mode:SelStartToSelEnd reverse:YES regexpr:[regExprButton state] cases:!
[ignoreCaseButton state] position:&pos size:&size];
         if (target==nil || result<0)
            {
               [messageTextField setStringValue:LocalString("INVALID_OP")];
               NXBeep();
            }
         else if (result==0)
            {
               [messageTextField setStringValue:LocalString("NOT_FOUND")];
               NXBeep();
            }
         else
            {
               [target selectTextFrom:pos to:pos+size];
               [target makeSelectionVisible];
            }
      }
   return self;
}

- findForward:sender
{
   [self _resetFindPanel];
   if (*[findTextField stringValue]=='\0' && _useFindPb)
      [self _getFindPbData];
   if (*[findTextField stringValue]=='\0')
      [self makeKeyAndOrderFront:nil];
   else
      {
         int pos, size, result;
         id target = [self _calcFindPanelTarget];
         if (_useFindPb)
            [self _setFindPbData];
         result = [target searchFor:[findTextField stringValue] mode:SelEndToSelStart reverse:NO regexpr:[regExprButton state] cases:!
```

```
[ignoreCaseButton state] position:&pos size:&size];
        if (target==nil || result<0)
          {
            [messageTextField setStringValue:LocalString("INVALID_OP")];
            NXBeep();
          }
        else if (result==0)
          {
            [messageTextField setStringValue:LocalString("NOT_FOUND")];
            NXBeep();
          }
        else
          {
            [target selectTextFrom:pos to:pos+size];
            [target makeSelectionVisible];
          }
      }
   return self;
}

- jumpToSelection:sender
{
   [[self _calcFindPanelTarget] makeSelectionVisible];
   return self;
}

- replace:sender
{
   [self _resetFindPanel];
   [[self _calcFindPanelTarget] replaceSelection:[replaceTextField stringValue]];
   return self;
}

- replaceAll:sender
{
   id target;
   int count;
   [self _resetFindPanel];
   if (*[findTextField stringValue]=='\0')
      return self;
   target = [self _calcFindPanelTarget];
   count = [target replaceAll:[findTextField stringValue] with:[replaceTextField stringValue] mode:([scopeMatrix selectedRow]?
SelStartToSelEnd:TextEdgeToTextEdge) regexpr:[regExprButton state] cases:![ignoreCaseButton state]];
   if (count<0 || target==nil)
      {
        [messageTextField setStringValue:LocalString("INVALID_OP")];
        NXBeep();
      }
   else
      {
        char buffer[256];
        sprintf(buffer, LocalString("N_REPLACED"), count);
        [messageTextField setStringValue:buffer];
        [target makeSelectionVisible];
      }
   return self;
```

```
}

- replaceAndFind:sender
{
  [[self replace:sender] findForward:sender];
  return self;
}

- orderFrontRangePanel:sender {
    id text;
    [rangePanel makeKeyAndOrderFront:self];

    if (((text =[self _calcFindPanelTarget]) == nil) ||
        (![text isKindOf:[Text class]]))
        return self;
    else {
        NXSelPt start, end;
        char buf[20];

        [text getSel:&start :&end];

        switch(rangeType)
        {
            case line :
                sprintf(buf, "%d", [text lineFromPosition:start.c1st]);
                if(start.c1st != end.c1st)
                {
                    sprintf((buf + strlen(buf)), ":%d",
                        [text lineFromPosition:end.c1st]);
                }
                break;
            case character :
                sprintf(buf, "%d", start.cp);
                if(start.cp != end.cp)
                {
                    sprintf((buf + strlen(buf)), ":%d", end.cp);
                }
                break;
        }
        [rangeField setStringValue:buf];
        [rangeField selectText:self];
    }

    return self;
}

- jumpToRange:sender {
    id text;
```

```
    [rangePanel makeKeyAndOrderFront:self];

    if (((text =[self _calcFindPanelTarget]) == nil) ||
        (![text isKindOf:[Text class]]))
        return self;
    else {
        char *test;
        int end;
        const char *buf = [rangeField stringValue];
        int start = atoi(buf);
        if((test = strchr(buf, ':')) != NULL)
            end = atoi(test + 1);
        else
            end = start;

        switch(rangeType)
        {
            case line :
            {
                [text setSel:[text positionFromLine:start]
                    :[text positionFromLine:end + 1]];
                break;
            }
            case character :
            {
                [text setSel:start :end];
                break;
            }
        }
    }
    [rangePanel orderOut:self];
    [text scrollSelToVisible];

    return self;
}
- setRangeType:sender {
    rangeType = [sender selectedTag];
    return [self orderFrontRangePanel:sender];
}
```

```
- (TextField *)findTextField
{
  return findTextField;
}

- (Button *)ignoreCaseButton
{
  return ignoreCaseButton;
}
```

```
- (TextField *)messageTextField
{
   return messageTextField;
}

- (Button *)regExprButton
{
   return regExprButton;
}

- (TextField *)replaceTextField
{
   return replaceTextField;
}

- (Matrix *)scopeMatrix
{
   return scopeMatrix;
}

- rangePanel {return rangePanel;}

- textDidEnd:sender endChar:(unsigned short)theChar
{
   if (theChar==NX_RETURN)
      [self orderOut:nil];
   return self;
}

- (BOOL)textWillChange:sender
{
   return NO;
}

- (BOOL)textWillEnd:sender
{
   return NO;
}

+ (BOOL)_canAlloc
{
   return _allocationOK;
}

+ alloc
{
   if (!_allocationOK)
      return [self doesNotRecognize:_cmd];
   return class_createInstanceFromZone(self, 0, NXDefaultMallocZone());
}

+ allocFromZone:(NXZone *)zone
{
   if (!_allocationOK)
```

```
    return [self doesNotRecognize:_cmd];
  if (zone!=NX_NOZONE)
    return class_createInstanceFromZone(self, 0, zone);
  return nil;
}

+ (BOOL)instancesRespondTo:(SEL)aSel
{
  if (aSel==@selector(copy) ||
      aSel==@selector(copyFromZone:) ||
      aSel==@selector(free) ||
      aSel==@selector(init) ||
      aSel==@selector(initContent:style:backing:buttonMask:defer:) ||
      aSel==@selector(initContent:style:backing:buttonMask:defer:screen:) ||
      aSel==@selector(miniaturize:) ||
      aSel==@selector(placeWindow:) ||
      aSel==@selector(placeWindow:screen:) ||
      aSel==@selector(placeWindowAndDisplay:) ||
      aSel==@selector(setDocEdited:) ||
      aSel==@selector(sizeWindow::))
    return NO;
  return [super instancesRespondTo:aSel];
}

+ new
{
  return [self doesNotRecognize:_cmd];
}

+ newContent:(const NXRect *)contentRect style:(int)aStyle backing:(int)bufferingType buttonMask:(int)mask defer:(BOOL)flag
{
  return [self doesNotRecognize:_cmd];
}

+ newContent:(const NXRect *)contentRect style:(int)aStyle backing:(int)bufferingType buttonMask:(int)mask defer:(BOOL)flag screen:
(const NXScreen *)screen
{
  return [self doesNotRecognize:_cmd];
}

- copyFromZone:(NXZone *)zone
{
  return self;
}

- free
{
  return self;
}

- init
{
  return [self doesNotRecognize:_cmd];
}

- initContent:(const NXRect *)contentRect style:(int)aStyle backing:(int)bufferingType buttonMask:(int)mask defer:(BOOL)flag
```

```
{
   if (!_allocationOK)
      return [self doesNotRecognize:_cmd];
    rangeType = line;
   return [super initContent:contentRect style:aStyle backing:bufferingType buttonMask:mask defer:flag];
}

- initContent:(const NXRect *)contentRect style:(int)aStyle backing:(int)bufferingType buttonMask:(int)mask defer:(BOOL)flag screen:
(const NXScreen *)screen
{
   return [self doesNotRecognize:_cmd];
}

- miniaturize:sender
{
   return self;
}

- orderWindow:(int)place relativeTo:(int)otherWin
{
   if (place!=NX_OUT)
      {
         if (_useFindPb)
            [self _getFindPbData];
         [self _resetFindPanel];
      }
   return [super orderWindow:place relativeTo:otherWin];
}

- placeWindow:(const NXRect *)aRect
{
   if (_resizeOK)
      return [super placeWindow:aRect];
   return [self moveTo:NX_X(aRect) :NX_Y(aRect)];
}

- placeWindow:(const NXRect *)aRect screen:(const NXScreen *)aScreen
{
   if (_resizeOK)
      return [super placeWindow:aRect screen:aScreen];
   return [self moveTo:NX_X(aRect) :NX_Y(aRect) screen:aScreen];
}

- placeWindowAndDisplay:(const NXRect *)aRect
{
   if (_resizeOK)
      return [super placeWindowAndDisplay:aRect];
   return [[self moveTo:NX_X(aRect) :NX_Y(aRect)] display];
}

- read:(NXTypedStream *)stream
{
   [super read:stream];
   findTextField = NXReadObject(stream);
   replaceTextField = NXReadObject(stream);
   messageTextField = NXReadObject(stream);
```

```
  ignoreCaseButton = NXReadObject(stream);
  regExprButton = NXReadObject(stream);
  scopeMatrix = NXReadObject(stream);
  _replBox = NXReadObject(stream);
  return self;
}

- (BOOL)respondsTo:(SEL)aSel
{
  if (aSel==@selector(alloc) ||
      aSel==@selector(allocFromZone:) ||
      aSel==@selector(new) ||
      aSel==@selector(newContent:style:backing:buttonMask:defer:) ||
      aSel==@selector(newContent:style:backing:buttonMask:defer:screen:) ||
      aSel==@selector(copy) ||
      aSel==@selector(copyFromZone:) ||
      aSel==@selector(free) ||
      aSel==@selector(init) ||
      aSel==@selector(initContent:style:backing:buttonMask:defer:) ||
      aSel==@selector(initContent:style:backing:buttonMask:defer:screen:) ||
      aSel==@selector(miniaturize:) ||
      aSel==@selector(placeWindow:) ||
      aSel==@selector(placeWindow:screen:) ||
      aSel==@selector(placeWindowAndDisplay:) ||
      aSel==@selector(setDocEdited:) ||
      aSel==@selector(sizeWindow::))
    return NO;
  return [super respondsTo:aSel];
}

- setDocEdited:(BOOL)aBool
{
  return self;
}

- sizeWindow:(float)x :(float)y
{
  return self;
}

- write:(NXTypedStream *)stream
{
  [super write:stream];
  NXWriteObjectReference(stream, findTextField);
  NXWriteObjectReference(stream, replaceTextField);
  NXWriteObjectReference(stream, messageTextField);
  NXWriteObjectReference(stream, ignoreCaseButton);
  NXWriteObjectReference(stream, regExprButton);
  NXWriteObjectReference(stream, scopeMatrix);
  NXWriteObjectReference(stream, _replBox);
  return self;
}

@end
```