```
 1
 2  // CTextDocView.cpp : implementation of the CTextDocView class
 3  //
 4
 5  #include "pch.h"
 6  #include "framework.h"
 7  // SHARED_HANDLERS can be defined in an ATL project implementing preview, thumbnail
 8  // and search filter handlers and allows sharing of document code with that project.
 9  #ifndef SHARED_HANDLERS
10  #include "DesignArk.h"
11  #endif
12
13  #include "CTextDocument.h"
14  #include "CTextDocView.h"
15
16  #include "CTextEditorObject.h"
17
18  #ifdef _DEBUG
19  #define new DEBUG_NEW
20  #endif
21
22
23  // CTextDocView
24  IMPLEMENT_DYNCREATE(CTextDocView, CScrollView)
25
26  BEGIN_MESSAGE_MAP(CTextDocView, CScrollView)
27
28      ON_WM_ERASEBKGND()
29
30      ON_WM_CREATE()
31      ON_WM_SETCURSOR()
32      ON_WM_SIZE()
33
34      // Standard printing commands
35      ON_COMMAND(ID_FILE_PRINT, &CScrollView::OnFilePrint)
36      ON_COMMAND(ID_FILE_PRINT_DIRECT, &CScrollView::OnFilePrint)
37      ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CTextDocView::OnFilePrintPreview)
38
39      ON_WM_HSCROLL()
40
41      ON_WM_LBUTTONUP()
42      ON_WM_LBUTTONDOWN()
43      ON_WM_LBUTTONDBLCLK()
44      ON_WM_MOUSEMOVE()
45      ON_WM_MOUSEWHEEL()
46      ON_WM_RBUTTONUP()
47      ON_WM_RBUTTONDOWN()
48
49      ON_WM_KEYDOWN()
50      ON_WM_KEYUP()
51
52      ON_WM_CONTEXTMENU()
53
```

```
54      ON_COMMAND(ID_EDIT_COPY, &CTextDocView::OnCopy)
55      ON_COMMAND(ID_EDIT_PASTE, &CTextDocView::OnPaste)
56      ON_COMMAND(ID_EDIT_CUT, &CTextDocView::OnCut)
57
58  END_MESSAGE_MAP()
59
60  // CTextDocView construction/destruction
61
62  CTextDocView::CTextDocView() noexcept
63  {
64      SetScrollSizes(MM_TEXT, CSize(0, 0));
65      this->rgn = new CRgn();
66      this->recentBlockLine = 0;
67
68      this->printIterator = 0;
69      this->returnNewLines = 0;
70      this->recentPrintZoom = theApp.zoom;
71      this->recentHlght = FALSE;
72      this->printing = FALSE;
73  }
74  CTextDocView::~CTextDocView()
75  {
76      delete this->rgn;
77  }
78
79  // Public attributes
80  CTextDocument* CTextDocView::GetDocument() const // non-debug version is inline
81  {
82      ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDocument)));
83      return (CTextDocument*)m_pDocument;
84  }
85  CSize CTextDocView::GetDocSize()
86  {
87      int sizeX = 0, sizeY = 0;
88
89      for (int i = 0; i < this->GetDocument()->objects.GetSize(); i++) {
90
91          if (sizeX < this->GetDocument()->objects[i]->getLineTextWidth()) {
92              sizeX = this->GetDocument()->objects[i]->getLineTextWidth();
93          }
94
95          sizeY += this->GetDocument()->objects[i]->getBounds().Height() + (this->GetDocument()->objects[i]->getBoxHeight());
96      }
97
98      CRect client;
99      GetClientRect(&client);
100
101     sizeX += (client.Width() * 0.1) + static_cast<int>(this->GetDocument()->lineOffset * theApp.zoom);
102
```

```
103     sizeY += client.Height() -
104             2*(this->GetDocument()->objects[this->GetDocument()->objects.GetUpperBound()]->getBoxHeight());
105
106     return CSize(sizeX, sizeY);
107 }
108
109 int CTextDocView::getActiveEditor()
110 {
111     return this->activeEditor;
112 }
113
114 // Public implementations
115 #ifdef _DEBUG
116 void CTextDocView::AssertValid() const
117 {
118     CView::AssertValid();
119 }
120 void CTextDocView::Dump(CDumpContext& dc) const
121 {
122     CView::Dump(dc);
123 }
124 #endif //_DEBUG
125 void CTextDocView::refresh(BOOL caret, BOOL window)
126 {
127     if (caret) {
128         this->CreateSolidCaret(this->GetDocument()->objects[this->activeEditor]->getBoxHeight() * 0.05625, this->GetDocument()->objects[this->activeEditor]->getBoxHeight() * 0.8);
129     }
130     if (window) {
131         this->updateWindow(FALSE, TRUE);
132     }
133 }
134
135 // Protected implementations
136 void CTextDocView::updateWindow(BOOL caret, BOOL window)
137 {
138     if (window) {
139         CRect client;
140         this->GetClientRect(client);
141         this->RedrawWindow(client);
142     }
143     else if (caret) {
144         this->SetCaretPos(this->GetDocument()->objects[this->activeEditor]->getCaretPoint(CSize(this->GetDocument()->objects[this->activeEditor]->getBoxHeight() * 0.05, this->GetDocument()->objects[this->activeEditor]->getBoxHeight() * 0.8)) - CSize(this->GetScrollPosition()));
145         this->ShowCaret();
146     }
147 }
148
149 void CTextDocView::ResyncScrollSizes(BOOL docSize, BOOL reposition)
```

```
150 {
151     CSize sizeDoc;
152     if (docSize) {
153         // Primary stuff with scroll bars
154         CClientDC dc(NULL);
155         OnPrepareDC(&dc);
156         sizeDoc = this->GetDocSize();
157         dc.LPtoDP(&sizeDoc);
158         SetScrollSizes(MM_TEXT, sizeDoc);
159     }
160
161     // Secondary stuff with objects and realignment
162     CRect client;
163     GetClientRect(&client); // Find current window size
164
165     BOOL redraw = FALSE;
166
167     int tempX = this->GetScrollPosition().x;
168
169     // Is the position of the cursor on the screen
170     int activeRight = this->GetDocument()->objects[this->activeEditor]->getBounds().left + static_cast<int>(this->GetDocument()->lineOffset * theApp.zoom) + this->GetDocument()->objects[this->activeEditor]->getCaretPos() * this->GetDocument()->objects[this->activeEditor]->getTextExtent().cx + 7;
171
172     // If the position of the cursor is beyond the edge of the window...
173     if (tempX + client.Width() < activeRight && client.Width() > 0) {
174
175         POINT scrollpt;
176         scrollpt.x = activeRight - client.Width() + 5; // Move the X to where the scroll is
177         scrollpt.y = this->GetScrollPosition().y; // Keep Y the same
178
179         this->ScrollToPosition(scrollpt);
180         redraw = TRUE; // We need to redraw the window now
181
182         for (int i = 0; i < this->GetDocument()->objects.GetSize(); i++) { // Resize all the objects so that the text fits inside
183
184             this->GetDocument()->objects[i]->OnSize(0, this->GetScrollPosition().x + client.Width(), this->GetScrollPosition().y + client.Height());
185         }
186     }
187
188     // Is the upper bounds of the active line
189     int activeBottom = this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).top;
190
```

```
191
192     if (this->GetScrollPosition().y + client.Height() <
          activeBottom && client.Height() > 0) {
193
194         POINT pt;
195         pt.x = this->GetScrollPosition().x;
196         pt.y = activeBottom - client.Height();
197
198         this->ScrollToPosition(pt);
199         redraw = TRUE;
200     } // If the cursor is under the screen, scroll down
201
202     if (!docSize) {
203         if (this->GetScrollPosition().x > activeRight &&
              client.Width() > 0) {
204
205             POINT pt;
206             pt.x = activeRight - this->GetDocument()->lineOffset -
                  7;
207             pt.y = this->GetScrollPosition().y;
208
209             this->ScrollToPosition(pt);
210             redraw = TRUE;
211         }
212         if (this->GetScrollPosition().y > activeBottom - this-
              >GetDocument()->defBoxHeight && client.Height() > 0) {
213
214             POINT pt;
215             pt.x = this->GetScrollPosition().x;
216             pt.y = activeBottom - this->GetDocument()-
                  >defBoxHeight;
217
218             this->ScrollToPosition(pt);
219             redraw = TRUE;
220         }
221     }
222
223     if (reposition) {
224
225         this->OnSize(0, client.Width(), client.Height());
226
227         CPoint point;
228         point.x = 0;
229         point.y = 0;
230
231         for (int i = 0; i < this->GetDocument()->objects.GetSize();
                i++) {
232
233             this->GetDocument()->objects[i]->setPosition(point);
234             point.y = this->GetDocument()->objects[i]->getBounds
                  ().bottom + this->GetDocument()->objects[i]-
                  >getBoxHeight();
235         }
236     }
237
238     if (tempX != this->GetScrollPosition().x) {
```

```
239             this->expandLineBounds();
240         }
241
242         this->updateWindow(FALSE, redraw);
243     }
244
245     // Protected overrides
246     void CTextDocView::OnDraw(CDC* pDc)
247     {
248         CTextDocument* pDoc = this->GetDocument();
249         ASSERT_VALID(pDoc);
250         if (!pDoc)
251             return;
252
253         if (!this->printing) {
254             this->fFont.CreateFont(static_cast<int>(pDoc->defBoxHeight
                  * theApp.zoom), 0, 0, 0, FW_NORMAL, FALSE, FALSE, 0,
                  ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
                  DEFAULT_QUALITY, 0, theApp.sFont);
255             pDc->SelectObject(&this->fFont);
256
257         } // If not printing, create a font here
258
259         BOOL finished = FALSE;
260         int i = 0; // Iterator
261         if (this->printing) { // If we are printing
262             i = this->printIterator; // Set the printIterator to where
                  we left off
263         }
264
265
266         while (!finished && i < pDoc->objects.GetSize()) {
267
268             if (this->printing) { // If we are printing, do this
269
270                 int bottom = std::get<0>(pDoc->objects[i]-
                      >getPrintBounds(returnNewLines, this->pInfo-
                      >m_rectDraw.Width())).bottom; // TODO : Remove this
                      bit
271
272                 if (bottom < this->pInfo->m_rectDraw.bottom) { // If
                      the object will completley fit on the print page
273
274                     this->printIterator++;
275                     returnNewLines = pDoc->objects[i]->draw(pDc, CSize
                       (), this->GetScrollPosition().x, returnNewLines,
                       this->printing, this->pInfo->m_rectDraw); // Draw
                       object on page as normal
276                 }
277                 else { // If not, ie is on the next page in any way
278
279                     // If the object should be printed on the next page
                        completley
280                     if (std::get<0>(pDoc->objects[i]->getPrintBounds
                       (returnNewLines, this->pInfo->m_rectDraw.Width
                       ())).top + pDoc->objects[i]->getBoxHeight() >=
```

```
                    this->pInfo->m_rectDraw.bottom) {
281
282                     pDoc->objects[i]->setPosition(CPoint(this->GetDocument()->objects[i]->getBounds().left, this->pInfo->m_rectDraw.top - (pDoc->objects[i]->getPrintLine() * pDoc->objects[i]->getBoxHeight())));
283
284                 }
285                 else { // If part of the object should be printed on the current page, and the rest on the next
286
287                     returnNewLines = pDoc->objects[i]->draw(pDc, CSize(), this->GetScrollPosition().x, returnNewLines, this->printing, this->pInfo->m_rectDraw);
288
289                     pDoc->objects[i]->setPosition(CPoint(pDoc->objects[i]->getBounds().left, this->pInfo->m_rectDraw.top - ((pDoc->objects[i]->getPrintLine() + returnNewLines) * pDoc->objects[i]->getBoxHeight())));
290                 }
291
292                 for (int j = i + 1; j < pDoc->objects.GetSize(); j++) {
293                     pDoc->objects[j]->move(0, pDoc->objects[j - 1]->getBounds().bottom + pDoc->objects[j - 1]->getBoxHeight() - pDoc->objects[j]->getBounds().top);
294                 } // Move the objects relative to the previous object
295
296
297                 this->drawHeader(pDc); // Draw the header for the next page
298
299                 finished = TRUE;
300             }
301         }
302         else {
303             returnNewLines = pDoc->objects[i]->draw(pDc, CSize(), this->GetScrollPosition().x, this->printing, returnNewLines);
304         }
305
306         i++;
307     }
308
309     if (this->printing) {
310
311         if (i == pDoc->objects.GetSize()) {
312             this->drawHeader(pDc);
313         }
314     }
315
```

```
316
317     if (!this->printing) {
318         this->updateWindow(TRUE, FALSE); // DO NOT CHANGE SECOND
              PARAMETER TO TRUE!!
319         this->fFont.DeleteObject();
320     }
321
322     //pDc->FrameRgn(this->rgn, new CBrush(RGB(255, 0, 0)), -1, -1);
323     this->rgn = new CRgn();
324 }
325 BOOL CTextDocView::OnEraseBkgnd(CDC* pDC)
326 {
327     pDC->SelectClipRgn(this->rgn);
328     CView::OnEraseBkgnd(pDC);
329     //this->rgn = new CRgn();
330
331     return 0;
332 }
333
334 BOOL CTextDocView::PreCreateWindow(CREATESTRUCT& cs)
335 {
336     return CView::PreCreateWindow(cs);
337 }
338 BOOL CTextDocView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
      message)
339 {
340     return TRUE;
341 }
342
343 void CTextDocView::OnActivateView(BOOL bActivate, CView*
      pActivateView, CView* pDeactiveView)
344 {
345     CView::OnActivateView(bActivate, pActivateView, pDeactiveView);
346     theApp.m_ActiveView = this;
347     this->refresh();
348 }
349
350 BOOL CTextDocView::OnPreparePrinting(CPrintInfo* pInfo)
351 {
352     return this->DoPreparePrinting(pInfo);
353 }
354 void CTextDocView::OnBeginPrinting(CDC* pDc, CPrintInfo* pInfo)
355 {
356     this->printing = TRUE;
357
358     this->recentPrintZoom = theApp.zoom; // Save the current zoom
          so we can resize after the printing
359     theApp.zoom = 5; // TODO : Fix this so that it isn't constant
360
361     this->fFont.CreateFont(static_cast<int>(this->GetDocument()-
          >defBoxHeight * theApp.zoom), 0, 0, 0, FW_NORMAL, FALSE,
          FALSE, 0, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
          CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, 0, theApp.sFont);
362     pDc->SelectObject(&this->fFont);
363 }
364
```

```
365 void CTextDocView::OnEndPrinting(CDC* pDc, CPrintInfo* pInfo)
366 {
367     theApp.zoom = this->recentPrintZoom;
368
369     CRect client;
370     this->GetClientRect(&client);
371     this->OnSize(PRINT_SIZE, client.Width(), client.Height());
372
373     this->GetDocument()->objects[0]->setPosition(CPoint(0, 0));
374
375     for (int i = 1; i < this->GetDocument()->objects.GetSize(); i+
        +) {
376         this->GetDocument()->objects[i]->move(0, this->GetDocument
            ()->objects[i - 1]->getBounds().bottom + this-
            >GetDocument()->objects[i - 1]->getBoxHeight() - this-
            >GetDocument()->objects[i]->getBounds().top);
377     } // Move the resized objects
378
379     this->printIterator = 0;
380     this->returnNewLines = 0;
381     pInfo->m_bContinuePrinting = FALSE;
382
383     this->printing = FALSE;
384
385     this->fFont.DeleteObject();
386     CScrollView::OnEndPrinting(pDc, pInfo);
387 }
388
389 void CTextDocView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
390 {
391     CScrollView::OnPrepareDC(pDC, pInfo);
392 }
393 void CTextDocView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
394 {
395     UNREFERENCED_PARAMETER(pInfo);
396
397     if (pInfo->m_nCurPage == 1) {// If we have just started
        printing, start the process of finding the length of the
        document
398
399         CRect client;
400         this->GetClientRect(&client);
401         this->OnSize(PRINT_SIZE, client.Width(), client.Height
            ()); // Resize the document and stuff
402
403         pInfo->m_rectDraw.top += 600; // Create a space for the
            headers
404         pInfo->m_rectDraw.bottom -= 300; // Create a space for the
            margin
405
406         this->GetDocument()->objects[0]->move(0, pInfo-
            >m_rectDraw.top);
407
408         for (int i = 1; i < this->GetDocument()->objects.GetSize();
             i++) {
409             this->GetDocument()->objects[i]->move(0, this-
```

```
                    >GetDocument()->objects[i-1]->getBounds().bottom +
                    this->GetDocument()->objects[i - 1]->getBoxHeight() -
                     this->GetDocument()->objects[i]->getBounds().top);
410         } // Move the resized objects
411
412         std::tuple<CRect, int> printData;
413         std::get<0>(printData) = CRect();
414         std::get<1>(printData) = 0;
415
416         CSize textExtent = pDC->GetTextExtent(L"A");
417
418         for (int i = 0; i < this->GetDocument()->objects.GetSize();
               i++) {
419             this->GetDocument()->objects[i]->setTextExtent
                  (textExtent);
420             printData = this->GetDocument()->objects[i]-
                  >getPrintBounds(std::get<1>(printData), pInfo-
                  >m_rectDraw.Width());
421         } // Find the size of the document
422
423         pInfo->SetMaxPage(std::ceil(float(std::get<0>
              (printData).bottom) / float(pInfo-
              >m_rectDraw.bottom))); // Finally, set the number of
              pages in the document
424
425     }
426
427     this->pInfo = pInfo; // Save the print info
428     this->OnDraw(pDC);
429
430     if (pInfo->GetMaxPage() == pInfo->m_nCurPage) { // Once we have
          finished printing, resize and reposition all the editors
431         theApp.zoom = this->recentPrintZoom;
432
433         CRect client;
434         this->GetClientRect(&client);
435         this->OnSize(PRINT_SIZE, client.Width(), client.Height());
436
437         this->GetDocument()->objects[0]->setPosition(CPoint(0, 0));
438
439         for (int i = 1; i < this->GetDocument()->objects.GetSize();
               i++) {
440             this->GetDocument()->objects[i]->move(0, this-
                  >GetDocument()->objects[i - 1]->getBounds().bottom +
                  this->GetDocument()->objects[i - 1]->getBoxHeight() -
                   this->GetDocument()->objects[i]->getBounds().top);
441         } // Move the resized objects
442
443         this->printIterator = 0;
444         this->returnNewLines = 0;
445         pInfo->m_bContinuePrinting = FALSE;
446     }
447 }
448
449 // Protected message handlers
450 int CTextDocView::OnCreate(LPCREATESTRUCT lpcs)
```

```
451 {
452     CView::OnCreate(lpcs);
453 
454     ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_IBEAM));
455 
456     this->activeEditor = 0;
457 
458     this->recentEditor = this->activeEditor;
459     this->recentLine = this->GetDocument()->objects[this->activeEditor]->getActiveLine();
460 
461     return 0;
462 }
463 void CTextDocView::OnInitialUpdate()
464 {
465     ResyncScrollSizes();
466     CScrollView::OnInitialUpdate();
467 }
468 void CTextDocView::OnSize(UINT nType, int cx, int cy)
469 {
470     CScrollView::OnSize(nType, cx, cy);
471 
472     if (nType != PRINT_SIZE) {
473         ResyncScrollSizes();        // ensure that scroll info is up-to-date
474     }
475 
476     cx += this->GetScrollPosition().x;
477     cy += this->GetScrollPosition().y;
478 
479     CRect newBounds;
480 
481     for (int i = 0; i < this->GetDocument()->objects.GetSize(); i++) {
482 
483         this->GetDocument()->objects[i]->OnSize(nType, max(cx, this->GetDocument()->objects[i]->getLineTextWidth() + static_cast<int>(this->GetDocument()->lineOffset * theApp.zoom)), cy);
484 
485         newBounds = this->GetDocument()->objects[i]->getBounds();
486         newBounds.right = max(cx, this->GetDocument()->objects[i]->getLineTextWidth() + static_cast<int>(this->GetDocument()->lineOffset * theApp.zoom));
487 
488         this->GetDocument()->objects[i]->setBounds(newBounds);
489     }
490 
491     if (nType != PRINT_SIZE) {
492         this->updateWindow();
493     }
494 
495 }
496 
497 void CTextDocView::OnFilePrintPreview()
498 {
```

```
499 #ifndef SHARED_HANDLERS
500     AFXPrintPreview(this);
501 #endif
502 }
503
504 void CTextDocView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
      pScrollBar)
505 {
506     CScrollView::OnHScroll(nSBCode, nPos, pScrollBar);
507
508     this->expandLineBounds();
509
510     //this->selectClippingRgn(H_SCROLL);
511     this->updateWindow(FALSE, TRUE);
512 }
513 void CTextDocView::OnLButtonUp(UINT nFlags, CPoint point)
514 {
515     point += this->GetScrollPosition();
516
517     BOOL redraw = this->GetDocument()->objects[this->activeEditor]-
        >OnLButtonUp(nFlags, point);
518
519     std::vector <int> lineNums = this->GetDocument()->objects[this-
        >activeEditor]->iGetLineNum();
520
521     if (lineNums[0] != 0) {
522
523         lineNums.push_back(1);
524
525         BOOL iUpdate = FALSE;
526         int i = this->activeEditor+1;
527
528         BOOL exists = FALSE;
529         int pos = 1;
530
531         int lineNumSize = lineNums.size();
532
533         // TODO : Possibly change to a binary search to make
              searching more efficient
534         while (!iUpdate && i < this->GetDocument()->objects.GetSize
              ()) { // Iterate through all editors under one that was
              clicked
535
536             if (lineNums == this->GetDocument()->objects[i]-
                  >iGetLineNum(1)) { // If the editor requested to be
                  made has already been made, go to that editor and
                  don't create a new one
537
538                 this->GetDocument()->objects[this->activeEditor]-
                     >setActive(FALSE);
539                 this->activeEditor = i;
540                 this->GetDocument()->objects[this->activeEditor]-
                     >setActive(TRUE);
541
542                 iUpdate = TRUE;
543                 exists = TRUE;
```

```
544             }
545
546             else { // If this editor does not match, we must check to see if the requested editor should go before this one, or if we should keep searching
547
548                 BOOL jUpdate = FALSE;
549                 int j = 0;
550
551                 while (!jUpdate && j < min(lineNumSize, this->GetDocument()->objects[i]->iGetLineNum(1).size()())) { // Iterate through the line numbers' sublines
552
553                     if (lineNums[j] < this->GetDocument()->objects[i]->iGetLineNum(1)[j]) { // If this passes, the editor should be above it
554
555                         jUpdate = TRUE;
556                         iUpdate = TRUE;
557
558                         pos = i;
559                     }
560                     else if(lineNums[j] > this->GetDocument()->objects[i]->iGetLineNum(1)[j]) {
561                         jUpdate = TRUE;
562                     }
563                     j++;
564                 }
565
566
567             }
568
569             i++;
570         }
571
572         if (!iUpdate) {
573             pos = this->GetDocument()->objects.GetSize();
574         }
575
576         if (!exists) {
577
578             this->GetDocument()->objects[this->activeEditor]->setActive(FALSE); // Set old editor not active
579
580             int oldEditor = this->activeEditor;
581             this->activeEditor = pos; // Change position for editing
582
583             CRect aboveRect = this->GetDocument()->objects[this->activeEditor - 1]->getBounds();
584
585             CRect editorBounds;
586             editorBounds.top = aboveRect.bottom + this->GetDocument()->objects[this->activeEditor - 1]->getBoxHeight();
587             editorBounds.bottom = editorBounds.top + this-
```

```
                >GetDocument()->objects[this->activeEditor - 1]-
                >getBoxHeight();
588             editorBounds.left = aboveRect.left;
589             editorBounds.right = aboveRect.right;
590
591             std::vector<CString> text = { L"" };
592
593             text[0] = L"// TODO : Add a subline";
594
595             this->GetDocument()->objects.InsertAt(
596                 this->activeEditor,
597                 new CTextEditorObject(editorBounds, L"0", TRUE, 0,
                  this->GetDocument()->objects[this->activeEditor -
                  1]->getBoxHeight(TRUE), TRUE, text, lineNums,
                  this->GetDocument()->lineOffset)); // Add a new
                  line
598
599             CString id;
600
601             id.Format(L"%d", this->activeEditor + 1);
602             this->GetDocument()->objects[this->activeEditor]->setID
                (id);
603
604             for (int i = this->activeEditor + 1; i < this-
                >GetDocument()->objects.GetSize(); i++) {
605
606                 id.Format(L"%d", i + 1);
607
608                 this->GetDocument()->objects[i]->move(0, this-
                  >GetDocument()->objects[this->activeEditor]-
                  >getBounds().Height() + this->GetDocument()-
                  >objects[this->activeEditor - 1]->getBoxHeight());
609                 this->GetDocument()->objects[i]->setID(id);
610             } // Move all lines and reset their IDs
611
612             this->selectClippingRgn(EDITOR_EDIT); // Will draw
                everything under and including the added editor
613
614             // We should have something like this, but it works
                without it for some reason so it's getting left out
                for now
615             /*CRgn* tempRgn = this->rgn;
616             this->rgn = new CRgn();
617
618             this->setClippingRgn(SELECT_LINE);
619             this->rgn->CombineRgn(this->rgn, tempRgn, RGN_OR);*/
620
621             this->ResyncScrollSizes();
622         }
623         redraw = TRUE;
624     }
625     this->updateWindow(FALSE, redraw);
626 }
627 void CTextDocView::OnLButtonDown(UINT nFlags, CPoint point)
628 {
629     point += this->GetScrollPosition();
```

```
630
631     if (this->recentEditor != this->activeEditor) {
632         this->recentEditor = this->activeEditor;
633     }
634     if (this->recentLine != this->GetDocument()->objects[this->recentEditor]->getActiveLine()) {
635         this->recentLine = this->GetDocument()->objects[this->recentEditor]->getActiveLine();
636     }
637
638     BOOL update = FALSE;
639     int i = 0;
640     while (!update && i < this->GetDocument()->objects.GetSize()) {
641
642         if (this->GetDocument()->objects[i]->checkPointInBounds(point) || point.y < this->GetDocument()->objects[i]->getBounds().bottom + this->GetDocument()->objects[i]->getBoxHeight()) {
643             if (i != this->activeEditor) {
644
645                 this->GetDocument()->objects[this->activeEditor]->setActive(FALSE);
646                 this->activeEditor = i;
647                 this->GetDocument()->objects[this->activeEditor]->setActive(TRUE);
648             }
649             update = TRUE;
650         }
651         i++;
652     }
653
654     if (!update) {
655
656         this->GetDocument()->objects[this->activeEditor]->setActive(FALSE);
657         this->activeEditor = this->GetDocument()->objects.GetSize() - 1;
658         this->GetDocument()->objects[this->activeEditor]->setActive(TRUE);
659     }
660
661     BOOL oldHlght = this->recentHlght;
662
663     if (this->recentHlght != this->GetDocument()->objects[this->recentEditor]->hasHighlight()) {
664         this->recentHlght = this->GetDocument()->objects[this->recentEditor]->hasHighlight();
665     }
666
667     BOOL sidebarbtn = this->GetDocument()->objects[this->activeEditor]->OnLButtonDown(nFlags, point);
668
669     this->selectClippingRgn(SELECT_LINE); // THIS NEEDS TO GO BEFORE THE ADD_RGN_SIDEBAR_BTN
670
671     this->recentBlockLine = this->GetDocument()->objects[this->
```

```
          >recentEditor]->getBlockLine();
672       int oldEditor = this->activeEditor;
673       this->recentEditor = this->activeEditor;
674
675       if (sidebarbtn) {
676
677           if (this->recentLine == this->recentBlockLine && !oldHlght
              && !this->GetDocument()->objects[oldEditor]->hasHighlight
              ()) {
678               this->selectClippingRgn(ADD_RGN_SIDEBAR_BTN, 0);
679           }
680           else {
681               this->selectClippingRgn(ADD_RGN_SIDEBAR_BTN, 1);
682           }
683       }
684
685       this->updateWindow(FALSE, TRUE);
686   }
687   void CTextDocView::OnLButtonDblClk(UINT nFlags, CPoint point)
688   {
689       point += this->GetScrollPosition();
690
691       this->GetDocument()->objects[this->activeEditor]-
            >OnLButtonDblClk(nFlags, point);
692
693       for (int i = 0; i < this->GetDocument()->objects.GetSize(); i+
            +) {
694           if (i != this->activeEditor) {
695
696               this->GetDocument()->objects[i]->hlghtingOff();
697           }
698       }
699
700       this->updateWindow(FALSE, TRUE);
701   }
702   void CTextDocView::OnMouseMove(UINT nFlags, CPoint point)
703   {
704       point += this->GetScrollPosition();
705
706       int old_cursor = this->GetDocument()->objects[this-
            >activeEditor]->getCursorArrow();
707
708       BOOL objectRedraw = this->GetDocument()->objects[this-
            >activeEditor]->OnMouseMove(nFlags, point);
709
710       if (nFlags == VK_LEFT || objectRedraw) {
711
712           for (int i = 0; i < this->GetDocument()->objects.GetSize();
               i++) {
713               if (i != this->activeEditor) {
714
715                   this->GetDocument()->objects[i]->hlghtingOff();
716               }
717           }
718
719           this->selectClippingRgn(HIGHLIGHTING);
```

```
720
721         if (this->recentBlockLine != this->GetDocument()->objects
              [this->recentEditor]->getBlockLine() && this-
              >recentBlockLine != -1) {
722             this->selectClippingRgn(ADD_RGN_SIDEBAR_BTN, 1);
723             this->recentEditor = this->activeEditor;
724             this->recentBlockLine = this->GetDocument()->objects
                  [this->recentEditor]->getBlockLine();
725         }
726         if (objectRedraw && nFlags != VK_LEFT && this->GetDocument
              ()->objects[this->activeEditor]->getBlockLine() != -1) {
727             this->recentBlockLine = this->GetDocument()->objects
                  [this->activeEditor]->getBlockLine();
728             this->selectClippingRgn(ADD_RGN_SIDEBAR_BTN, 1);
729         }
730         this->updateWindow(FALSE, TRUE);
731     }
732
733     int new_cursor = this->GetDocument()->objects[this-
        >activeEditor]->getCursorArrow();
734
735     if (new_cursor != old_cursor) {
736
737         if (new_cursor == 0) {
738
739             ::SetCursor(AfxGetApp()->LoadStandardCursor
                  (IDC_IBEAM));
740         }
741         else if (new_cursor == 1) {
742
743             ::SetCursor(AfxGetApp()->LoadStandardCursor
                  (IDC_ARROW));
744         }
745         else {
746
747             ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_HAND));
748         }
749     }
750 }
751
752 BOOL CTextDocView::OnMouseWheel(UINT nFlags, short zDelta, CPoint
      pt)
753 {
754     if (nFlags == MK_CONTROL) {
755
756         BOOL adjust = TRUE;
757
758         if ((theApp.zoom <= 0.75 && zDelta < 0) || (theApp.zoom >=
              2.8 && zDelta > 0)) {
759             adjust = FALSE;
760         }
761
762         if (adjust) {
763
764             theApp.zoom += zDelta / 1000.f;
765             this->ResyncScrollSizes(TRUE, TRUE);
```

```
766             }
767
768             this->refresh(TRUE, FALSE);
769
770             return TRUE;
771         }
772         else {
773             return CScrollView::OnMouseWheel(nFlags, zDelta, pt);
774         }
775     }
776
777     void CTextDocView::OnRButtonUp(UINT nFlags, CPoint point)
778     {
779         point += this->GetScrollPosition();
780
781         this->GetDocument()->objects[this->activeEditor]->OnRButtonUp
              (nFlags, point);
782
783
784         if (point.x - this->GetScrollPosition().x > this->GetDocument
              ()->lineOffset) {
785             this->ClientToScreen(&point);
786             OnContextMenu(this, point);
787         }
788     }
789     void CTextDocView::OnRButtonDown(UINT nFlags, CPoint point)
790     {
791         point += this->GetScrollPosition();
792
793         if (this->recentEditor != this->activeEditor) {
794             this->recentEditor = this->activeEditor;
795         }
796         if (this->recentLine != this->GetDocument()->objects[this-
              >recentEditor]->getActiveLine()) {
797             this->recentLine = this->GetDocument()->objects[this-
                  >recentEditor]->getActiveLine();
798         }
799
800         BOOL update = FALSE;
801         int i = 0;
802         while (!update && i < this->GetDocument()->objects.GetSize()) {
803
804             if (this->GetDocument()->objects[i]->checkPointInBounds
                  (point) || point.y < this->GetDocument()->objects[i]-
                  >getBounds().bottom + this->GetDocument()->objects[i]-
                  >getBoxHeight()) {
805                 if (i != this->activeEditor) {
806
807                     this->GetDocument()->objects[this->activeEditor]-
                          >setActive(FALSE);
808                     this->activeEditor = i;
809                     this->GetDocument()->objects[this->activeEditor]-
                          >setActive(TRUE);
810                 }
811                 update = TRUE;
812             }
```

```
813             i++;
814         }
815
816         if (!update) {
817
818             this->GetDocument()->objects[this->activeEditor]->setActive
                  (FALSE);
819             this->activeEditor = this->GetDocument()->objects.GetSize()
                   - 1;
820             this->GetDocument()->objects[this->activeEditor]->setActive
                  (TRUE);
821         }
822
823         if (this->recentHlght != this->GetDocument()->objects[this-
              >recentEditor]->hasHighlight()) {
824             this->recentHlght = this->GetDocument()->objects[this-
                  >recentEditor]->hasHighlight();
825         }
826
827         this->GetDocument()->objects[this->activeEditor]->OnRButtonDown
              (nFlags, point);
828
829         this->selectClippingRgn(SELECT_LINE); // THIS NEEDS TO GO
              BEFORE THE ADD_RGN_SIDEBAR_BTN
830
831         this->recentEditor = this->activeEditor;
832
833         this->updateWindow(FALSE, TRUE);
834     }
835     void CTextDocView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
836     {
837         BOOL redraw = TRUE;
838
839         switch (nChar) {
840
841         case VK_RETURN: {
842
843             // TODO : Make this more efficient (but make it work first)
844
845             BOOL move = TRUE, increment = TRUE;
846
847             std::vector<int> line = this->GetDocument()->objects[this-
                  >activeEditor]->iGetLineNum(this->GetDocument()->objects
                  [this->activeEditor]->getActiveLine() + 1);
848             line.push_back(1);
849
850             for (int i = this->activeEditor + 1; i < this->GetDocument
                  ()->objects.GetSize(); i++) {
851
852                 if (this->GetDocument()->objects[i]->iGetLineNum(1) ==
                      line) {
853
854                     if (this->GetDocument()->objects[this-
                          >activeEditor]->getCaretPos() == this->GetDocument
                          ()->objects[this->activeEditor]->getLineText
                          ().GetLength()) {
```

```
855                         increment = FALSE;
856
857                     }
858                     else if (this->GetDocument()->objects[this-
                         >activeEditor]->getCaretPos() != 0) {
859
860                         move = FALSE;
861
862                         // SEND A WARNING MESSAGE
863                         cout << "UNABLE TO PROCESS COMMAND" << endl;
864                     }
865                 }
866             }
867
868             if (move) {
869
870                 if (this->GetDocument()->objects[this->activeEditor]-
                      >hasHighlight()) {
871
872                     this->OnKeyDown(VK_BACK, 1, 0);
873                 }
874
875                 this->GetDocument()->objects[this->activeEditor]-
                      >OnRecieveReturn(); // Send the return message to the
                       active editor so it can handle itself
876
877                 // Move editors and increment sublines
878                 for (int i = this->activeEditor + 1; i < this-
                      >GetDocument()->objects.GetSize(); i++) { // Iterate
                      through all the editors underneath the active editor
879
880                     this->GetDocument()->objects[i]->move(0, this-
                         >GetDocument()->objects[this->activeEditor]-
                         >getBoxHeight()); // Move the editors down the
                         appropriate length
881
882                     if (this->GetDocument()->objects[this-
                         >activeEditor]->iGetLineNum(1).size() < this-
                         >GetDocument()->objects[i]->iGetLineNum(1).size())
                          { // Check is the line to be changed is bigger,
                         as any line that is the same size of smaller than
                         the avtive editor will never need changed
883
884                         if (this->GetDocument()->objects[i]-
                         >iGetLineNum(1)[this->GetDocument()->objects[this-
                         >activeEditor]->iGetLineNum(1).size() - 1] > /*Get
                          line 1's number of the iterated editor and take
                         the index that is the same as the last index in
                         the active editor line number's*/
885                             this->GetDocument()->objects[this-
                         >activeEditor]->iGetLineNum(this->GetDocument()-
                         >objects[this->activeEditor]->getActiveLine
                         ()).back()) /*Get the active line's number and
                         take the last index*/ {
886                             // Compare the numbers. See if the editor
                         needs to be incremented by having larger
```

```
                    appropriate index
887
888                         this->GetDocument()->objects[i]->incrementSublines(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1, 1);
889                     }
890                     else if (this->GetDocument()->objects[i]->iGetLineNum(1)[this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1] ==
891                         this->GetDocument()->objects[this->activeEditor]->iGetLineNum(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).back()) {
892                         // Compare the same numbers as in the previous statement but check for equality
893
894                         if (increment) { // Check that an increment is appropriate
895
896                             this->GetDocument()->objects[i]->incrementSublines(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1, 1);
897                         }
898                     }
899                 }
900             }
901
902             this->ResyncScrollSizes();
903             this->selectClippingRgn(RETURN_BACK);
904         }
905
906         break;
907         }
908
909     case VK_BACK: {
910
911
912         BOOL move = TRUE;
913         BOOL removeEditor = FALSE;
914
915         if (this->GetDocument()->objects[this->activeEditor]->getCaretPos() == 0 || this->GetDocument()->objects[this->activeEditor]->isHlghtMultiline()) { // If the the caret is at the start of the line
916
917             std::vector<std::vector<int>> lines = { };
918
919             // Get the line number of the active line
920             if (this->GetDocument()->objects[this->activeEditor]->getCaretPos() == 0) {
921                 lines.push_back(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(this->GetDocument()->objects[this->activeEditor]->getActiveLine() + 1));
```

```
922             }
923
924             if (this->GetDocument()->objects[this->activeEditor]->isHlghtMultiline()) {
925                 BOOL first = FALSE;
926                 BOOL last = FALSE;
927
928                 int i = 0;
929
930                 while (!last && i < this->GetDocument()->objects[this->activeEditor]->getNumLines()) {
931
932                     switch (this->GetDocument()->objects[this->activeEditor]->lineHighlight(i)) {
933
934                     case 0:
935                         if (first) {
936                             last = TRUE;
937                         }
938                         break;
939
940                     case 1:
941                         if (first) {
942                             lines.push_back(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(i + 1));
943                             last = TRUE;
944                         }
945                         else {
946                             first = TRUE;
947                         }
948                         break;
949
950                     case 2:
951                         if (!first) {
952                             first = TRUE;
953                         }
954
955                         lines.push_back(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(i + 1));
956                         break;
957                     }
958
959                     i++;
960                 }
961             }
962
963             for (auto& it : lines) {
964                 it.push_back(1);
965             }
966
967             // Check if the line wanting to delete has a subline
968             for (int i = this->activeEditor + 1; i < this->GetDocument()->objects.GetSize(); i++) {
969
970                 for (auto& line : lines) {
971
```

```
972                     if (this->GetDocument()->objects[i]->iGetLineNum(1) == line) {
973
974                         move = FALSE;
975
976                         // SEND A WARNING MESSAGE
977                         cout << "UNABLE TO PROCESS COMMAND" << endl;
978                     }
979                 }
980             }
981
982             // If the line we want to delete does not have a subline...
983             if (move) {
984
985                 // and we are not on the first line
986                 if (this->GetDocument()->objects[this->activeEditor]->getActiveLine() != 0) {
987
988                     // Move all the editors up one line space and increment them appropriatley
989                     for (int i = this->activeEditor + 1; i < this->GetDocument()->objects.GetSize(); i++) {
990                         this->GetDocument()->objects[i]->move(0, -this->GetDocument()->objects[this->activeEditor]->getBoxHeight());
991
992                         if (this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
993
994                             if (this->GetDocument()->objects[i]->iGetLineNum(1)[this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1] >
995                                 this->GetDocument()->objects[this->activeEditor]->iGetLineNum(this->GetDocument()->objects[this->activeEditor]->getStartLine()).back()) {
996
997                                 this->GetDocument()->objects[i]->incrementSublines(this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1, -1);
998                             }
999                         }
1000
1001                         else if (this->GetDocument()->objects[i]->iGetLineNum(1)[this->GetDocument()->objects[this->activeEditor]->iGetLineNum(1).size() - 1] >
1002                                 this->GetDocument()->objects[this->activeEditor]->iGetLineNum(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).back()) {
1003
1004                             this->GetDocument()->objects[i]->incrementSublines(this->GetDocument()->objects
```

```
                    [this->activeEditor]->iGetLineNum(1).size() - 1, -1);
1005                        }
1006                    }
1007                }
1008
1009                // and there is only one line, but it is not the first editor
1010                else if (this->GetDocument()->objects[this->activeEditor]->getNumLines() == 1 && this->activeEditor != 0 && !this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
1011
1012                    // this means that the user want to delete this editor
1013
1014                    this->selectClippingRgn(EDITOR_EDIT); // Will draw everything under and including the removed editor
1015
1016                    // Remove the editor and move all the editors back
1017
1018                    for (int i = this->activeEditor + 1; i < this->GetDocument()->objects.GetSize(); i++) {
1019                        this->GetDocument()->objects[i]->move(0, -(this->GetDocument()->objects[this->activeEditor]->getBounds().Height() + this->GetDocument()->objects[this->activeEditor]->getBoxHeight()));
1020                    }
1021                    this->GetDocument()->objects.RemoveAt(this->activeEditor);
1022
1023                    std::vector<int> oldLine = lines[0];
1024                    lines.pop_back();
1025                    lines.pop_back();
1026
1027                    // TODO : Fix this so the parent line of the deleted editor becomes the active editor
1028                    if (lines.size() > 0) {
1029
1030                        BOOL found = FALSE;
1031                        int i = 0;
1032
1033                        while (!found) {
1034
1035                            found = TRUE;
1036                        }
1037
1038                        this->activeEditor--;
1039                    }
1040
1041                    else {
1042
1043                        this->activeEditor--;
1044                    }
```

```
1045
1046                    this->GetDocument()->objects[this->activeEditor]->setActive(TRUE);
1047
1048                    removeEditor = TRUE;
1049
1050                    // Adds the line that needs to be selected to the region
1051                    CRgn rgn2;
1052                    rgn2.CreateRectRgn( this->GetDocument()->objects[this->activeEditor]->getBounds().left,
1053                                        this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).top,
1054                                        this->GetDocument()->objects[this->activeEditor]->getBounds().right,
1055                                        this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).bottom);
1056
1057                    this->rgn->CombineRgn(this->rgn, &rgn2, RGN_OR);
1058                }
1059                else if(!this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
1060                    move = FALSE;
1061                }
1062            }
1063        }
1064        if (move && !removeEditor) {
1065
1066            if(this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
1067
1068                if (this->GetDocument()->objects[this->activeEditor]->isHlghtMultiline()) {
1069                    this->selectClippingRgn(REMOVE_HIGHLIGHT);
1070                    this->GetDocument()->objects[this->activeEditor]->OnRecieveBackspace();
1071                    this->selectClippingRgn(ADD_RGN_ACTIVE_LINE);
1072                }
1073                else {
1074                    this->GetDocument()->objects[this->activeEditor]->OnRecieveBackspace();
1075                    this->selectClippingRgn(TEXT);
1076                    this->selectClippingRgn(ADD_RGN_ACTIVE_LINE);
1077                }
1078
1079                if (this->GetDocument()->objects.GetSize() > this->activeEditor + 1) {
1080
1081                    int dy = this->GetDocument()->objects[this->activeEditor + 1]->getBounds().top - (this->GetDocument()->objects[this->activeEditor]->
```

```
                    >getBounds().bottom + this->GetDocument()->objects
                    [this->activeEditor]->getBoxHeight());
1082
1083                    for (int i = this->activeEditor + 1; i < this-
                    >GetDocument()->objects.GetSize(); i++) {
1084
1085                        this->GetDocument()->objects[i]->move(0, -
                    dy);
1086                    }
1087                }
1088            }
1089            else {
1090
1091                if (this->GetDocument()->objects[this-
                    >activeEditor]->getCaretPos() == 0) {
1092                    // Send backspace to editor AFTER setting the
                    clipping region here
1093                    this->selectClippingRgn(RETURN_BACK);
1094                    this->GetDocument()->objects[this-
                    >activeEditor]->OnRecieveBackspace();
1095                }
1096                else {
1097                    // Send backspace to editor BEFORE setting the
                    clipping region here
1098                    this->GetDocument()->objects[this-
                    >activeEditor]->OnRecieveBackspace();
1099                    this->selectClippingRgn(TEXT);
1100                }
1101            }
1102            this->ResyncScrollSizes();
1103        }
1104
1105        break;
1106    }
1107
1108    case VK_TAB:
1109        this->GetDocument()->objects[this->activeEditor]-
              >OnRecieveTab();
1110        break;
1111
1112    default:
1113        if (this->textInput.OnKeyDown(nChar, nRepCnt, nFlags)) {
1114            if (this->GetDocument()->objects[this->activeEditor]-
                  >hasHighlight()) {
1115
1116                this->OnKeyDown(VK_BACK, 1, 0);
1117            }
1118            else {
1119
1120                // Set the clipping region for basic text input,
                    with no highlighting
1121                this->selectClippingRgn(TEXT);
1122            }
1123
1124            this->GetDocument()->objects[this->activeEditor]-
                  >OnRecieveText(this->textInput.RecieveText());
```

```
1125
1126
1127
1128                this->ResyncScrollSizes();
1129            }
1130            else {
1131                int oldLine = this->GetDocument()->objects[this->activeEditor]->getActiveLine();
1132                int oldCaretPos = this->GetDocument()->objects[this->activeEditor]->getCaretPos();
1133
1134                this->GetDocument()->objects[this->activeEditor]->OnKeyDown(nChar, nRepCnt, nFlags);
1135
1136                if (oldLine != this->GetDocument()->objects[this->activeEditor]->getActiveLine()) {
1137                    this->recentLine = oldLine;
1138                    this->selectClippingRgn(SELECT_LINE);
1139                }
1140                else if (oldCaretPos != this->GetDocument()->objects[this->activeEditor]->getCaretPos()) {
1141                    this->selectClippingRgn(CARET_MOVE);
1142                    redraw = FALSE;
1143                }
1144
1145                this->ResyncScrollSizes(FALSE, TRUE);
1146            }
1147
1148            break;
1149        }
1150
1151        if (redraw) {
1152            this->updateWindow(FALSE, TRUE);
1153        }
1154    }
1155    void CTextDocView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
1156    {
1157        this->textInput.OnKeyUp(nChar, nRepCnt, nFlags);
1158    }
1159
1160    void CTextDocView::OnContextMenu(CWnd* pWnd, CPoint point)
1161    {
1162        point -= this->GetScrollPosition();
1163
1164        this->GetDocument()->objects[this->activeEditor]->OnContextMenu(pWnd, point);
1165    }
1166
1167    void CTextDocView::OnCopy()
1168    {
1169        if (this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
1170            if (!OpenClipboard())
1171            {
1172                AfxMessageBox(_T("Cannot open the Clipboard"));
1173                return;
```

```
1174            }
1175
1176            // Remove the current Clipboard contents
1177            if (!EmptyClipboard())
1178            {
1179                AfxMessageBox(_T("Cannot empty the Clipboard"));
1180                return;
1181            }
1182
1183            // Get the currently selected data
1184            CStringA text(this->GetDocument()->objects[this->activeEditor]->getHighlightedText());
1185
1186            const char* x = text;
1187
1188            HGLOBAL hGlob = GlobalAlloc(GMEM_FIXED, text.GetLength() * 2);
1189
1190            strcpy_s((char*)hGlob, text.GetLength() * 2, x);
1191
1192            // For the appropriate data formats...
1193            if (::SetClipboardData(CF_TEXT, hGlob) == NULL)
1194            {
1195                CString msg;
1196                msg.Format(_T("Unable to set Clipboard data, error: %d"), GetLastError());
1197                AfxMessageBox(msg);
1198                CloseClipboard();
1199                GlobalFree(hGlob);
1200                return;
1201            }
1202            CloseClipboard();
1203        }
1204    }
1205    void CTextDocView::OnPaste()
1206    {
1207        HANDLE hGlob;
1208
1209        // If the application is in edit mode,
1210        // get the clipboard text.
1211
1212        if (!IsClipboardFormatAvailable(CF_TEXT)) {
1213            return;
1214        }
1215
1216        if (!OpenClipboard()) {
1217            AfxMessageBox(_T("Cannot open the Clipboard"));
1218            return;
1219        }
1220
1221        hGlob = GetClipboardData(CF_TEXT);
1222        if (hGlob != NULL)
1223        {
1224            if (this->GetDocument()->objects[this->activeEditor]->hasHighlight()) {
1225                this->GetDocument()->objects[this->activeEditor]->
```

```
                    >OnRecieveBackspace();
1226            }
1227
1228            this->recentLine = this->GetDocument()->objects[this-
                  >activeEditor]->getActiveLine();
1229
1230            int carridgeReturn = this->GetDocument()->objects[this-
                  >activeEditor]->OnRecieveText(CString((char*)GlobalLock
                  (hGlob)), TRUE);
1231
1232            for (int i = this->activeEditor + 1; i < this->GetDocument
                  ()->objects.GetSize(); i++) {
1233
1234                this->GetDocument()->objects[i]->move(0, this-
                      >GetDocument()->objects[this->activeEditor]-
                      >getBoxHeight() * carridgeReturn);
1235            }
1236            GlobalUnlock(hGlob);
1237        }
1238        else {
1239            AfxMessageBox(_T("Clipboard is Empty"));
1240            return;
1241        }
1242        CloseClipboard();
1243
1244        this->selectClippingRgn(PASTE);
1245
1246        this->ResyncScrollSizes(TRUE, TRUE);
1247        this->updateWindow(FALSE, TRUE);
1248
1249        return;
1250    }
1251    void CTextDocView::OnCut()
1252    {
1253        this->OnCopy();
1254        if (this->GetDocument()->objects[this->activeEditor]-
              >hasHighlight()) {
1255            this->OnKeyDown(VK_BACK, 1, 0);
1256        }
1257    }
1258
1259    void CTextDocView::expandLineBounds()
1260    {
1261        CRect client;
1262        this->GetClientRect(&client);
1263
1264        CRect newBounds;
1265
1266        for (int i = 0; i < this->GetDocument()->objects.GetSize(); i+
              +) {
1267
1268            newBounds = this->GetDocument()->objects[i]->getBounds();
1269            newBounds.right = max(client.right + this-
                  >GetScrollPosition().x, this->GetDocument()->objects[i]-
                  >getLineTextWidth() + static_cast<int>(this->GetDocument
                  ()->lineOffset * theApp.zoom));
```

```
1270
1271            this->GetDocument()->objects[i]->setBounds(newBounds);
1272        }
1273    }
1274
1275    void CTextDocView::selectClippingRgn(int nAction, int type)
1276    {
1277        int x1 = 0, y1 = 0, x2 = 0, y2 = 0;
1278
1279        switch (nAction) {
1280
1281        case TEXT: { // Working
1282
1283            if (this->recentHlght) {
1284                x2 = this->GetDocument()->objects[this->activeEditor]-
                      >getBounds().left + this->GetDocument()->lineOffset +
                       ((this->GetDocument()->objects[this->activeEditor]-
                      >getLineText(this->GetDocument()->objects[this-
                      >activeEditor]->getActiveLine() + 1).GetLength() + 1)
                       * this->GetDocument()->objects[this->activeEditor]-
                      >getTextExtent().cx) - this->GetScrollPosition().x;
1285            }
1286            else {
1287                x2 = this->GetDocument()->objects[this->activeEditor]-
                      >getLineBounds(this->GetDocument()->objects[this-
                      >activeEditor]->getActiveLine()).right - this-
                      >GetScrollPosition().x;
1288            }
1289
1290            int drawpos = this->GetDocument()->objects[this-
                  >activeEditor]->getCaretPos();
1291
1292            int i = this->GetDocument()->objects[this->activeEditor]-
                  >getCaretPos();
1293            BOOL end = FALSE;
1294
1295            while (i >= 0 && !end) {
1296
1297                if (this->GetDocument()->objects[this->activeEditor]-
                      >getLineText().Mid(i, 1) == L" " || i == 0) {
1298                    drawpos = i;
1299                    end = TRUE;
1300                }
1301
1302                i--;
1303            }
1304
1305            x1 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().left + this->GetDocument()->lineOffset +
                  (drawpos * this->GetDocument()->objects[this-
                  >activeEditor]->getTextExtent().cx) - this-
                  >GetScrollPosition().x;
1306            y1 = this->GetDocument()->objects[this->activeEditor]-
                  >getLineBounds(this->GetDocument()->objects[this-
                  >activeEditor]->getActiveLine()).top - this-
                  >GetScrollPosition().y;
```

```
1307            y2 = this->GetDocument()->objects[this->activeEditor]-
                  >getLineBounds(this->GetDocument()->objects[this-
                  >activeEditor]->getActiveLine()).bottom - this-
                  >GetScrollPosition().y;
1308
1309            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1310            break;
1311        }
1312
1313        case RETURN_BACK: // Working
1314
1315            x1 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().left - this->GetScrollPosition().x;
1316            y1 = this->GetDocument()->objects[this->activeEditor]-
                  >getLineBounds(this->GetDocument()->objects[this-
                  >activeEditor]->getActiveLine()-1).top - this-
                  >GetScrollPosition().y;
1317            x2 = this->GetDocument()->objects[this->GetDocument()-
                  >objects.GetSize() - 1]->getBounds().right - this-
                  >GetScrollPosition().x;
1318            y2 = this->GetDocument()->objects[this->GetDocument()-
                  >objects.GetSize() - 1]->getBounds().bottom + this-
                  >GetDocument()->objects[this->activeEditor]->getBoxHeight
                  () - this->GetScrollPosition().y;
1319            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1320            break;
1321
1322        case EDITOR_EDIT: { // Working
1323            x1 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().left - this->GetScrollPosition().x;
1324            y1 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().top - this->GetScrollPosition().y;
1325            x2 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().right - this->GetScrollPosition().x;
1326            y2 = this->GetDocument()->objects[this->GetDocument()-
                  >objects.GetSize() - 1]->getBounds().bottom - this-
                  >GetScrollPosition().y;
1327            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1328
1329            CRgn* rgn1 = new CRgn();
1330            CRect tempRect = this->GetDocument()->objects[this-
                  >recentEditor]->getLineBounds(this->GetDocument()-
                  >objects[this->recentEditor]->getActiveLine());
1331
1332            x1 = tempRect.left;
1333            y1 = tempRect.top;
1334            x2 = tempRect.right;
1335            y2 = tempRect.bottom;
1336
1337            VERIFY(rgn1->CreateRectRgn(x1, y1, x2, y2));
1338            this->rgn->CombineRgn(this->rgn, rgn1, RGN_OR);
1339
1340            this->selectClippingRgn(ADD_RGN_SIDEBAR_BTN, 1);
1341
1342            break;
1343        }
```

```
1344
1345        case HIGHLIGHTING: { // Working
1346            CRect temp = this->GetDocument()->objects[this->activeEditor]->getHighlightClippingRect();
1347            x1 = temp.left - this->GetScrollPosition().x;
1348            y1 = temp.top - this->GetScrollPosition().y;
1349            x2 = temp.right - this->GetScrollPosition().x;
1350            y2 = temp.bottom - this->GetScrollPosition().y;
1351            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1352            break;
1353        }
1354
1355        case SELECT_LINE: { // Working
1356
1357            if (this->GetDocument()->objects[this->activeEditor]->getActiveLine() != this->recentLine) {
1358                CRgn* rgn1 = new CRgn();
1359                CRgn* rgn2 = new CRgn();
1360                CRgn* tempRgn = new CRgn();
1361
1362                CRect temp = this->GetDocument()->objects[this->recentEditor]->getLineBounds(this->recentLine);
1363
1364                x1 = temp.left - this->GetScrollPosition().x;
1365                y1 = temp.top - this->GetScrollPosition().y;
1366                x2 = temp.right - this->GetScrollPosition().x;
1367                y2 = temp.bottom - this->GetScrollPosition().y;
1368
1369                VERIFY(rgn1->CreateRectRgn(x1, y1, x2, y2));
1370
1371                x1 += 1;
1372                y1 += 1;
1373                x2 -= 1;
1374                y2 -= 1;
1375
1376                VERIFY(tempRgn->CreateRectRgn(x1, y1, x2, y2));
1377
1378                int nCombineResult = rgn1->CombineRgn(rgn1, tempRgn, RGN_XOR);
1379                ASSERT(nCombineResult != ERROR && nCombineResult != NULLREGION);
1380
1381                tempRgn = new CRgn();
1382
1383                x1 = this->GetCaretPos().x;
1384                x2 = x1 + this->GetDocument()->objects[0]->getTextExtent().cx*0.25;
1385
1386                VERIFY(tempRgn->CreateRectRgn(x1, y1, x2, y2));
1387
1388                nCombineResult = rgn1->CombineRgn(rgn1, tempRgn, RGN_XOR);
1389                ASSERT(nCombineResult != ERROR && nCombineResult != NULLREGION);
1390
1391                tempRgn = new CRgn();
```

```
1392
1393            temp = this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine());
1394
1395            x1 = temp.left - this->GetScrollPosition().x;
1396            y1 = temp.top - this->GetScrollPosition().y;
1397            x2 = temp.right - this->GetScrollPosition().x;
1398            y2 = temp.bottom - this->GetScrollPosition().y;
1399
1400            VERIFY(rgn2->CreateRectRgn(x1, y1, x2, y2));
1401
1402            x1 += 1;
1403            y1 += 1;
1404            x2 -= 1;
1405            y2 -= 1;
1406
1407            VERIFY(tempRgn->CreateRectRgn(x1, y1, x2, y2));
1408
1409            nCombineResult = rgn2->CombineRgn(rgn2, tempRgn, RGN_XOR);
1410            ASSERT(nCombineResult != ERROR && nCombineResult != NULLREGION);
1411
1412            VERIFY(this->rgn->CreateRectRgn(0, 0, 0, 0)); // Must initialise the rgn first
1413            nCombineResult = this->rgn->CombineRgn(rgn1, rgn2, RGN_OR);
1414
1415            ASSERT(nCombineResult != ERROR && nCombineResult != NULLREGION);
1416
1417            if (this->recentHlght) {
1418
1419                CRgn* rgn3 = this->GetDocument()->objects[this->recentEditor]->getHighlightExactRgn(this->GetScrollPosition().x, this->GetScrollPosition().y);
1420
1421                nCombineResult = this->rgn->CombineRgn(this->rgn, rgn3, RGN_OR);
1422
1423                ASSERT(nCombineResult != ERROR && nCombineResult != NULLREGION);
1424            }
1425
1426        }
1427
1428        else if (this->recentHlght) {
1429
1430            CRect temp = this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine());
1431
1432            x1 = temp.left - this->GetScrollPosition().x;
1433            y1 = temp.top - this->GetScrollPosition().y;
```

```
1434                x2 = temp.right - this->GetScrollPosition().x;
1435                y2 = temp.bottom - this->GetScrollPosition().y;
1436
1437                VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1438
1439                CRgn* rgn3 = this->GetDocument()->objects[this-
                      >recentEditor]->getHighlightExactRgn(this-
                      >GetScrollPosition().x, this->GetScrollPosition().y);
1440
1441                int nCombineResult = this->rgn->CombineRgn(this->rgn,
                      rgn3, RGN_OR);
1442
1443                ASSERT(nCombineResult != ERROR && nCombineResult !=
                      NULLREGION);
1444            }
1445
1446            else {
1447
1448                x1 = (this->GetDocument()->objects[this->activeEditor]-
                      >getRecentPos() - 0.5) * this->GetDocument()->objects
                      [this->activeEditor]->getTextExtent().cx ;
1449                y1 = this->GetDocument()->objects[this->activeEditor]-
                      >getLineBounds(this->GetDocument()->objects[this-
                      >activeEditor]->getActiveLine()).top;
1450                x2 = x1 + this->GetDocument()->objects[this-
                      >activeEditor]->getTextExtent().cx;
1451                y2 = this->GetDocument()->objects[this->activeEditor]-
                      >getLineBounds(this->GetDocument()->objects[this-
                      >activeEditor]->getActiveLine()).bottom;
1452
1453                this->rgn->CreateRectRgn(x1, y1, x2, y2);
1454            }
1455
1456            break;
1457        }
1458
1459        case REMOVE_HIGHLIGHT: // Working
1460            x1 = this->GetDocument()->objects[this->activeEditor]-
                    >getBounds().left - this->GetScrollPosition().x;
1461            y1 = this->GetDocument()->objects[this->activeEditor]-
                    >getLineBounds(min(this->GetDocument()->objects[this-
                    >activeEditor]->getActiveLine(), this->GetDocument()-
                    >objects[this->activeEditor]->getStartLine())).top -
                    this->GetScrollPosition().y;
1462            x2 = this->GetDocument()->objects[this->GetDocument()-
                    >objects.GetSize() - 1]->getBounds().right - this-
                    >GetScrollPosition().x;
1463            y2 = this->GetDocument()->objects[this->GetDocument()-
                    >objects.GetSize() - 1]->getBounds().bottom - this-
                    >GetScrollPosition().y;
1464            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1465            break;
1466
1467        case CARET_MOVE: // Working
1468
1469            x1 = this->GetCaretPos().x - this->GetDocument()->objects
```

```
            [this->activeEditor]->getTextExtent().cx - this->GetScrollPosition().x;
1470        y1 = this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).top - this->GetScrollPosition().y;
1471        x2 = this->GetCaretPos().x + this->GetDocument()->objects[this->activeEditor]->getTextExtent().cx - this->GetScrollPosition().x;
1472        y2 = this->GetDocument()->objects[this->activeEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine()).bottom - this->GetScrollPosition().y;
1473
1474        VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1475        break;
1476
1477    case ADD_RGN_SIDEBAR_BTN: { // Working
1478
1479        if (type != 0 && type != 1) {
1480            throw("ERROR : type error. type must be 0 or 1");
1481        }
1482
1483        Gdiplus::Bitmap bitmap(L"res/arrow_right_click.bmp");
1484
1485        x1 = this->GetDocument()->objects[this->recentEditor]->getBounds().left + static_cast<int>(this->GetDocument()->defBoxHeight * theApp.zoom) / 2 - (bitmap.GetWidth() * theApp.zoom) / 2;
1486        y1 = this->GetDocument()->objects[this->recentEditor]->getLineBounds(this->recentBlockLine).top + static_cast<int>(this->GetDocument()->defBoxHeight * theApp.zoom) / 2 - (bitmap.GetHeight() * theApp.zoom) / 2 - this->GetScrollPosition().y;
1487        x2 = x1 + (bitmap.GetWidth() * theApp.zoom);
1488        y2 = y1 + (bitmap.GetHeight() * theApp.zoom) - this->GetScrollPosition().y;
1489
1490        if (type == 0) {
1491            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1492        }
1493        else {
1494            CRgn* rgn1 = new CRgn();
1495            VERIFY(rgn1->CreateRectRgn(x1, y1, x2, y2));
1496            this->rgn->CombineRgn(this->rgn, rgn1, RGN_OR);
1497        }
1498
1499        break;
1500    }
1501    case ADD_RGN_ACTIVE_LINE: {
1502        CRgn* rgn1 = new CRgn();
1503        CRect temp = this->GetDocument()->objects[this->recentEditor]->getLineBounds(this->GetDocument()->objects[this->activeEditor]->getActiveLine());
1504
1505        x1 = temp.left - this->GetScrollPosition().x;
```

```
1506            y1 = temp.top - this->GetScrollPosition().y;
1507            x2 = temp.right - this->GetScrollPosition().x;
1508            y2 = temp.bottom - this->GetScrollPosition().y;
1509
1510            VERIFY(rgn1->CreateRectRgn(x1, y1, x2, y2));
1511
1512            this->rgn->CombineRgn(this->rgn, rgn1, RGN_OR);
1513
1514            break;
1515        }
1516
1517        case H_SCROLL:
1518
1519            x1 = 0;
1520            y1 = this->GetDocument()->objects[0]->getBounds().top;
1521            x2 = this->GetScrollPosition().x + this->GetDocument()-
                  >lineOffset;
1522            y2 = this->GetDocument()->objects[this->GetDocument()-
                  >objects.GetSize() - 1]->getBounds().bottom;
1523
1524            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1525            break;
1526
1527        case PASTE:
1528
1529            x1 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().left - this->GetScrollPosition().x;
1530            y1 = this->GetDocument()->objects[this->activeEditor]-
                  >getLineBounds(this->GetDocument()->objects[this-
                  >activeEditor]->getActiveLine()).top - this-
                  >GetScrollPosition().y;
1531            x2 = this->GetDocument()->objects[this->activeEditor]-
                  >getBounds().right - this->GetScrollPosition().x;
1532            y2 = this->GetDocument()->objects[this->GetDocument()-
                  >objects.GetSize() - 1]->getBounds().bottom - this-
                  >GetScrollPosition().y;
1533
1534            VERIFY(this->rgn->CreateRectRgn(x1, y1, x2, y2));
1535            break;
1536        }
1537        //cout << x1 << ", " << y1 << ", " << x2 << ", " << y2 <<
              endl;// KEEP THIS FOR DEBUGGING
1538    }
1539
1540    void CTextDocView::drawHeader(CDC* pDc)
1541    {
1542        pDc->TextOut(this->GetDocument()->objects[0]->getLineBounds
              ().left, 400 - this->GetDocument()->objects[0]->getTextExtent
              ().cy,
1543            this->GetDocument()->filename, this->GetDocument()-
                  >filename.GetLength());
1544
1545        CString page;
1546        page.Format(L"%d", this->pInfo->m_nCurPage);
1547
1548        pDc->TextOut(this->pInfo->m_rectDraw.Width() - (this-
```

```
            >GetDocument()->objects[0]->getLineBounds().left - this-
            >GetDocument()->objects[0]->getBounds().left) / 2 -
            page.GetLength()* this->GetDocument()->objects[0]-
            >getTextExtent().cx, 400 - this->GetDocument()->objects[0]-
            >getTextExtent().cy, page, page.GetLength());
1549
1550        pDc->MoveTo(CPoint((this->GetDocument()->objects[0]-
            >getLineBounds().left - this->GetDocument()->objects[0]-
            >getBounds().left)/2, 400));
1551        pDc->LineTo(CPoint(this->pInfo->m_rectDraw.Width() - (this-
            >GetDocument()->objects[0]->getLineBounds().left - this-
            >GetDocument()->objects[0]->getBounds().left) / 2, 400));
1552    }
1553
```