```cpp

// CTextEditorObject.cpp : implementation of the CTextEditorObject class
//

#include "pch.h"
#include "framework.h"
// SHARED_HANDLERS can be defined in an ATL project implementing preview, thumbnail
// and search filter handlers and allows sharing of document code with that project.
#ifndef SHARED_HANDLERS
#include "DesignArk.h"
#endif

#include "CTextEditorObject.h"

// Public contructors
CTextEditorObject::CTextEditorObject(CRect bounds, CString ID,
                                     BOOL lineNums, int maxLines, int defBoxHeight,
                                     BOOL active,
    std::vector<CString> text, std::vector<int> line, int lineOffset)
    : CAppObject(bounds, ID, active)
{
    this->activeLine = 1;
    this->mouseLine = 0;
    this->blockLine = 0;
    this->printLine = 0;
    this->caretPos = text[text.size() - 1].GetLength();
    this->maxLines = maxLines;

    this->defBoxHeight = defBoxHeight;

    this->hlght = FALSE;
    this->hlghting = FALSE;
    this->hlghtStartP.x = 0; this->hlghtStartP.y = 0;
    this->hlghtEndP.x = 0; this->hlghtEndP.y = 0;

    this->lineNums = lineNums;
    this->lineOffset = lineOffset;
    this->hoverSubLine = FALSE;
    this->clickSubLine = FALSE;

    int i = 1;
    CString id;
    CRect lineBounds = bounds;
    std::vector<int> lineNumbers = line;

    if (this->lineNums) {

        lineBounds.left += (this->lineOffset * theApp.zoom);
    }

    lineBounds.bottom = lineBounds.top + static_cast<int>(this-
```

```
          >defBoxHeight * theApp.zoom);
 52
 53     for (auto& text : text) {
 54
 55         id.Format(L"%d", i-1);
 56
 57         this->lines.push_back(new CTextLineObject(lineBounds, id,
              lineNumbers, text, FALSE));
 58         lineBounds.top += static_cast<int>(this->defBoxHeight *
              theApp.zoom);
 59         lineBounds.bottom += static_cast<int>(this->defBoxHeight *
              theApp.zoom);
 60
 61         lineNumbers[lineNumbers.size()-1]++;
 62         i++;
 63     }
 64
 65     this->lines[0]->setActive(TRUE);
 66
 67     this->cursor_arrow = 0;
 68     this->lMouseDown = FALSE;
 69
 70
 71     this->setSidebar();
 72     this->setBracketsNull();
 73 }
 74 CTextEditorObject::~CTextEditorObject()
 75 {
 76     for (auto& it : this->lines) {
 77         delete it;
 78     }
 79 }
 80
 81 // Public implementations
 82 int CTextEditorObject::draw(CDC* pDc, CSize textExtent, int
      xScrollPosition, int returnNewLines, BOOL printing, CRect
      printAreaLength)
 83 {
 84     this->textExtent = pDc->GetTextExtent(L"A", 1);
 85
 86     if (printing) {
 87         this->setBracketsNull();
 88     }
 89
 90     if (this->lineNums) {
 91
 92         if (!printing) {
 93             for (auto& it : this->lines) {
 94                 returnNewLines = it->draw(pDc, this->textExtent,
                      this->brackets, returnNewLines, printing);
 95             }
 96         }
 97
 98         // Prints the sidebar
 99         if (!printing) {
100             this->sidebar.left = xScrollPosition;
```

```
101             this->sidebar.right = xScrollPosition +
                  static_cast<int>(this->defBoxHeight * theApp.zoom);
102         }
103         else {
104             this->sidebar.left = 0;
105             this->sidebar.right = 0;
106         }
107
108         if (!printing) {
109
110             CPen penWhite;
111             penWhite.CreatePen(PS_SOLID, 0, RGB(255, 255, 255));
112             CBrush brushWhite;
113             brushWhite.CreateSolidBrush(RGB(255, 255, 255));
114
115             CPen* pOldPen = pDc->SelectObject(&penWhite);
116             CBrush* oldBrush = pDc->SelectObject(&brushWhite);
117
118             pDc->Rectangle(CRect(this->sidebar.left, this-
                  >sidebar.top + (returnNewLines * this->defBoxHeight *
                   theApp.zoom), this->sidebar.left + (this->lineOffset
                   * theApp.zoom), this->sidebar.bottom + 2));
119
120             pDc->SelectObject(pOldPen);
121             pDc->SelectObject(oldBrush);
122
123             pDc->RoundRect(this->sidebar, CPoint(8 * theApp.zoom, 8
                   * theApp.zoom));
124
125             pDc->SetTextColor(RGB(50, 50, 150));
126
127             if (this->blockLine != 0) {
128
129                 Gdiplus::Graphics g(pDc->GetSafeHdc());
130
131                 if (this->lMouseDown) {
132                     Gdiplus::Bitmap bitmap(L"res/
                     arrow_right_click.bmp");
133
134                     Gdiplus::Rect expansionRect(this->sidebar.left
                     + static_cast<int>(this->defBoxHeight *
                     theApp.zoom) / 2 - (bitmap.GetWidth() *
                     theApp.zoom) / 2, this->lines[this->blockLine -
                     1]->getBounds().top + static_cast<int>(this-
                     >defBoxHeight * theApp.zoom) / 2 -
                     (bitmap.GetHeight() * theApp.zoom) / 2,
135                      (bitmap.GetWidth() * theApp.zoom),
                     (bitmap.GetHeight() * theApp.zoom));
136
137                     g.DrawImage(&bitmap, expansionRect);
138                 }
139                 else {
140                     Gdiplus::Bitmap bitmap(L"res/arrow_right.bmp");

141
142                     Gdiplus::Rect expansionRect(this->sidebar.left
```

```
                    + static_cast<int>(this->defBoxHeight *
                    theApp.zoom) / 2 - (bitmap.GetWidth() *
                    theApp.zoom) /2, this->lines[this->blockLine - 1]-
                    >getBounds().top + static_cast<int>(this-
                    >defBoxHeight * theApp.zoom) / 2 -
                    (bitmap.GetHeight() * theApp.zoom) /2,
143                                                                    
    (bitmap.GetWidth() * theApp.zoom), (bitmap.GetHeight() *
    theApp.zoom));
144
145                     g.DrawImage(&bitmap, expansionRect);
146                 }
147             }
148         }
149
150         if (printing) {
151
152             int i = this->printLine;
153             BOOL finished = FALSE;
154
155             while (!finished && i < this->lines.size()) {
156
157                 int lines = std::ceil(float(this->lines[i]->getText
                    ().GetLength()) / float((printAreaLength.Width() -
                     this->lines[i]->getBounds().left * 2) / this-
                    >textExtent.cx));
158                 if (lines == 0) {
159                     lines++;
160                 }
161
162                 if (this->lines[i]->getBounds().top + (lines +
                    returnNewLines) * this->getBoxHeight() <
                    printAreaLength.bottom) {
163
164                     pDc->SetTextColor(RGB(50, 50, 150));
165                     pDc->TextOut(this->sidebar.left + 400 - (this-
                    >lines[i]->sGetLineNums().GetLength() * this-
                    >textExtent.cx), this->lines[i]->getBounds().top +
                     this->lines[i]->getBounds().Height() / 2 - this-
                    >textExtent.cy / 2 + (returnNewLines * this-
                    >defBoxHeight * theApp.zoom), this->lines[i]-
                    >sGetLineNums(), this->lines[i]->sGetLineNums
                    ().GetLength());
166
167                     pDc->SetTextColor(RGB(0, 0, 0));
168                     returnNewLines = this->lines[i]->draw(pDc,
                    this->textExtent, this->brackets, returnNewLines,
                    printing, printAreaLength.Width());
169
170                     i++;
171                 }
172                 else {
173                     finished = TRUE;
174                 }
175
176             }
```

```
177
178             this->printLine = i;
179
180             if (this->printLine == this->lines.size()) {
181                 this->printLine = 0;
182             }
183         }
184         else {
185             for (auto& it : this->lines) {
186
187                 pDc->TextOut(this->sidebar.left + (this->lineOffset
                      * theApp.zoom) - it->sGetLineNums().GetLength() *
                      this->textExtent.cx - 2, it->getBounds().top +
                     it->getBounds().Height() / 2 - this-
                     >textExtent.cy / 2, it->sGetLineNums(), it-
                     >sGetLineNums().GetLength());
188             }
189         }
190         pDc->SetTextColor(RGB(0, 0, 0));
191     }
192     else {
193         for (int i = this->printLine; i < this->lines.size(); i++)
              {
194             returnNewLines = this->lines[i]->draw(pDc, this-
                  >textExtent, this->brackets, returnNewLines,
                  printing, printAreaLength.Width());
195         }
196     }
197
198     return returnNewLines;
199 }
200
201 // Public getters and checkers
202 int CTextEditorObject::getCaretPos()
203 {
204     return this->caretPos;
205 }
206
207 BOOL CTextEditorObject::pointHighlighted(CPoint point)
208 {
209     // TODO : Check if point is highlighted
210     return FALSE;
211 }
212 BOOL CTextEditorObject::hasHighlight()
213 {
214     return this->hlght;
215 }
216 void CTextEditorObject::hlghtingOff()
217 {
218     this->hlghting = FALSE;
219     this->hlght = FALSE;
220     this->startLine = 0;
221     this->startPos = 0;
222
223     this->recentHlght = FALSE;
224
```

```
225     if (this->active) {
226         this->lines[this->activeLine - 1]->setActive(TRUE);
227     }
228
229     for (auto& it : this->lines) {
230         it->setHighlighting(FALSE);
231     }
232 }
233
234 CRect CTextEditorObject::getHighlightClippingRect()
235 {
236     CRect rgn;
237
238     if (this->hlghting) {
239
240         if (this->recentLine == this->activeLine) {
241
242             if (!this->recentHlght && this->hlghting) {
243                 rgn = this->lines[this->activeLine - 1]->getBounds
                      ();
244             }
245             else if (this->recentPos != this->caretPos) {
246
247                 rgn = this->lines[this->activeLine - 1]->getBounds
                      ();
248                 rgn.right = rgn.left + max(this->recentPos, this-
                      >caretPos) * this->textExtent.cx + (2 *
                      theApp.zoom);
249                 rgn.left = rgn.left + min(this->recentPos, this-
                      >caretPos) * this->textExtent.cx - (2 *
                      theApp.zoom);
250             }
251         }
252         else {
253             rgn = this->lines[min(this->recentLine, this-
                    >activeLine) - 1]->getBounds();
254             rgn.bottom = this->lines[max(this->recentLine, this-
                    >activeLine) - 1]->getBounds().bottom;
255         }
256     }
257
258     return rgn;
259 }
260
261 CRgn* CTextEditorObject::getHighlightExactRgn(int x_offset, int
      y_offset)
262 {
263     CRgn* rgn = new CRgn();
264     VERIFY(rgn->CreateRectRgn(0, 0, 0, 0));
265
266     for (auto& it : this->lines) {
267
268         if(it->getHStart() != it->getHEnd()){
269
270             CRgn* rgn1 = new CRgn();
271             int x1 = (it->getHStart() * this->textExtent.cx) + it-
```

```
                >getBounds().left - (2 * theApp.zoom) - x_offset,
272               y1 = it->getBounds().top - y_offset,
273               x2 = (it->getHEnd() * this->textExtent.cx) + it-
                   >getBounds().left + (2 * theApp.zoom) - x_offset,
274               y2 = it->getBounds().bottom - y_offset;
275
276           ASSERT(rgn1->CreateRectRgn(x1, y1, x2, y2));
277
278           int nCombineResult = rgn->CombineRgn(rgn, rgn1,
                RGN_OR);
279           ASSERT(nCombineResult != ERROR && nCombineResult !=
                NULLREGION);
280
281       }
282   }
283   return rgn;
284 }
285
286 int CTextEditorObject::getStartLine()
287 {
288   return this->startLine - 1;
289 }
290 BOOL CTextEditorObject::isHlghtMultiline()
291 {
292   BOOL first = FALSE;
293
294   for (auto& it : this->lines) {
295       if (it->isHighlighting()) {
296           if (!first) {
297               first = TRUE;
298           }
299           else {
300               return TRUE;
301           }
302       }
303   }
304   return FALSE;
305 }
306
307 int CTextEditorObject::lineHighlight(int line)
308 {
309   if (this->lines[line]->isHighlighting()) {
310       if (this->lines[line]->isLineHighlighted()) {
311           return 2;
312       }
313       return 1;
314   }
315   return 0;
316 }
317
318 CPoint CTextEditorObject::getCaretPoint(CSize caretSize)
319 {
320   CPoint caretPoint;
321   BOOL els = TRUE;
322
323   int offset = 0;
```

```
324
325     if (this->lines[this->activeLine - 1]->isHighlighting() &&
          (this->lines[this->activeLine - 1]->getLength() == 0)) {
326         offset = 1;
327     }
328
329     if (this->caretPos >= this->lines[this->activeLine - 1]-
          >getLength() + offset) {
330         caretPoint.x = (this->lines[this->activeLine - 1]-
              >getLength() + offset) * (this->textExtent.cx) + (this-
              >lineOffset * theApp.zoom) + 1;
331     }
332     else {
333         caretPoint.x = (this->caretPos) * this->textExtent.cx +
              (this->lineOffset * theApp.zoom) + 1;
334     }
335
336     caretPoint.y = this->bounds.top + ((this->activeLine - 0.5) *
          static_cast<int>(this->defBoxHeight * theApp.zoom)) -
          caretSize.cy / 2 ;
337
338     return caretPoint;
339 }
340
341 CSize CTextEditorObject::getTextExtent()
342 {
343     return this->textExtent;
344 }
345
346 void CTextEditorObject::setTextExtent(CSize size)
347 {
348     this->textExtent = size;
349 }
350
351 int CTextEditorObject::getRecentPos()
352 {
353     return this->recentPos;
354 }
355
356 void CTextEditorObject::incrementSublines(int subline, int val)
357 {
358     for (auto& it : this->lines) {
359         it->incrementLine(subline, val);
360     }
361 }
362
363 void CTextEditorObject::setActive(BOOL active)
364 {
365     this->active = active;
366
367     if (!this->active) {
368
369         this->hoverSubLine = FALSE;
370         this->clickSubLine = FALSE;
371         this->blockLine = 0;
372
```

```
373             this->hlghtingOff();
374 
375             this->lines[this->activeLine - 1]->setActive(FALSE);
376         }
377         else {
378             if (this->blockLine != 0) {
379                 this->activeLine = this->blockLine;
380             }
381             this->lines[this->activeLine - 1]->setActive(TRUE);
382             this->caretPos = 0;
383         }
384 }
385 
386 void CTextEditorObject::move(int x, int y)
387 {
388     CAppObject::move(x, y);
389     for (auto& it : this->lines) {
390         it->move(x, y);
391     }
392     this->sidebar += CPoint(x, y);
393 }
394 void CTextEditorObject::setBounds(CRect bounds)
395 {
396     this->bounds = bounds;
397     CRect newBounds = this->bounds;
398     newBounds.left = newBounds.left + static_cast<int>(this->
      >lineOffset * theApp.zoom);
399     newBounds.bottom = newBounds.top + static_cast<int>(this->
      >defBoxHeight * theApp.zoom);
400 
401     for (auto& it : this->lines) {
402 
403         it->setBounds(newBounds);
404         newBounds.top += static_cast<int>(this->defBoxHeight *
          theApp.zoom);
405         newBounds.bottom += static_cast<int>(this->defBoxHeight *
          theApp.zoom);
406     }
407 }
408 
409 std::tuple<CRect, int> CTextEditorObject::getPrintBounds(int
    returnNewLines, int printAreaLength)
410 {
411     CRect printBounds = this->lines[0]->getBounds();
412     int thisBoundsNewLines = returnNewLines;
413 
414     int printMaxCharLength = (printAreaLength - this->lines[0]-
      >getBounds().left * 2) / this->textExtent.cx;
415 
416     for (int i = 0; i < this->lines.size(); i++) {
417         thisBoundsNewLines += std::ceil(double(this->lines[i]-
          >getLength() / printMaxCharLength));
418 
419         if (i == this->printLine) {
420             printBounds.top += this->defBoxHeight * theApp.zoom *
              thisBoundsNewLines;
```

```
421             }
422         }
423
424         printBounds.right = printBounds.left + printMaxCharLength *
              this->textExtent.cx;
425         printBounds.bottom = printBounds.top + this->defBoxHeight *
              theApp.zoom * (thisBoundsNewLines - returnNewLines + this-
              >lines.size());
426
427         return std::tuple<CRect, int>(printBounds, thisBoundsNewLines);
428     }
429
430     int CTextEditorObject::getCursorArrow()
431     {
432         return this->cursor_arrow;
433     }
434     void CTextEditorObject::initialise()
435     {
436         this->caretPos = 0;
437         this->lines[this->activeLine - 1]->setActive(FALSE);
438         this->activeLine = 1;
439         this->lines[this->activeLine - 1]->setActive(TRUE);
440     }
441
442     // Public message handlers
443     void CTextEditorObject::OnSize(UINT nType, int cx, int cy)
444     {
445         this->bounds.bottom = this->bounds.top + static_cast<int>
              (defBoxHeight * theApp.zoom) * (this->lines.size());
446
447         CRect lineBounds;
448         lineBounds.left = static_cast<int>(this->lineOffset *
              theApp.zoom);
449         lineBounds.right = this->bounds.right;
450         lineBounds.top = this->bounds.top;
451         lineBounds.bottom = lineBounds.top + static_cast<int>(this-
              >defBoxHeight * theApp.zoom);
452
453         for (auto& it : this->lines) {
454
455             it->setBounds(lineBounds);
456
457             lineBounds.top += static_cast<int>(this->defBoxHeight *
                  theApp.zoom);
458             lineBounds.bottom += static_cast<int>(this->defBoxHeight *
                  theApp.zoom);
459         }
460
461         this->setSidebar();
462     }
463
464     BOOL CTextEditorObject::OnLButtonUp(UINT nFlags, CPoint point)
465     {
466         BOOL redraw = FALSE;
467
468         this->lMouseDown = FALSE;
```

```
469
470     if (this->hlghting) {
471         this->hlghting = FALSE;
472
473         if ((this->startLine == this->activeLine && this->startPos
              == this->caretPos) ||
474
475             (this->startLine == this->activeLine &&
476             this->startPos >= this->lines[this->activeLine - 1]-
                >getLength() &&
477             this->caretPos >= this->lines[this->activeLine - 1]-
                >getLength())) {
478
479
480             this->hlghtingOff();
481             redraw = TRUE;
482         }
483         else {
484             this->hlght = TRUE;
485         }
486     }
487
488     if (point.x <= this->sidebar.right) {
489         BOOL update = FALSE;
490         int i = 1;
491
492         while (!update && i <= this->lines.size()) {
493
494             if (point.y >= this->lines[i - 1]->getBounds().top &&
495                 point.y < this->lines[i - 1]->getBounds().bottom) {
496
497                 this->clickSubLine = TRUE;
498
499                 this->caretPos = 0;
500                 update = TRUE;
501
502             }
503             i++;
504         }
505
506     }
507     return redraw;
508 }
509 BOOL CTextEditorObject::OnLButtonDown(UINT nFlags, CPoint point)
510 {
511     this->hlghtingOff();
512     BOOL sidebarbtnhold = FALSE;
513     this->lMouseDown = TRUE;
514
515     BOOL update = FALSE;
516     int i = 1;
517
518     // Iterate through all the lines
519     while (!update && i <= this->lines.size()) {
520
521         // The point is in the lines
```

```
522         if (point.y >= this->lines[i - 1]->getBounds().top &&
              point.y < this->lines[i - 1]->getBounds().bottom) {
523
524             // If we have changed lines
525             if (i != this->activeLine) {
526
527                 this->lines[this->activeLine - 1]->setActive
                    (FALSE);
528                 this->activeLine = i;
529                 this->lines[i - 1]->setActive(TRUE);
530             }
531
532             // If we have selected part of the text
533             if (this->lines[i - 1]->checkPointInBounds(point)) {
534
535                 if (point.x - static_cast<int>(this->defBoxHeight *
                      theApp.zoom) - (this->lineOffset * theApp.zoom)
                     >= this->lines[i - 1]->getLength() * this-
                     >textExtent.cx) {
536                     this->caretPos = this->lines[i - 1]->getLength
                     ();
537                 }
538                 else {
539                     this->caretPos = (point.x - (this->lineOffset *
                      theApp.zoom)) / this->textExtent.cx;
540                 }
541             }
542
543             // If we have selected the side of the text editor
544             else {
545
546                 // If we selected a button
547                 if (point.x < static_cast<int>(this->defBoxHeight *
                      theApp.zoom)) {
548                     this->blockLine = i;
549                     sidebarbtnhold = TRUE;
550                 }
551
552                 // If we select the bit that isn't the sidebar
553                 else {
554                     this->hlghtingOff();
555                     this->hlght = TRUE;
556                     this->lines[this->activeLine - 1]-
                     >highlightLine();
557                 }
558                 this->caretPos = 0;
559             }
560
561             update = TRUE;
562         }
563
564         i++;
565     }
566
567     if (!update && point.x >= static_cast<int>(this->defBoxHeight *
          theApp.zoom) + (this->lineOffset * theApp.zoom)) {
```

```
568             this->lines[this->activeLine - 1]->setActive(FALSE);
569             this->activeLine = this->lines.size();
570             this->lines[this->activeLine - 1]->setActive(TRUE);
571             this->caretPos = this->lines[this->activeLine - 1]->getLength();
572         }
573
574         this->hlghtStartP = point;
575         this->setBracketsNull();
576
577         return sidebarbtnhold;
578     }
579     void CTextEditorObject::OnLButtonDblClk(UINT nFlags, CPoint point)
580     {
581         int leftpos = this->caretPos - 1, rightpos = this->caretPos;
582         BOOL leftstop = FALSE, rightstop = FALSE;
583
584         CString lineText = this->lines[this->activeLine - 1]->getText();
585
586         while (!(leftstop && rightstop) && !(leftpos == 0 && rightpos == lineText.GetLength())) {
587
588             if (leftpos == 0) {
589                 leftstop = TRUE;
590             }
591             else if(lineText.Mid(leftpos, 1) == L" " && !leftstop) {
592                 leftpos++;
593                 leftstop = TRUE;
594             }
595             else if(!leftstop) {
596                 leftpos--;
597             }
598
599             if ((lineText.Mid(rightpos, 1) == L" " && !rightstop) || rightpos == lineText.GetLength()) {
600                 rightstop = TRUE;
601             }
602             else if(!rightstop) {
603                 rightpos++;
604             }
605         }
606
607         this->lines[this->activeLine - 1]->setHighlighter(leftpos, rightpos);
608         this->hlght = TRUE;
609
610         this->caretPos = rightpos;
611     }
612     void CTextEditorObject::OnRButtonUp(UINT nFlags, CPoint point)
613     {
614     }
615     void CTextEditorObject::OnRButtonDown(UINT nFlags, CPoint point)
616     {
617
618         if (!this->hlght) {
```

```
619
620         BOOL update = FALSE;
621         int i = 1;
622
623         while (!update && i <= this->lines.size()) {
624
625             if (this->lines[i - 1]->checkPointInBounds(point)) {
626
627                 this->lines[this->activeLine - 1]->setActive
                     (FALSE);
628                 this->activeLine = i;
629                 this->lines[i - 1]->setActive(TRUE);
630
631                 if (point.x - static_cast<int>(this->defBoxHeight *
                       theApp.zoom) - (this->lineOffset * theApp.zoom)
                      >= this->lines[i - 1]->getLength() * this-
                      >textExtent.cx) {
632                     this->caretPos = this->lines[i - 1]->getLength
                      ();
633                 }
634                 else {
635                     this->caretPos = (point.x - (this->lineOffset *
                      theApp.zoom)) / this->textExtent.cx;
636                 }
637                 update = TRUE;
638             }
639
640             i++;
641         }
642
643     }
644
645     //if (!update && point.x >= static_cast<int>(this->defBoxHeight
           * theApp.zoom)) {
646     //  this->lines[this->activeLine - 1]->setActive(FALSE);
647     //  this->activeLine = this->lines.size();
648     //  this->lines[this->activeLine - 1]->setActive(TRUE);
649     //  this->caretPos = this->lines[this->activeLine - 1]-
          >getLength();
650     //}
651
652     this->setBracketsNull();
653 }
654 BOOL CTextEditorObject::OnMouseMove(UINT nFlags, CPoint point)
655 {
656     BOOL mUpdate = FALSE, bUpdate = FALSE;
657     BOOL brk = FALSE;
658
659     int i = 1;
660     BOOL redraw = FALSE;
661
662     while (!mUpdate && !brk && i <= this->lines.size()) {
663
664         if (point.y >= this->lines[i - 1]->getBounds().top &&
              point.y < this->lines[i - 1]->getBounds().bottom) {
665
```

```
666             this->mouseLine = i;
667             mUpdate = TRUE;
668
669             if (this->lineNums) {
670
671                 if (!this->lines[i - 1]->checkPointInBounds(point))
                    {
672
673                     if (point.x <= this->sidebar.right) {
674
675                         if (this->blockLine != i) {
676
677                             redraw = TRUE;
678                             this->blockLine = i;
679                         }
680                         bUpdate = TRUE;
681                         this->cursor_arrow = 2;
682                     }
683                     else {
684                         this->cursor_arrow = 1;
685                     }
686                 }
687                 else {
688                     this->cursor_arrow = 0;
689                 }
690             }
691
692             brk = TRUE;
693         }
694         i++;
695     }
696     if (!mUpdate) {
697         this->mouseLine = 0;
698         this->cursor_arrow = 1;
699     }
700     if (!bUpdate) {
701         if (this->blockLine != 0) {
702             redraw = TRUE;
703         }
704         this->blockLine = 0;
705     }
706
707     if (nFlags == MK_LBUTTON) {
708
709         if (this->checkPointInBounds(point)) {
710             this->hlghtEndP = point;
711             this->setHighlighter();
712         }
713         else if (point.y < this->bounds.top) {
714             this->hlghtEndP = CPoint(this->lines[0]->getBounds
                  ().left + 1, this->lines[0]->getBounds().top +
                  static_cast<int>(this->defBoxHeight *
                  theApp.zoom) /2);
715         }
716         else {
717             this->hlghtEndP = CPoint(this->lines[this->lines.size()
```

```
                    - 1]->getLength()*this->textExtent.cx + 1, this-
                   >lines[this->lines.size() - 1]->getBounds().top +
                   static_cast<int>(this->defBoxHeight * theApp.zoom) /
                   2);
718             }
719             redraw = TRUE;
720         }
721         else {
722             this->hlghting = FALSE;
723         }
724
725         return redraw;
726     }
727
728     BOOL CTextEditorObject::OnContextMenu(CWnd* pWnd, CPoint point)
729     {
730         // TODO : Allow restriccted choice in context menu when
              selecting point that isnt highlighted
731     #ifndef SHARED_HANDLERS
732         theApp.GetContextMenuManager()->ShowPopupMenu(IDR_POPUP_EDIT,
              point.x, point.y, pWnd, TRUE);
733     #endif
734         return TRUE;
735     }
736
737     int CTextEditorObject::OnRecieveText(CString text, BOOL open)
738     {
739         int itPos; // This is needed as it gets changed in this-
              >brackContains
740         int carridgeOccur = 0;
741
742         if ((text == L")" && this->bracketContains(this->caretPos,
              itPos, 0, 1)) ||
743             (text == L"}" && this->bracketContains(this->caretPos,
                itPos, 1, 1)) ||
744             (text == L"]" && this->bracketContains(this->caretPos,
                itPos, 2, 1)) ||
745             (text == L"\"" && this->bracketContains(this->caretPos,
                itPos, 3, 1))||
746             (text == L"'" && this->bracketContains(this->caretPos,
                itPos, 4, 1))) {
747
748             this->brackets.erase(this->brackets.begin() + itPos);
749             this->brackets.shrink_to_fit();
750
751             this->caretPos++;
752         }
753
754         else {
755             int carridge = text.Find(L"\n");
756             CString recursiveText;
757             if (carridge != -1) {
758
759                 recursiveText = text.Mid(carridge + 1);
760                 text = text.Mid(0, carridge);
761             }
```

```
762         this->lines[this->activeLine - 1]->concatenateString(text,
              this->caretPos);
763
764         if (text == L"(") {
765
766             this->lines[this->activeLine - 1]->concatenateString
                  (L")", this->caretPos + 1);
767             this->moveBrackets(2, this->caretPos);
768
769             this->brackets.push_back({ this->caretPos, this-
                  >caretPos + 1, 0 });
770         }
771         else if (text == L"{") {
772
773             this->lines[this->activeLine - 1]->concatenateString
                  (L"}", this->caretPos + 1);
774             this->moveBrackets(2, this->caretPos);
775
776             this->brackets.push_back({ this->caretPos, this-
                  >caretPos + 1 , 1 });
777         }
778         else if (text == L"[") {
779
780             this->lines[this->activeLine - 1]->concatenateString
                  (L"]", this->caretPos + 1);
781             this->moveBrackets(2, this->caretPos);
782
783             this->brackets.push_back({ this->caretPos, this-
                  >caretPos + 1 , 2 });
784         }
785         else if (text == L"\"") {
786
787             this->lines[this->activeLine - 1]->concatenateString
                  (L"\"", this->caretPos + 1);
788             this->moveBrackets(2, this->caretPos);
789
790             this->brackets.push_back({ this->caretPos, this-
                  >caretPos + 1 , 3 });
791         }
792         else if (text == L"'") {
793
794             this->lines[this->activeLine - 1]->concatenateString
                  (L"'", this->caretPos + 1);
795             this->moveBrackets(2, this->caretPos);
796
797             this->brackets.push_back({ this->caretPos, this-
                  >caretPos + 1 , 4 });
798         }
799         else {
800             this->moveBrackets(text.GetLength(), this->caretPos);
801         }
802         this->caretPos += text.GetLength();
803
804         if (carridge != -1) {
805
806             this->OnRecieveReturn(open);
```

```
807             carridgeOccur++;
808             carridgeOccur += this->OnRecieveText(recursiveText);
809         }
810     }
811     return carridgeOccur;
812 }
813 void CTextEditorObject::OnRecieveBackspace()
814 {
815     if (!this->hlght && this->activeLine > 0 && this->caretPos != 0) { // Remove a character
816 
817         CString activeChar = this->lines[this->activeLine - 1]->getText().Mid(this->caretPos - 1, 1);
818         int itPos;
819 
820         if (((activeChar == L"(" || activeChar == L"{" || activeChar == L"[" || activeChar == L"\"" || activeChar == L"'") && this->bracketContains(this->caretPos - 1, itPos, 5, 0))){
821             this->lines[this->activeLine - 1]->backspace(this->brackets[itPos][1]);
822             this->moveBrackets(-1, this->brackets[itPos][1]);
823 
824             this->brackets.erase(this->brackets.begin() + itPos);
825             this->brackets.shrink_to_fit();
826         }
827 
828         else {
829 
830             CString spaces = L"";
831 
832             if ((this->caretPos % theApp.indentSize) == 0) {
833                 spaces = L"    ";
834 
835                 if (this->lines[this->activeLine - 1]->getText().Mid(this->caretPos - theApp.indentSize, theApp.indentSize) == spaces) {
836 
837                     for (int j = 0; j < theApp.indentSize-1; j++) {
838 
839                         this->lines[this->activeLine - 1]->backspace(this->caretPos);
840                         this->moveBrackets(-1, this->caretPos);
841                         this->caretPos--;
842                     }
843                 }
844             }
845             else {
846                 for (int i = 0; i < (this->caretPos % theApp.indentSize); i++) {
847                     spaces += L" ";
848                 }
849 
850                 if (this->lines[this->activeLine - 1]->getText().Mid(this->caretPos - (this->caretPos % theApp.indentSize), (this->caretPos %
```

```
                    theApp.indentSize)) == spaces) {
851
852                     int it = (this->caretPos % theApp.indentSize) -
                     1;
853
854                     for (int j = 0; j < it; j++) {
855
856                         this->lines[this->activeLine - 1]-
                    >backspace(this->caretPos);
857                         this->moveBrackets(-1, this->caretPos);
858                         this->caretPos--;
859                     }
860                 }
861             }
862         }
863         this->lines[this->activeLine - 1]->backspace(this-
              >caretPos);
864         this->moveBrackets(-1, this->caretPos);
865         this->caretPos--;
866     }
867
868     else if (!this->hlghting && !this->hlght && this->activeLine >
          1 && this->caretPos == 0) { // Remove a line
869
870         this->bounds.bottom -= static_cast<int>(this->defBoxHeight
              * theApp.zoom);
871
872         this->caretPos = this->lines[this->activeLine - 2]-
              >getLength();
873         this->lines[this->activeLine - 2]->concatenateString(this-
              >lines[this->activeLine - 1]->getText(), this->caretPos);
874
875         for (int i = this->lines.size() - 1; i >= this->activeLine;
               i--) {
876             this->lines[i]->move(0, -static_cast<int>(this-
                  >defBoxHeight * theApp.zoom));
877             this->lines[i]->setID(this->lines[i - 1]->getID());
878             this->lines[i]->incrementLine(this->lines[i]-
                  >iGetLineNums().size() - 1, -1);
879         }
880
881         this->lines.erase(this->lines.begin() + this->activeLine -
              1);
882         this->activeLine--;
883         this->lines[this->activeLine - 1]->setActive(TRUE);
884
885         this->setSidebar();
886     }
887
888     else if (this->hlght) { // Remove a highlight
889
890         BOOL first = FALSE;
891         int i = 0;
892         BOOL end = FALSE;
893
894         while (!end && i < this->lines.size()) { // Iterate through
```

```
                all the lines
895
896             if (this->lines[i]->isHighlighting()) {  // If the line is highlighting
897
898                 if (!first) { // If this is the first line that we have found
899
900                     this->activeLine = i + 1; // This will become our active line
901                     this->caretPos = this->lines[i]->getHStart(); // Set caret at the end of the line
902                     this->lines[i]->backspace(this->caretPos); // Remove all that we have highlighted on this line
903                     this->lines[i]->setHighlighting(FALSE); // Tell it we are no longer highlighting
904
905                     first = TRUE; // Let the loop know that we have found the first highlighted line
906                 }
907                 else if(!this->lines[i]->isLineHighlighted() || i == this->lines.size() - 1) { // If the entire line is not highlighted
908
909                     this->lines[i]->backspace(this->caretPos);
910                     this->lines[this->activeLine - 1]->concatenateString(this->lines[i]->getText(), this->caretPos);
911
912                     for (int j = i; j >= this->activeLine; j--) {
913
914                         this->lines.erase(this->lines.begin() + j);
915                     }
916
917                     for (int k = this->lines.size() - 1; k >= this->activeLine; k--) {
918
919                         this->lines[k]->move(0, -static_cast<int>(this->defBoxHeight * theApp.zoom) * (i - this->activeLine + 1));
920                         this->lines[k]->setID(this->lines[k]->getID());
921                         this->lines[k]->incrementLine(this->lines[k]->iGetLineNums().size() - 1, -(i - this->activeLine + 1));
922                     }
923
924                     this->bounds.bottom -= static_cast<int>(this->defBoxHeight * theApp.zoom) * (i - this->activeLine + 1);
925
926                     end = TRUE;
927                 }
928             }
929             else if (first) {
930
```

```
931                     end = TRUE;
932                 }
933                 i++;
934             }
935 
936             this->hlghtingOff();
937             this->setSidebar();
938         }
939 }
940 void CTextEditorObject::OnRecieveTab()
941 {
942     int itPos;
943     if (this->bracketContains(this->caretPos, itPos, 5, 1)) {
944 
945         this->brackets.erase(this->brackets.begin() + itPos);
946         this->brackets.shrink_to_fit();
947 
948         this->caretPos++;
949     }
950     else {
951         CString indent = L"";
952         for (int i = 0; i < theApp.indentSize; i++) {
953             indent.Append(L" ");
954         }
955         this->OnRecieveText(indent);
956     }
957 }
958 void CTextEditorObject::OnRecieveReturn(BOOL open)
959 {
960     if (this->maxLines > this->lines.size() || this->maxLines == 0)
        { // If there is space for another line
961 
962         this->bounds.bottom += static_cast<int>(this->defBoxHeight
          * theApp.zoom);
963         CString newLineText = L"";
964         int numTab = 0;
965 
966         if (!open) {
967             BOOL cln = TRUE, checkEnd = TRUE, checkStart = TRUE;
968             int cmtPos = -1, clnPos = -1;
969 
970             int frontIt = 0;
971 
972             for (int i = this->lines[this->activeLine - 1]->getText
              ().GetLength() - 1; i >= 0; i--) {
973 
974                 if ((this->lines[this->activeLine - 1]->getText
                  ().Mid(i, 1) != L":" && this->lines[this-
                  >activeLine - 1]->getText().Mid(i, 1) != L" "/* &&
                   i == 0 // This may need put back if bugs are
                  found */) && checkEnd) {
975                     cln = FALSE;
976                 }
977                 else if (this->lines[this->activeLine - 1]->getText
                  ().Mid(i, 1) == L":" && cln) {
978                     clnPos = i;
```

```
979                     checkEnd = FALSE;
980                 }
981
982                 if ((this->lines[this->activeLine - 1]->getText
                     ().Mid(frontIt, 1) != L" " || frontIt == this-
                     >lines[this->activeLine - 1]->getText().GetLength
                     () - 1) && checkStart) {
983
984                     numTab = (frontIt - (frontIt %
                     theApp.indentSize)) / theApp.indentSize;
985                     checkStart = FALSE;
986                 }
987                 if (this->lines[this->activeLine - 1]->getText
                     ().Mid(frontIt, theApp.commentType.GetLength()) ==
                      theApp.commentType) {
988                     cmtPos = frontIt;
989                 }
990
991                 frontIt++;
992             }
993
994             if (frontIt == 0) {
995                 cln = FALSE;
996             }
997
998             if (cln && (cmtPos >= clnPos || cmtPos == -1)) {
999                 numTab++;
1000            }
1001
1002            CString indent = L"";
1003            for (int i = 0; i < theApp.indentSize; i++) {
1004                indent.Append(L" ");
1005            }
1006
1007            for (int i = 0; i < numTab; i++) {
1008                newLineText.Append(indent);
1009            }
1010        }
1011        newLineText.Append(this->lines[this->activeLine - 1]-
            >getText().Mid(this->caretPos));
1012
1013        auto itpos = this->lines.begin() + this->activeLine;
1014
1015        auto newit = this->lines.insert(
1016            itpos,
1017            new CTextLineObject(
1018                this->lines[this->activeLine - 1]->getBounds(),
1019                L"0",
1020                this->lines[this->activeLine - 1]->iGetLineNums(),
1021                newLineText,
1022                TRUE)); // Add a new line
1023
1024
1025        this->lines[this->activeLine - 1]->setText(this->lines
            [this->activeLine - 1]->getText().Mid(0, this-
            >caretPos));
```

```
1026            this->lines[this->activeLine - 1]->setActive(FALSE); // Set old line not active
1027
1028            this->caretPos = 0 + numTab * theApp.indentSize;
1029            this->activeLine++; // Change position for editing
1030
1031            CString id;
1032
1033            for (int i = this->activeLine-1; i < this->lines.size(); i++) {
1034
1035                id.Format(L"%d", i);
1036                this->lines[i]->move(0, static_cast<int>(this->defBoxHeight * theApp.zoom));
1037                this->lines[i]->setID(id);
1038                this->lines[i]->incrementLine(this->lines[i]->iGetLineNums().size()-1, 1);
1039            } // Move all lines and reset all ids
1040        }
1041
1042        this->setSidebar();
1043        this->setBracketsNull();
1044    }
1045
1046    void CTextEditorObject::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
1047    {
1048        switch (nChar) {
1049
1050        case VK_LEFT:
1051            if (this->caretPos > 0) {
1052                this->caretPos--;
1053
1054                int itPos;
1055
1056                if (this->bracketContains(this->caretPos, itPos, 5, 0)) {
1057
1058                    this->setBracketsNull();
1059                }
1060            }
1061            else if (this->caretPos == 0 && this->activeLine > 1) {
1062
1063                this->lines[this->activeLine - 1]->setActive(FALSE);
1064                this->activeLine--;
1065                this->lines[this->activeLine - 1]->setActive(TRUE);
1066
1067                this->setBracketsNull();
1068
1069                this->caretPos = this->lines[this->activeLine - 1]->getLength();
1070            }
1071            break;
1072
1073        case VK_UP:
1074            if (this->activeLine > 1) {
```

```
1075                this->lines[this->activeLine - 1]->setActive(FALSE);
1076                this->activeLine--;
1077                this->lines[this->activeLine - 1]->setActive(TRUE);
1078
1079                this->setBracketsNull();
1080            }
1081            break;
1082
1083        case VK_RIGHT:
1084            if (this->caretPos < this->lines[this->activeLine - 1]->getLength()) {
1085
1086                int itPos;
1087
1088                if (this->bracketContains(this->caretPos, itPos, 5, 1)) {
1089
1090                    this->brackets.erase(this->brackets.begin() + itPos);
1091                    this->brackets.shrink_to_fit();
1092                }
1093
1094                this->caretPos++;
1095            }
1096            else if (this->caretPos == this->lines[this->activeLine - 1]->getLength() && this->activeLine != this->lines.size()) {
1097
1098                this->lines[this->activeLine - 1]->setActive(FALSE);
1099                this->activeLine++;
1100                this->lines[this->activeLine - 1]->setActive(TRUE);
1101
1102                this->caretPos = 0;
1103            }
1104
1105            break;
1106
1107        case VK_DOWN:
1108            if (this->activeLine < this->lines.size()) {
1109
1110                this->lines[this->activeLine - 1]->setActive(FALSE);
1111                this->activeLine++;
1112                this->lines[this->activeLine - 1]->setActive(TRUE);
1113
1114                this->setBracketsNull();
1115            }
1116
1117            break;
1118
1119        }
1120    }
1121
1122    // Public Queries
1123    std::vector <int> CTextEditorObject::iGetLineNum(int a)
1124    {
1125        if (a != 0) {
```

```
1126            return this->lines[a-1]->iGetLineNums();
1127        }
1128
1129        else if (this->clickSubLine) {
1130            this->clickSubLine = FALSE;
1131            return this->lines[this->activeLine-1]->iGetLineNums();
1132        }
1133        return std::vector <int> { 0 };
1134    }
1135    CString CTextEditorObject::sGetLineNum(int a)
1136    {
1137        if (a != 0) {
1138            return this->lines[a - 1]->sGetLineNums();
1139        }
1140
1141        else if (this->clickSubLine) {
1142            this->clickSubLine = FALSE;
1143            return this->lines[this->activeLine - 1]->sGetLineNums();
1144        }
1145        return L"0";
1146    }
1147
1148    int CTextEditorObject::getActiveLine()
1149    {
1150        return this->activeLine-1;
1151    }
1152    int CTextEditorObject::getBlockLine()
1153    {
1154        return this->blockLine - 1;
1155    }
1156    int CTextEditorObject::getPrintLine()
1157    {
1158        return this->printLine;
1159    }
1160    CString CTextEditorObject::getLineText(int line)
1161    {
1162        if (line == 0) {
1163            return this->lines[this->activeLine - 1]->getText();
1164        }
1165        return this->lines[line - 1]->getText();
1166    }
1167    CRect CTextEditorObject::getLineBounds(int line)
1168    {
1169        return this->lines[line]->getBounds();
1170    }
1171    CString CTextEditorObject::getHighlightedText()
1172    {
1173        if (this->hlght || this->hlghting) {
1174
1175            int i = 0;
1176            BOOL done = FALSE;
1177
1178            CString selectedText = L"";
1179
1180            while (!done && i < this->lines.size()) {
1181
```

```
1182                if (this->lines[i]->isHighlighting()) {
1183
1184                    int j = i;
1185                    BOOL it = TRUE;
1186
1187                    while (it && this->lines[j]->isHighlighting()) {
1188
1189                        selectedText.Append(this->lines[j]->getHighlightedText());
1190                        selectedText.Append(L"\n");
1191
1192                        if (j == this->lines.size() - 1) {
1193                            it = FALSE;
1194                        }
1195                        else {
1196                            j++;
1197                        }
1198                    }
1199
1200                    selectedText = selectedText.Mid(0, selectedText.GetLength() - 1);
1201                    done = TRUE;
1202                }
1203                i++;
1204            }
1205
1206            return selectedText;
1207        }
1208        else {
1209            return CString();
1210        }
1211    }
1212    int CTextEditorObject::getLineTextWidth()
1213    {
1214        int max = this->lines[0]->getText().GetLength();
1215
1216        for (int i = 1; i < this->lines.size(); i++) {
1217            if (max < this->lines[i]->getText().GetLength()) {
1218                max = this->lines[i]->getText().GetLength();
1219            }
1220        }
1221
1222        max *= this->textExtent.cx;
1223        max += 2;
1224
1225        return max;
1226    }
1227
1228    int CTextEditorObject::getNumLines()
1229    {
1230        return this->lines.size();
1231    }
1232
1233    int CTextEditorObject::getBoxHeight(BOOL default)
1234    {
1235        if (default) {
```

```
1236            return this->defBoxHeight;
1237        }
1238        return static_cast<int>(this->defBoxHeight * theApp.zoom);
1239    }
1240
1241    void CTextEditorObject::setSidebar()
1242    {
1243        if (this->lineNums) {
1244            this->sidebar.top = this->bounds.top;
1245            this->sidebar.left = this->bounds.left;
1246            this->sidebar.bottom = this->bounds.top + this->lines.size
                  () * static_cast<int>(this->defBoxHeight * theApp.zoom);
1247            this->sidebar.right = this->bounds.left + static_cast<int>
                  (this->defBoxHeight * theApp.zoom);
1248        }
1249    }
1250    void CTextEditorObject::setHighlighter()
1251    {
1252        // Find which lines the hlght points lie in
1253        int i = 1;
1254        BOOL startP = FALSE, endP = FALSE;
1255        int endLine = 0;
1256        int endPos = 0;
1257
1258        if (this->recentLine != this->activeLine) {
1259            this->recentLine = this->activeLine;
1260        }
1261        if (this->recentPos != this->caretPos) {
1262            this->recentPos = this->caretPos;
1263        }
1264        if (this->recentHlght != this->hlghting) {
1265            this->recentHlght = this->hlghting;
1266        }
1267
1268        while (!(startP && endP) && i <= this->lines.size()) {
1269
1270            if (!startP && this->hlghtStartP.y >= this->lines[i - 1]-
                  >getBounds().top && this->hlghtStartP.y < this->lines[i -
                   1]->getBounds().bottom && !this->hlghting) {
1271
1272                this->startLine = i;
1273                if (this->hlghtStartP.x > (this->lineOffset *
                      theApp.zoom)) {
1274                    this->startPos = (this->hlghtStartP.x - (this-
                          >lineOffset * theApp.zoom)) / this->textExtent.cx;
1275                }
1276                else {
1277                    this->startPos = 0;
1278                }
1279
1280                this->hlghting = TRUE;
1281                startP = TRUE;
1282            }
1283
1284            if (!endP && this->hlghtEndP.y >= this->lines[i - 1]-
                  >getBounds().top && this->hlghtEndP.y < this->lines[i -
```

```
                1]->getBounds().bottom) {
1285
1286                endLine = i;
1287                if (this->hlghtEndP.x > (this->lineOffset *
                      theApp.zoom)) {
1288                    endPos = (this->hlghtEndP.x - (this->lineOffset *
                        theApp.zoom)) / this->textExtent.cx;
1289                }
1290                else {
1291                    endPos = 0;
1292                }
1293
1294                endP = TRUE;
1295            }
1296
1297            i++;
1298        }
1299        if (!endP) {
1300            endLine = this->lines.size();
1301            endPos = this->lines[endLine - 1]->getLength();
1302        }
1303
1304        // Unhighlight all lines up to start line and after end line
1305        for (int i = 0; i < this->startLine - 1; i++) {
1306            this->lines[i]->setHighlighting(FALSE);
1307        }
1308
1309        for (int i = endLine; i < this->lines.size(); i++) {
1310            this->lines[i]->setHighlighting(FALSE);
1311        }
1312
1313        // If we need to highlight over multiple lines
1314        if (this->startLine != endLine) {
1315
1316            // Reset the caret on the highlighter
1317            this->lines[this->activeLine - 1]->setActive(FALSE);
1318            this->activeLine = endLine;
1319
1320            // If we need to highlight in reverse
1321            if (this->startLine > endLine) {
1322
1323                if (endPos >= this->lines[endLine - 1]->getLength() +
                      1) {
1324                    endPos = this->lines[endLine - 1]->getLength();
1325                }
1326
1327                this->lines[endLine - 1]->setHighlighter(endPos, this-
                      >lines[endLine - 1]->getLength() + 1);
1328                this->lines[endLine - 1]->setHighlighting(TRUE);
1329
1330                // To make sure the highlight doesn't go past the
                      length of the last line
1331                if (this->startPos > this->lines[this->startLine - 1]-
                      >getLength()) {
1332                    if (this->lines[this->startLine - 1]->getLength()
                        == 0) {
```

```
1333                        this->startPos = 2;
1334                    }
1335                    else {
1336                        this->startPos = this->lines[this->startLine -
                             1]->getLength() + 1;
1337                    }
1338                }
1339
1340                this->lines[this->startLine - 1]->setHighlighter(0,
                      this->startPos);
1341                this->lines[this->startLine - 1]->setHighlighting
                      (TRUE);
1342
1343                // Highlight all lines inbetween end and start line
1344                for (int i = endLine + 1; i < this->startLine; i++) {
1345
1346                    this->lines[i - 1]->highlightLine();
1347                    this->lines[i - 1]->setHighlighting(TRUE);
1348                }
1349            }
1350            else {
1351
1352                this->lines[this->startLine - 1]->setHighlighter(this-
                      >startPos, this->lines[this->startLine - 1]-
                      >getLength() + 1);
1353                this->lines[this->startLine - 1]->setHighlighting
                      (TRUE);
1354
1355                // To make sure the highlight doesn't go past the
                      length of the last line
1356                if (endPos > this->lines[this->activeLine - 1]-
                      >getLength()) {
1357                    if (this->lines[this->activeLine - 1]->getLength()
                         == 0) {
1358                        endPos = 1;
1359                    }
1360                    else {
1361                        endPos = this->lines[this->activeLine - 1]-
                         >getLength();
1362                    }
1363                }
1364
1365                this->lines[endLine - 1]->setHighlighter(0, endPos);
1366                this->lines[endLine - 1]->setHighlighting(TRUE);
1367
1368                // Highlight all lines inbetween start and end line
1369                for (int i = startLine + 1; i < endLine; i++) {
1370
1371                    this->lines[i - 1]->highlightLine();
1372                    this->lines[i - 1]->setHighlighting(TRUE);
1373                }
1374            }
1375            this->caretPos = endPos;
1376        }
1377
1378        // If it is just a one line highlight
```

```
1379      else {
1380
1381          if (this->startPos > this->lines[this->startLine - 1]->getLength() && endPos > this->lines[this->startLine - 1]->getLength()) {
1382              this->hlghting = FALSE;
1383          }
1384
1385          else {
1386
1387              if (this->startPos > this->lines[this->startLine - 1]->getLength()) {
1388                  this->startPos = this->lines[this->startLine - 1]->getLength();
1389              }
1390              if (endPos > this->lines[this->startLine - 1]->getLength()) {
1391                  endPos = this->lines[this->startLine - 1]->getLength();
1392                  this->caretPos = endPos + 1;
1393              }
1394              else {
1395                  this->caretPos = endPos;
1396              }
1397
1398              this->lines[this->startLine - 1]->setHighlighter(this->startPos, endPos);
1399              this->lines[this->startLine - 1]->setHighlighting(TRUE);
1400
1401              this->activeLine = this->startLine;
1402          }
1403      }
1404  }
1405  void CTextEditorObject::setBracketsNull()
1406  {
1407      for (auto& it : this->brackets) {
1408          it = std::vector<int>{};
1409      }
1410  }
1411
1412  BOOL CTextEditorObject::bracketContains(int value, int& itPos, int type, int side)
1413  {
1414      int i = 0;
1415
1416      for (auto& it : this->brackets) {
1417
1418          if (it[side] == value) {
1419              if (it[2] == type || type == 5) {
1420                  itPos = i;
1421                  return TRUE;
1422              }
1423          }
1424          i++;
1425      }
```

```
1426      return FALSE;
1427  }
1428  void CTextEditorObject::moveBrackets(int val, int index)
1429  {
1430      for (auto& bracket : this->brackets) {
1431
1432          if (index <= bracket[0]) {
1433              bracket[0] += val;
1434              bracket[1] += val;
1435          }
1436          else if (index <= bracket[1]) {
1437              bracket[1] += val;
1438          }
1439      }
1440  }
1441
1442
```