

```

1  #include "pch.h"
2  #include "CTextLineObject.h"
3
4  // Public constructors
5  CTextLineObject::CTextLineObject(CRect bounds, CString ID,           ↗
    std::vector<int> lineNums, CString text, BOOL active)
6      : CAppObject(bounds, ID, active)
7  {
8      this->text = text;
9
10     this->highlighter.Y = bounds.top;
11     this->highlighter.Height = bounds.Height();
12     this->highlight = FALSE;
13     hStart = 0;
14     hEnd = 0;
15
16     this->numSubLines = lineNums.size();
17     this->lineNums = lineNums;
18
19     this->smartColour = TRUE;
20 }
21 CTextLineObject::~CTextLineObject()
22 {
23 }
24
25 // Public commands
26 int CTextLineObject::draw(CDC* pDc, CSize textExtent,           ↗
    std::vector<vector<int>> brackets, int returnNewLines, BOOL   ↗
    printing, int printAreaLength)
27 {
28     // Local variables
29     BOOL autoBrkt = brackets.size() > 0; // If we have any auto   ↗
        brackets (used when not printing)
30     int lengthPrint = 0; // Stores the current length of the print ↗
        on the line that is currently being drawn to (only used when ↗
        printing)
31
32     // Gdiplus::Bitmap newLineArrow(L"res/newLine.png"); // Taking ↗
        this out as it does not work and is not needed
33     int printMaxCharLength = (printAreaLength - this->bounds.left * ↗
        2)/textExtent.cx;
34
35     // If we art using SmartColour
36     if (this->smartColour) {
37
38         // Local variables
39         CString word = L""; // Stores the word that is being found ↗
            and is being colour checked
40         CString character = L""; // Stores any characters that ↗
            should seperate words
41         int pos = 0; // Stores the position that is being drawn at
42         BOOL comment = FALSE; // Stores whether the rest of the line ↗
            is a comment
43         int i = 0; // iterator for the loop, letter position that is ↗
            being looked at
44

```

```

45     // Iterate through characters in the line
46     while (i < this->text.GetLength() + 1 && !comment) {
47
48         // Checks to see if there is a comment
49         if (this->text.Mid(i, theApp.commentType.GetLength()) !=
theApp.commentType) {
50
51             // Character of the current position of the loop
52             character = this->text.Mid(i, 1);
53             BOOL hit = FALSE;
54
55             // Checks to see if the character is a 'break'
character - ie, if the character should separate
words - or if we are at the end of the loop
56             if (character == L" " ||
57                 character == L"!" ||
58                 character == L"£" ||
59                 character == L"$" ||
60                 character == L"%" ||
61                 character == L"^" ||
62                 character == L"&" ||
63                 character == L"*" ||
64                 character == L"(" ||
65                 character == L")" ||
66                 character == L"+" ||
67                 character == L"-" ||
68                 character == L"/" ||
69                 character == L":" ||
70                 character == L";" ||
71                 character == L"=" ||
72                 character == L"<" ||
73                 character == L"," ||
74                 character == L"_" ||
75                 character == L"-" ||
76                 character == L">" ||
77                 character == L"." ||
78                 character == L"?" ||
79                 character == L"/" ||
80                 character == L"@" ||
81                 character == L"{" ||
82                 character == L"[" ||
83                 character == L"|" ||
84                 character == L"\" ||
85                 character == L"}" ||
86                 character == L"]" ||
87                 i == this->text.GetLength()) {
88
89                 BOOL newColour = FALSE; // Stores if we have
changed the colour, and if it needs to be changed
back
90
91                 // Checks if the word is a number
92                 if (this->onlyNums(word)) {
93                     // If so, sets the word to the number colour
94                     pDc->SetTextColor(theApp.numberColour);
95                     newColour = TRUE;

```

```
96         }
97         else { // If the word is not a number colour
98
99             // Binary search through all of the words to
100             find if the current word matches one in the list
101             int it = 0, n =
102             theApp.smartColour_String.size();
103
104             while (!newColour && it < n) {
105                 // Binary Search
106                 int it2 = 0, n2 =
107                 theApp.smartColour_String[it].size();
108                 int min = 0, max = (n2)-1;
109
110                 while (!newColour && it2 < n2) {
111                     if (max < min) {
112                         newColour == TRUE;
113                     }
114
115                     int guess = floor((min + max) / 2);
116
117                     CString tempWord = word;
118                     CString tempAppWord =
119                     theApp.smartColour_String[it][guess];
120
121                     tempWord.MakeLower();
122                     tempAppWord.MakeLower();
123
124                     // If this word on the list matches
125                     our searching word
126                     if (tempWord == tempAppWord) {
127                         newColour = TRUE;
128                         pDc->SetTextColor
129                         (theApp.smartColour_Colour[it]); // Set the word's
130                         colour
131                     }
132                     else if (theApp.smartColour_String
133                     [it][guess] < word) {
134                         min = guess + 1;
135                     }
136                     else {
137                         max = guess - 1;
138                     }
139                     it2++;
140                 }
141                 it++;
142             }
143
144             // Drawing the word
145
146             // Drawing if we have a printer
147             if (printing) {
```

```

144          // Add the length if the word to the length of our print
145          lengthPrint += word.GetLength();
146
147          while (lengthPrint > printMaxCharLength)
148          { // As a word could technically have a length greater than 50, we must use a while to check for lines
149
150              CString tempWord = L"";
151              if ((i - word.GetLength()) % printMaxCharLength != 0) {
152                  tempWord = word.Mid(0, printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Create a temporary word to store the start of the word that will go on the end of the current line
153                  word = word.Mid(printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Change the word to be whatever we didn't print and continue the loop
154              }
155              else {
156                  pos++;
157              }
158
159              pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line
160
161              lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
162
163              /*pos += tempWord.GetLength() + 2;
164
165              Gdiplus::Graphics g(pDc->GetSafeHdc());
166              Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(), textExtent.cx, textExtent.cy);
167
168              g.DrawImage(&newLineArrow, expansionRect);*/
169
170              returnNewLines++;
171              pos = 0;
172          }
173          // Print the 'scraps' of the word at the end on the next line
174          pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this-

```

```

>bounds.Height() / 2 - textExtent.cy / 2 +
returnNewLines * this->bounds.Height(), word,
word.GetLength());
175
176         // Reset the position to draw from
177         pos += word.GetLength();
178     }
179     // Not using printer
180     else {
181         pDc->TextOut(this->bounds.left + 2 +
(textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2, word,
word.GetLength());
182
183         pos = i + 1;
184
185         /*if (character != L"\"" && character !=
L"'"") {
186             pos++;
187         }*/
188     }
189
190     word = L""; // Reset the word since we are no
longer using it
191
192     // Now we print the character that we used to
find the separation
193
194     BOOL grey = FALSE; // Stores if we used the
autobracket
195
196     // If there is a auto bracket for the character
(never used when printing)
197     if (this->active && autoBrkt && (character ==
L")" || character == L"}" || character == L"]") &&
this->brkPosContains(brackets, i) && !printing) {
198
199         grey = TRUE;
200         newColour = TRUE;
201
202         pDc->SetTextColor(RGB(128, 128, 128)); //
Set colour to grey for the auto bracket
203
204         pDc->TextOut(this->bounds.left + 3 +
(textExtent.cx * (i + word.GetLength())), this-
>bounds.top + this->bounds.Height() / 2 -
textExtent.cy / 2, character, character.GetLength
()); // Draw the auto bracket
205     }
206
207     // If, at any point, changed the colour
208     if (newColour) {
209         pDc->SetTextColor(RGB(0, 0, 0)); // Revert
colour back to black (original colour)
210     }
211

```

```

212         // If the character was not an auto bracket
213         if (!grey && character != L"\"" && character !=  ⤴
L"'"") {
214             lengthPrint += character.GetLength();
215
216             if (printing) {
217
218                 if (lengthPrint > printMaxCharLength) {
219
220
221                     pDc->TextOut(this->bounds.left + 2,  ⤴
this->bounds.top + this->bounds.Height() / 2 -  ⤴
textExtent.cy / 2, character, character.GetLength  ⤴
());
222
223                     /*pos += character.GetLength() + 2;
224
225                     Gdiplus::Graphics g(pDc->GetSafeHdc  ⤴
());
226                     Gdiplus::Rect expansionRect(this-  ⤴
>bounds.left + 2 + (textExtent.cx * pos), this-  ⤴
>bounds.top + this->bounds.Height() / 2 +  ⤴
returnNewLines * this->bounds.Height(),
227                     textExtent.cx, textExtent.cy);
228
229                     g.DrawImage(&newLineArrow,  ⤴
expansionRect);*/
230
231                     lengthPrint -=  ⤴
printMaxCharLength; // Since we are creating a new  ⤴
line, subtract the old 60 characters from this
232                     returnNewLines++;
233                     pos = 0;
234
235
236                 }
237                 else {
238
239                     pDc->TextOut(this->bounds.left + 2 +  ⤴
(textExtent.cx * pos), this->bounds.top + this-  ⤴
>bounds.Height() / 2 - textExtent.cy / 2 +  ⤴
returnNewLines * this->bounds.Height(), character,  ⤴
character.GetLength());
240                 }
241
242                 pos++;
243             }
244             else {
245
246                 pDc->TextOut(this->bounds.left + 2 +  ⤴
(textExtent.cx * (i + word.GetLength())), this-  ⤴
>bounds.top + this->bounds.Height() / 2 -  ⤴
textExtent.cy / 2, character, character.GetLength  ⤴
());
247             }
248         }

```

[illegible]

```

285
286                                     Gdiplus::Graphics g(pDc-
>GetSafeHdc());
287                                     Gdiplus::Rect expansionRect
(this->bounds.left + 2 + (textExtent.cx * pos),
this->bounds.top + this->bounds.Height() / 2 +
returnNewLines * this->bounds.Height(),
288                                     textExtent.cx,
textExtent.cy);
289
290                                     g.DrawImage(&newLineArrow,
expansionRect);*/
291
292                                     returnNewLines++;
293                                     pos = 0;
294                                 }
295                             }
296
297                                     // Print the 'scraps' of the word at the
end on the next line
298                                     pDc->TextOut(this->bounds.left + 2 +
(textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2 +
returnNewLines * this->bounds.Height(), word,
word.GetLength());
299
300                                     pos += word.GetLength();
301                                     found = TRUE;
302                                 }
303                                 j++;
304                             }
305
306                                     if (!found) {
307                                         word = this->text.Mid(i);
308                                         pDc->TextOut(this->bounds.left + 2 +
(textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2 +
returnNewLines * this->bounds.Height(), word,
word.GetLength());
309
310                                     }
311
312
313                                     pDc->SetTextColor(RGB(0, 0, 0)); // Revert
colour back to black (original colour)
314                                     i += word.GetLength() - 1;
315
316                                     word = L""; // Reset the word
317
318                                     pos = i + 1;
319
320                                 }
321
322                                     // If character is not a break away character, or we
are not at the end of the loop
323                                     else {

```



```
324         // Append the character to the word we are to look at
325
326         word += character;
327     }
328 }
329
330 // This means that there is a comment
331 else {
332
333     // Set colour of the comment
334     pDc->SetTextColor(theApp.commentColour);
335
336     // Add the rest of the line to the word for drawing
337     word.Append(this->text.Mid(i));
338
339     // If we are printing...
340     if (printing) {
341
342         // Add the length if the word to the length of our print
343         lengthPrint += word.GetLength();
344
345         while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines
346
347             CString tempWord = L"";
348             if ((i - word.GetLength()) % printMaxCharLength != 0) {
349
350                 tempWord = word.Mid(0, printMaxCharLength - pos); // Create a temporary word to store the start of the word that will go on the end of the current line
351                 word = word.Mid(printMaxCharLength - pos); // Change the word to be whatever we didn't print and continue the loop
352             }
353             else {
354                 pos++;
355             }
356
357             pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line
358
359             lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
360
361             /*pos += tempWord.GetLength() + 2;
362
```

```

363         Gdiplus::Graphics g(pDc->GetSafeHdc());
364         Gdiplus::Rect expansionRect(this-
>bounds.left + 2 + (textExtent.cx * pos), this-
>bounds.top + this->bounds.Height() / 2 +
returnNewLines * this->bounds.Height(),
        textExtent.cx, textExtent.cy);

365
366
367         g.DrawImage(&newLineArrow, expansionRect);*/
368
369         returnNewLines++;
370         pos = 0;
371     }
372     // Print the 'scraps' of the word at the end on
the next line
373     pDc->TextOut(this->bounds.left + 2 +
(textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2 +
returnNewLines * this->bounds.Height(), word,
word.GetLength());

374
375     }
376     // Not printing...
377     else {
378         pDc->TextOut(this->bounds.left + 2 +
(textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2, word,
word.GetLength());

379     }
380
381     // Revert to base colour
382     pDc->SetTextColor(0, 0, 0);
383
384     // comment has been found, so break from loop
385     comment = TRUE;
386     }
387     i++;
388 }
389 }
390 // If we are not using SmartColour
391 else {
392
393     if (printing) {
394         // Add the length of the word to the length of our print
395         CString word = this->text;
396         lengthPrint += word.GetLength();
397
398         while (lengthPrint > printMaxCharLength) { // As a word
could technically have a length greater than 60, we
must use a while to check for lines

399
400         CString tempWord = word.Mid(0,
printMaxCharLength); // Create a temporary word to
store the start of the word that will go on the end
of the current line
401         pDc->TextOut(this->bounds.left + 2, this->bounds.top
+ this->bounds.Height() / 2 - textExtent.cy / 2 +

```

```

        returnNewLines * this->bounds.Height(), tempWord,
        tempWord.GetLength()); // Print what we can to fill
        the line


402
403         /*int pos = tempWord.GetLength() + 2;
404
405         Gdiplus::Graphics g(pDc->GetSafeHdc());
406         Gdiplus::Rect expansionRect(this->bounds.left + 2 +
        (textExtent.cx * pos), this->bounds.top + this-
        >bounds.Height() / 2 + returnNewLines * this-
        >bounds.Height(),
407             textExtent.cx, textExtent.cy);
408
409         g.DrawImage(&newLineArrow, expansionRect);*/
410
411         word = word.Mid(printMaxCharLength); // Change the
        word to be whatever we didn't print and continue
        the loop
412         lengthPrint -= printMaxCharLength; // Since we are
        creating a new line, subtract the old 60 characters
        from this
413         returnNewLines++;
414     }
415     // Print the 'scraps' of the word at the end on the next
        line
416     pDc->TextOut(this->bounds.left + 2, this->bounds.top +
        this->bounds.Height() / 2 - textExtent.cy / 2 +
        returnNewLines * this->bounds.Height(), word,
        word.GetLength());
417 }
418 else {
419     pDc->TextOut(this->bounds.left + 2, this->bounds.top +
        this->bounds.Height() / 2 - textExtent.cy / 2, this-
        >text, this->text.GetLength());
420 }
421 }
422
423 if (this->active && !this->highlight && !printing) {
424
425     pDc->SelectStockObject(HOLLOW_BRUSH);
426     pDc->Rectangle(bounds);
427 }
428
429 if (this->highlight && !printing) {
430
431     this->highlighter.X = hStart * textExtent.cx + this-
        >bounds.left + 2;
432     this->highlighter.Width = (hEnd * textExtent.cx + this-
        >bounds.left + 2) - this->highlighter.X;
433
434     Gdiplus::Graphics g(pDc->GetSafeHdc());
435
436     Gdiplus::SolidBrush solidBrush(Gdiplus::Color(100, GetRValue
        (theApp.highlightColour), GetGValue
        (theApp.highlightColour), GetBValue
        (theApp.highlightColour)));

```

```
437         g.FillRectangle(&solidBrush, this->highlighter);
438     }
439     return returnNewLines;
440 }
441
442 // Public getters & setters
443 CString CTextLineObject::getText()
444 {
445     return this->text;
446 }
447 int CTextLineObject::getLength()
448 {
449     return this->text.GetLength();
450 }
451 void CTextLineObject::setText(CString text)
452 {
453     this->text = text;
454 }
455 void CTextLineObject::concatenateString(CString text, int position)
456 {
457     if (!GetKeyState(VK_INSERT)) {
458         this->text = this->text.Mid(0, position) + text + this->
459             >text.Mid(position, this->text.GetLength());
460     }
461     else {
462         this->text = this->text.Mid(0, position) + text + this->
463             >text.Mid(position + 1, this->text.GetLength());
464     }
465 }
466 void CTextLineObject::backspace(int position)
467 {
468     if (!this->highlight) {
469         this->text = this->text.Mid(0, position - 1) + this->
470             >text.Mid(position, this->text.GetLength());
471     }
472     else {
473         this->text = this->text.Mid(0, min(this->hStart, this->
474             >hEnd)) + this->text.Mid(max(this->hStart, this->hEnd),
475             this->text.GetLength());
476         this->setHighlighting(FALSE);
477     }
478 }
479 void CTextLineObject::isHighlighting()
480 {
481     return this->highlight;
482 }
483 void CTextLineObject::setHighlighting(BOOL val)
484 {
485     if (!val) {
486         this->highlighter.X = 0;
487         this->highlighter.Width = 0;
488         this->lineHighlight = FALSE;
489     }
490     this->highlight = val;
491 }
```

```
488 void CTextLineObject::setHighlighter(int pos1, int pos2)
489 {
490     if (pos1 > this->text.GetLength()) {
491         pos1 = this->text.GetLength() + 1;
492     }
493     if (pos2 > this->text.GetLength()) {
494         pos2 = this->text.GetLength() + 1;
495     }
496
497     if ((pos1 == 0 && pos2 == this->text.GetLength() + 1) || (pos2 == 0 && pos1 == this->text.GetLength() + 1)) {
498         this->lineHighlight = TRUE;
499     }
500
501     this->hStart = min(pos1, pos2);
502     this->hEnd = max(pos1, pos2);
503
504     if (pos1 != pos2) {
505         this->highlight = TRUE;
506     }
507 }
508 void CTextLineObject::highlightLine()
509 {
510     hStart = 0;
511     hEnd = this->text.GetLength()+1;
512     this->highlight = TRUE;
513     this->lineHighlight = TRUE;
514 }
515
516 BOOL CTextLineObject::isLineHighlighted()
517 {
518     return this->lineHighlight;
519 }
520
521 int CTextLineObject::getHStart()
522 {
523     return this->hStart;
524 }
525
526 int CTextLineObject::getHEnd()
527 {
528     return this->hEnd;
529 }
530
531 CString CTextLineObject::getHighlightedText()
532 {
533     if (this->highlight) {
534         return this->text.Mid(this->hStart, this->hEnd - this->hStart);
535     }
536     return CString();
537 }
538
539 int CTextLineObject::getNumSubLines()
540 {
541     return this->numSubLines;
```

```
542 }
543 std::vector<int> CTextLineObject::iGetLineNums()
544 {
545     return this->lineNums;
546 }
547 CString CTextLineObject::sGetLineNums()
548 {
549     CString num;
550     for (auto& it : this->lineNums) {
551
552         CString concat;
553         concat.Format(L"%d", it);
554         num.Append(concat);
555         num.Append(L".");
556     }
557     return num.Mid(0, num.GetLength() - 1);
558 }
559 void CTextLineObject::addSublines(std::vector<int> subs)
560 {
561     this->numSubLines += subs.size();
562     for (auto& it : subs) {
563         this->lineNums.push_back(it);
564     }
565 }
566
567 void CTextLineObject::incrementLine(int subline, int val)
568 {
569     if (subline > this->lineNums.size()) {
570         throw("ERROR::subline::OUT OF RANGE");
571     }
572     this->lineNums[subline] += val;
573 }
574
575 void CTextLineObject::setBounds(CRect bounds)
576 {
577     CAppObject::setBounds(bounds);
578
579     this->highlighter.X = bounds.left;
580     this->highlighter.Y = bounds.top;
581     this->highlighter.Height = bounds.Height();
582 }
583
584 void CTextLineObject::move(int x, int y)
585 {
586     CAppObject::move(x, y);
587     this->highlighter.Offset(x, y);
588 }
589
590 BOOL CTextLineObject::onlyNums(CString str)
591 {
592     for (int i = 0; i < str.GetLength(); i++) {
593
594         if (str.Mid(i, 1) != L"0" &&
595             str.Mid(i, 1) != L"1" &&
596             str.Mid(i, 1) != L"2" &&
597             str.Mid(i, 1) != L"3" &&
```

```
598         str.Mid(i, 1) != L"4" &&
599         str.Mid(i, 1) != L"5" &&
600         str.Mid(i, 1) != L"6" &&
601         str.Mid(i, 1) != L"7" &&
602         str.Mid(i, 1) != L"8" &&
603         str.Mid(i, 1) != L"9" ) {
604             return FALSE;
605         }
606     }
607
608     return TRUE;
609 }
610 BOOL CTextLineObject::brkPosContains(std::vector<std::vector<int>>> 
    vector, int value, int side)
611 {
612     for (int i = 0; i < vector.size(); i++) {
613
614         if (vector[i][side] == value) {
615             return TRUE;
616         }
617     }
618
619     return FALSE;
620 }
621
```