```cpp

// CTextDocument.cpp : implementation of the CTextDocument 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 "CTextDocument.h"
#include "ChildFrm.h"
#include "CTextDocView.h"

#include <propkey.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CTextDocument

IMPLEMENT_DYNCREATE(CTextDocument, CDocument)

BEGIN_MESSAGE_MAP(CTextDocument, CDocument)
    ON_COMMAND(ID_FILE_SAVE, &CTextDocument::OnFileSave)
    ON_COMMAND(ID_FILE_OPEN, &CTextDocument::OnFileOpen)
END_MESSAGE_MAP()

// Public constructors
CTextDocument::CTextDocument() noexcept
{
    this->SetTitle(L"Pseudocode Editor");
    std::vector<CString> text = { L"" };

    this->defBoxHeight = 20;
    this->lineOffset = 100;
    this->filename = L"local";

    CRect editorBounds;
    editorBounds.top = 0;
    editorBounds.bottom = this->defBoxHeight*theApp.zoom * (text.size());
    editorBounds.left = 0;
    editorBounds.right = 1000;

    std::vector <int> lineNums = { 1 };

    this->objects.Add(new CTextEditorObject(editorBounds, L"1", TRUE, 0, this->defBoxHeight*theApp.zoom, TRUE, text, lineNums, this->lineOffset));
}
```

```
52  CTextDocument::~CTextDocument()
53  {
54      for (int i = 0; i < this->objects.GetSize(); i++) {
55          delete this->objects[i];
56      }
57  }
58
59  // Public overrides
60  BOOL CTextDocument::OnNewDocument()
61  {
62      if (!CDocument::OnNewDocument())
63          return FALSE;
64
65      // TODO: add reinitialization code here
66      // (SDI documents will reuse this document)
67
68      return TRUE;
69  }
70  void CTextDocument::Serialize(CArchive& ar)
71  {
72      if (ar.IsStoring())
73      {
74
75      }
76      else
77      {
78
79      }
80  }
81
82  void CTextDocument::OnFileSave()
83  {
84      const TCHAR szFilter[] = _T("Text Files(*.txt) | *.txt | All
        File | *.* | ");
85
86      CFileDialog fileDialog(FALSE, _T("txt"), NULL, OFN_HIDEREADONLY
        | OFN_FILEMUSTEXIST, szFilter);
87
88      if (fileDialog.DoModal() == IDOK) {
89
90          CFile file;
91
92          if (file.Open(fileDialog.GetFolderPath() + L"\\" +
              fileDialog.GetFileName(), CFile::modeCreate |
              CFile::modeWrite)) {
93
94              CString data = L"";
95
96              int max_size = this->objects[0]->sGetLineNum(this-
                >objects[0]->getNumLines()).GetLength();
97
98              for (int i = 1; i < this->objects.GetSize(); i++) {
99                  if (this->objects[i]->sGetLineNum(1).GetLength() >
                     max_size) {
100                     max_size = this->objects[i]->sGetLineNum(this-
                     >objects[i]->getNumLines()).GetLength();
```

```
101                     }
102                 }
103
104                 for (int i = 0; i < this->objects.GetSize(); i++) {
105                     for (int j = 1; j < this->objects[i]->getNumLines()
                         + 1; j++) {
106
107                         for (int k = 0; k < (max_size - this->objects
                        [i]->sGetLineNum(j).GetLength()); k++) {
108                             data.Append(L" ");
109                         }
110
111                         data.Append(this->objects[i]->sGetLineNum(j));
112                         data.Append(L"    ");
113                         data.Append(this->objects[i]->getLineText(j));
114                         data.Append(L"\n");
115                     }
116                     data.Append(L"\n\n");
117                 }
118
119                 file.Write(data.GetBuffer(), data.GetLength()*2);
120
121                 this->filename = fileDialog.GetFileName();
122             }
123         }
124 }
125 void CTextDocument::OnFileOpen()
126 {
127     const TCHAR szFilter[] = _T("Text Files (*.txt)|*.txt| All File|
        *.*|"); // Set a filter for file types
128
129     CFileDialog fileDialog(TRUE, _T("*.txt"), NULL, OFN_HIDEREADONLY
           | OFN_FILEMUSTEXIST, szFilter); // Instantiate file dialog
        box
130
131     if (fileDialog.DoModal() == IDOK) { // Open the file dialog and
        wait for the user to be done
132
133         CString filePath = fileDialog.GetFolderPath() + L"\\" +
            fileDialog.GetFileName(); // Create the file path
134         CDocument* newDoc = theApp.OpenDocumentFile(filePath); //
            Open a new document for the file
135
136         CFile file; // Open the file pathway
137
138         if (file.Open(filePath, CFile::modeRead)) { // Read through
            the file
139
140             // Read the text from the file
141
142             int iFileSiz = file.GetLength();  // Getting the content
                length
143             BYTE* pData = new BYTE[iFileSiz]; // To save the data in
144
145             file.Read(pData, iFileSiz);       //  Reading file
                content
```

```
146
147             pData[iFileSiz] = '\0';           //  Add last character as NULL
148
149             file.Close(); // Close the file
150
151             CString character;
152             CString data;
153
154             int k = 0;
155
156             for (int i = 0; i < iFileSiz; i++) { // Iterate through all the readable characters
157
158                 character.Format(L"%C", pData[i]); // Format each character correctlyS
159                 data += character; // Add the character to the current line (data)
160
161                 if (character == L"\n" || i == iFileSiz-1) { // Once we have recieved a new space or have gotten to the end of the file
162
163                     int j = 0;
164                     BOOL end = FALSE; // Iterators
165
166                     int space = 0;
167                     BOOL startNum = FALSE;
168
169                     CString subnum = L"";
170                     std::vector<int>writingLineNum = {};
171
172                     while (j < data.GetLength() && !end) { // Iterate through the collected line
173
174                         CString active = data.Mid(j, 1); // Is the secondary iterated character
175
176                         if (active == L"0" ||
177                             active == L"1" ||
178                             active == L"2" ||
179                             active == L"3" ||
180                             active == L"4" ||
181                             active == L"5" ||
182                             active == L"6" ||
183                             active == L"7" ||
184                             active == L"8" ||
185                             active == L"9") { // Once we have found a number ie, where the line number is in the data
186
187                             j++;
188                             startNum = TRUE; // This tells us that we have found start of the line number
189
190                             subnum.Append(active);
191                         }
```

```
192                     else if (active == L"." && space == 0) { // Once we have reached the next subline
193                         j++;
194                         writingLineNum.push_back(_ttoi(subnum));
195                         subnum = L"";
196                     }
197                     else if (active == L" " && space < 4/*There should be 4 spaces between the end of the line number and the start of the text*/) { // If we have not reached the start of the actual text, either before or after the subline
198
199                         j++;
200
201                         if (startNum) { // If we have found the start of the line number we can start incrementing space
202
203                             space++;
204                         }
205                     }
206                     else { // This means that we have gotten to the start of the text
207
208                         if (subnum != L"") {
209                             writingLineNum.push_back(_ttoi(subnum));
210                         }
211                         subnum = L"";
212                         end = TRUE;
213                     }
214                 }
215
216
217                 BOOL newEdit = FALSE;
218
219                 if (writingLineNum.size() > 0) { // If there is not a gap in the line numbers
220
221                     std::vector<int>writingTemp = writingLineNum;
222                     writingTemp.pop_back(); // Remove the last subline in order to search for a parent line
223
224                     std::vector<int>itTemp;
225                     int i = ((CTextDocument*)newDoc)->objects.GetSize() - 1; // Find the number of objects created so far
226                     BOOL fin = FALSE; // Iterators
227
228                     while (i >= 0 && !fin) { // Iterate through all the objects in the new doc, in reverse
229
230                         itTemp = ((CTextDocument*)newDoc)->objects[i]->iGetLineNum(1);
```

```
231                         itTemp.pop_back(); // Get the line number and remove the recent subline
232
233                         if (writingTemp == itTemp) { // Check if this is the editor we are searching
234
235                             fin = TRUE; // If it is, say we found it with this
236                         }
237
238                         i--;
239                     }
240
241                     if (!fin) {
242                         newEdit = TRUE; // If we did not find an editor that our line belongs to, create a new one
243                     }
244
245                     if (newEdit) {
246
247                         // If we get here, it means that the new line should be made in a new editor
248
249                         CRect prevBounds = ((CTextDocument*)newDoc)->objects[k]->getBounds(); // Find the bounds of the editor that lies above the one we are about to create
250
251                         CRect bounds = prevBounds; // Create bounds for the new object
252                         bounds.top = prevBounds.bottom + ((CTextDocument*)newDoc)->objects[k]->getBoxHeight(); // Move the top
253                         bounds.bottom = bounds.top + ((CTextDocument*)newDoc)->objects[k]->getBoxHeight(); // Move the bottom
254
255                         CString id;
256                         id.Format(L"%d", k + 1);
257
258                         ((CTextDocument*)newDoc)->objects[k]->OnRecieveBackspace();
259
260                         ((CTextDocument*)newDoc)->objects.Add(
261                             new CTextEditorObject(bounds, id, TRUE, 0, ((CTextDocument*)newDoc)->defBoxHeight, FALSE,
262                                 std::vector<CString>{data.Mid(j)}, writingLineNum,
263                                 this->lineOffset)
264                         );
265
266                         ((CTextDocument*)newDoc)->objects[k + 1]->OnRecieveReturn(TRUE);
267                         ((CTextDocument*)newDoc)->objects[k]->setActive(FALSE);
```

```
268                                 k++;
269                                 ///((CTextDocument*)newDoc)->objects[k]->OnRecieveReturn(TRUE);
270                             }
271                             else {
272                                 ((CTextDocument*)newDoc)->objects[k]->OnRecieveText(data.Mid(j), TRUE); // Send the data through to the current editor
273                                 ///((CTextDocument*)newDoc)->objects[k]->OnRecieveReturn(TRUE);
274
275                             }
276                         }
277
278                         data = L"";
279                     }
280                 }
281
282                 ((CTextDocument*)newDoc)->objects[k]->OnRecieveBackspace();
283                 ((CTextDocument*)newDoc)->objects[k]->setActive(FALSE);
284                 ((CTextDocument*)newDoc)->objects[0]->setActive(TRUE);
285
286                 ((CTextDocView*)theApp.m_ActiveView)->refresh();
287
288                 ((CTextDocument*)newDoc)->filename = fileDialog.GetFileName();
289             }
290         }
291 }
292
293 #ifdef SHARED_HANDLERS
294
295 // Support for thumbnails
296 void CTextDocument::OnDrawThumbnail(CDC& dc, LPRECT lprcBounds)
297 {
298     // Modify this code to draw the document's data
299     dc.FillSolidRect(lprcBounds, RGB(0, 0, 255));
300
301     CString strText = _T("TODO: implement thumbnail drawing here");
302     LOGFONT lf;
303
304     CFont* pDefaultGUIFont = CFont::FromHandle((HFONT)GetStockObject(DEFAULT_GUI_FONT));
305     pDefaultGUIFont->GetLogFont(&lf);
306     lf.lfHeight = 36;
307
308     CFont fontDraw;
309     fontDraw.CreateFontIndirect(&lf);
310
311     CFont* pOldFont = dc.SelectObject(&fontDraw);
312     dc.DrawText(strText, lprcBounds, DT_CENTER | DT_WORDBREAK);
313     dc.SelectObject(pOldFont);
314 }
315
316 // Support for Search Handlers
```

```
317 void CTextDocument::InitializeSearchContent()
318 {
319     CString strSearchContent;
320     // Set search contents from document's data.
321     // The content parts should be separated by ";"
322
323     // For example:  strSearchContent = _T
        ("point;rectangle;circle;ole object;");
324     SetSearchContent(strSearchContent);
325 }
326
327 void CTextDocument::SetSearchContent(const CString& value)
328 {
329     if (value.IsEmpty())
330     {
331         RemoveChunk(PKEY_Search_Contents.fmtid,
            PKEY_Search_Contents.pid);
332     }
333     else
334     {
335         CMFCFilterChunkValueImpl *pChunk = nullptr;
336         ATLTRY(pChunk = new CMFCFilterChunkValueImpl);
337         if (pChunk != nullptr)
338         {
339             pChunk->SetTextValue(PKEY_Search_Contents, value,
                CHUNK_TEXT);
340             SetChunkValue(pChunk);
341         }
342     }
343 }
344
345 #endif // SHARED_HANDLERS
346
347 // Public implementations
348 #ifdef _DEBUG
349 void CTextDocument::AssertValid() const
350 {
351     CDocument::AssertValid();
352 }
353
354 void CTextDocument::Dump(CDumpContext& dc) const
355 {
356     CDocument::Dump(dc);
357 }
358 #endif //_DEBUG
359
360
```