```
1
2  // DesignArk.h : main header file for the DesignArk application
3  //
4  #pragma once
5
6  #ifndef __AFXWIN_H__
7      #error "include 'pch.h' before including this file for PCH"
8  #endif
9
10 #include "resource.h"       // main symbols
11
12 /*///////
13
14 To find the number of lines in project, run
15
16 (gci -include *.h,*.cpp -recurse | select-string .).Count
17
18 in the code folder
19
20 (last count: 5298)
21
22 ///////*/
23
24
25 // CDesignArkApp:
26 // See DesignArk.cpp for the implementation of this class
27 //
28
29 class CDesignArkApp : public CWinAppEx
30 {
31 public:
32     // Public Constructors
33     CDesignArkApp() noexcept;
34     ~CDesignArkApp();
35
36     // Public Overrides
37     virtual BOOL InitInstance();
38     virtual int ExitInstance();
39
40     // Public Commands
41     afx_msg void OnSmartColour();
42
43     afx_msg void OnAppAbout();
44     afx_msg void OnOpenSite();
45     afx_msg void OnUserdocs();
46
47     // Public Implementations
48     UINT  m_nAppLook;
49     BOOL  m_bHiColorIcons;
50
51     CView* m_ActiveView;
52
53     // Public Resources
54     // SmartColour Preferences
55     std::vector<std::vector<CString>> smartColour_String;
56     std::vector<COLORREF> smartColour_Colour;
```

```
57
58      CString commentType;
59      COLORREF commentColour;
60      COLORREF numberColour;
61      COLORREF highlightColour;
62
63      // App work preferences
64      float zoom;
65      CString sFont;
66      int indentSize;
67
68  protected:
69      // Protected Implementation
70      virtual void PreLoadState();
71      virtual void LoadCustomState();
72      virtual void SaveCustomState();
73
74      DECLARE_MESSAGE_MAP()
75  };
76
77  extern CDesignArkApp theApp;
78
```

```cpp

// DesignArk.cpp : Defines the class behaviors for the application.
//

#include "pch.h"
#include "framework.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "DesignArk.h"
#include "MainFrm.h"

#include "ChildFrm.h"
#include "CTextDocument.h"
#include "CTextDocView.h"

#include "CAboutDlg.h"
#include "CSmartColour.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CDesignArkApp

BEGIN_MESSAGE_MAP(CDesignArkApp, CWinAppEx)
    ON_COMMAND(ID_VIEW_SMARTCOLOURWINDOW, &CDesignArkApp::OnSmartColour)
    ON_COMMAND(ID_HELP_OPENWEBSITE, &CDesignArkApp::OnOpenSite)
    ON_COMMAND(ID_HELP_VIEWUSERDOCSONLINE, &CDesignArkApp::OnUserdocs)
    ON_COMMAND(ID_APP_ABOUT, &CDesignArkApp::OnAppAbout)

    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, &CWinAppEx::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, &CWinAppEx::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, &CWinAppEx::OnFilePrintSetup)
END_MESSAGE_MAP()


// CDesignArkApp construction
CDesignArkApp::CDesignArkApp() noexcept
{
    m_bHiColorIcons = TRUE;
    m_nAppLook = 0;

    // support Restart Manager
    m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_ALL_ASPECTS;
#ifdef _MANAGED
    // If the application is built using Common Language Runtime support (/clr):
    //     1) This additional setting is needed for Restart Manager support to work properly.
    //     2) In your project, you must add a reference to
```

```
        System.Windows.Forms in order to build.
52      System::Windows::Forms::Application::SetUnhandledExceptionMode
        (System::Windows::Forms::UnhandledExceptionMode::ThrowExceptio
        n);
53  #endif
54
55      SetAppID(_T("DesignArk.Verison.1.0"));
56
57      // TODO: add construction code here,
58      // Place all significant initialization in InitInstance
59      this->smartColour_String.resize(5);
60      this->smartColour_Colour.resize(5);
61
62      this->smartColour_String[0] = std::vector<CString>{
63          L"ELIF",
64          L"ELSE",
65          L"FOR",
66          L"IF",
67          L"IN",
68          L"RETURN",
69          L"WHILE"
70      };
71      this->smartColour_Colour[0] = COLORREF(RGB(128, 0, 128));
72
73      this->smartColour_String[1] = std::vector<CString>{
74          L"AS",
75          L"CREATE",
76          L"DECLARE",
77          L"DISPLAY",
78          L"DO",
79          L"FUNCTION",
80          L"INITIALISE",
81          L"INITIALLY",
82          L"PRINT",
83          L"PROCEDURE",
84          L"SET",
85          L"TO",
86      };
87      this->smartColour_Colour[1] = COLORREF(RGB(128, 0, 32));
88
89      this->smartColour_String[2] = std::vector<CString>{
90          L"ARRAY",
91          L"BOOLEAN",
92          L"CLASS",
93          L"CONST",
94          L"DICTIONARY",
95          L"FLOAT",
96          L"INT",
97          L"LIST",
98          L"OVERLOAD"
99          L"REAL",
100         L"STRING",
101         L"VIRTUAL",
102         L"VOID"
103     };
104     this->smartColour_Colour[2] = COLORREF(RGB(0, 0, 255));
```

```
105
106     this->smartColour_String[3] = std::vector<CString>{
107         L"vector"
108     };
109     this->smartColour_Colour[3] = COLORREF(RGB(0, 255, 0));
110
111     this->smartColour_String[4] = std::vector<CString>{
112         L"RANDOM"
113     };
114     this->smartColour_Colour[4] = COLORREF(RGB(255, 0, 0));
115
116     this->commentType = L"//";
117     this->commentColour = COLORREF(RGB(255, 0, 0));
118
119     this->numberColour = COLORREF(RGB(0, 255, 0));
120     this->highlightColour = COLORREF(RGB(110, 192, 250));
121
122     this->zoom = 1.f;
123     this->sFont = L"Courier New";
124     this->indentSize = 4;
125
126     // Initilise the GDI stuff
127     Gdiplus::GdiplusStartupInput gdiplusStartupInput;
128     ULONG_PTR gdiplusToken;
129
130     Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput,
        NULL);
131 }
132 CDesignArkApp::~CDesignArkApp()
133 {
134 }
135
136 // The one and only CDesignArkApp object
137 CDesignArkApp theApp;
138
139 // CDesignArkApp initialisation
140 BOOL CDesignArkApp::InitInstance()
141 {
142     /*////////////////////////////////////////
143     AllocConsole();
144     FILE* fDummy;
145     freopen_s(&fDummy, "CONOUT$", "w", stdout);
146     freopen_s(&fDummy, "CONOUT$", "w", stderr);
147     freopen_s(&fDummy, "CONIN$", "r", stdin); // Console can be used
        be debugging
148     ////////////////////////////////////////*/
149
150     // Initialise the common controls
151     INITCOMMONCONTROLSEX InitCtrls;
152     InitCtrls.dwSize = sizeof(InitCtrls);
153     InitCtrls.dwICC = ICC_WIN95_CLASSES;
154     InitCommonControlsEx(&InitCtrls);
155
156     CWinAppEx::InitInstance();
157
158     // Initialize OLE libraries
```

```
159     if (!AfxOleInit())
160     {
161         AfxMessageBox(IDP_OLE_INIT_FAILED);
162         return FALSE;
163     }
164
165     AfxEnableControlContainer();
166     EnableTaskbarInteraction();
167
168     AfxInitRichEdit2(); // is required to use RichEdit control
169
170     // Register app with OS and load INI file options (including
        MRU)
171     SetRegistryKey(_T("DesignArk"));
172     LoadStdProfileSettings(4);  // Load standard INI file options
        (including MRU)
173
174     // Initialise managers
175     InitContextMenuManager();
176     InitKeyboardManager();
177     InitTooltipManager();
178     CMFCToolTipInfo ttParams;
179     ttParams.m_bVislManagerTheme = TRUE;
180     theApp.GetTooltipManager()->SetTooltipParams
        (AFX_TOOLTIP_TYPE_ALL,
181         RUNTIME_CLASS(CMFCToolTipCtrl), &ttParams);
182
183     // Initialise document template
184     CMultiDocTemplate* pDocTemplate;
185     pDocTemplate = new CMultiDocTemplate(IDR_DesignArkTYPE,
186         RUNTIME_CLASS(CTextDocument),
187         RUNTIME_CLASS(CChildFrame), // custom MDI child frame
188         RUNTIME_CLASS(CTextDocView));
189     if (!pDocTemplate)
190         return FALSE;
191     this->AddDocTemplate(pDocTemplate);
192
193     // create main MDI Frame window
194     CMainFrame* pMainFrame = new CMainFrame;
195     if (!pMainFrame || !pMainFrame->LoadFrame(IDR_MAINFRAME))
196     {
197         delete pMainFrame;
198         return FALSE;
199     }
200     m_pMainWnd = pMainFrame;
201
202     // Parse command line for standard shell commands, DDE, file
        open
203     CCommandLineInfo cmdInfo;
204     ParseCommandLine(cmdInfo);
205
206     // Dispatch commands specified on the command line.  Will return
         FALSE if
207     // app was launched with /RegServer, /Register, /Unregserver
        or /Unregister.
208     if (!ProcessShellCommand(cmdInfo))
```

```
209         return FALSE;
210
211     // The main window has been initialized, so show and update it
212
213     //CRect client;
214     //pMainFrame->GetWindowRect(client);
215     //pMainFrame->MoveWindow(client.left, client.top, 750, 900);
216
217     pMainFrame->ShowWindow(m_nCmdShow);
218     pMainFrame->UpdateWindow();
219
220     return TRUE;
221 }
222 int CDesignArkApp::ExitInstance()
223 {
224     //TODO: handle additional resources you may have added
225     AfxOleTerm(FALSE);
226
227     return CWinAppEx::ExitInstance();
228 }
229
230 // Public Commands
231 void CDesignArkApp::OnSmartColour()
232 {
233     CString tempIndentSize;
234     tempIndentSize.Format(L"%d", this->indentSize);
235     CSmartColour smartColour(this->smartColour_String, this->
        >smartColour_Colour, this->commentType, this->commentColour,
        this->numberColour, this->highlightColour, this->sFont,
        tempIndentSize);
236
237     if (smartColour.DoModal() == IDOK) {
238
239         this->smartColour_String[0] = ((CSmartColourStatements*)
            smartColour.GetPage(0))->m_Strings_Statements_syntax_var;
240         this->smartColour_Colour[0] = ((CSmartColourStatements*)
            smartColour.GetPage(0))->m_Colour_Statements_syntax_var;
241
242         this->smartColour_String[1] = ((CSmartColourStatements*)
            smartColour.GetPage(0))-
            >m_Strings_Statements_statements_var;
243         this->smartColour_Colour[1] = ((CSmartColourStatements*)
            smartColour.GetPage(0))-
            >m_Colour_Statements_statements_var;
244
245         this->smartColour_String[2] = ((CSmartColourTypes*)
            smartColour.GetPage(1))->m_listbox_types_strings;
246         this->smartColour_Colour[2] = ((CSmartColourTypes*)
            smartColour.GetPage(1))->m_Combo_Types_colour_var;
247
248         this->smartColour_String[3] = ((CSmartColourBuiltin*)
            smartColour.GetPage(2))->m_Listbox_Builtin_Classes_var;
249         this->smartColour_Colour[3] = ((CSmartColourBuiltin*)
            smartColour.GetPage(2))-
            >m_Combo_Builtin_Classes_colour_var;
250
```

```
251         this->smartColour_String[4] = ((CSmartColourBuiltin*)
              smartColour.GetPage(2))->m_Listbox_Builtin_Functions_var;
252         this->smartColour_Colour[4] = ((CSmartColourBuiltin*)
              smartColour.GetPage(2))-
              >m_Combo_Builtin_Functions_colour_var;
253
254         this->commentType = ((CSmartColourOther*)smartColour.GetPage
              (3))->m_pCombo_comments_type_var;
255         this->commentColour = ((CSmartColourOther*)
              smartColour.GetPage(3))->m_pCombo_comments_colour_var;
256
257         this->numberColour = ((CSmartColourOther*)
              smartColour.GetPage(3))->m_pCombo_numbers_colour_var;
258
259         this->highlightColour = ((CSmartColourOther*)
              smartColour.GetPage(3))->m_pCombo_highlight_colour_var;
260
261         this->sFont = ((CSmartColourOther*)smartColour.GetPage(3))-
              >m_pCombo_fonts_var;
262
263         this->indentSize = _wtoi(((CSmartColourOther*)
              smartColour.GetPage(3))->m_pCombo_indent_var);
264
265         // Will need to fixed if more views/docs are implemeted but
              will do for v1.0
266         if (this->m_ActiveView) {
267             ((CTextDocView*)this->m_ActiveView)->refresh();
268         }
269     }
270 }
271
272 void CDesignArkApp::OnAppAbout() // This bit is not working
273 {
274     CAboutDlg aboutDlg;
275     aboutDlg.DoModal(); // This line is triggering a debug assertion
          failure
276 }
277 void CDesignArkApp::OnOpenSite()
278 {
279     ShellExecute(0, 0, L"http://localhost/DesignArk/", 0, 0,
        SW_SHOW);
280 }
281 void CDesignArkApp::OnUserdocs()
282 {
283     ShellExecute(0, L"open", L"http://localhost/DesignArk/
        userdocs.php", 0, 0, SW_SHOW);
284 }
285
286 // Protected Implementation
287 void CDesignArkApp::PreLoadState()
288 {
289     BOOL bNameValid;
290     CString strName;
291     bNameValid = strName.LoadString(IDS_EDIT_MENU);
292     ASSERT(bNameValid);
293     GetContextMenuManager()->AddMenu(strName, IDR_POPUP_EDIT);
```

```
294     bNameValid = strName.LoadString(IDS_EXPLORER);
295     ASSERT(bNameValid);
296     GetContextMenuManager()->AddMenu(strName, IDR_POPUP_EXPLORER);
297 }
298 void CDesignArkApp::LoadCustomState()
299 {
300 }
301 void CDesignArkApp::SaveCustomState()
302 {
303 }
304
305
```

```
1
2  // MainFrm.h : interface of the CMainFrame class
3  //
4
5  #pragma once
6
7  class CMainFrame : public CMDIFrameWndEx
8  {
9  public:
10     // Public Constructors
11     CMainFrame() noexcept;
12     virtual ~CMainFrame();
13
14     //Public Overrides
15     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
16     virtual BOOL LoadFrame(UINT nIDResource, DWORD dwDefaultStyle =
         WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, CWnd* pParentWnd =
         nullptr, CCreateContext* pContext = nullptr);
17
18     // Public Implementations
19     DECLARE_DYNAMIC(CMainFrame)
20 #ifdef _DEBUG
21     virtual void AssertValid() const;
22     virtual void Dump(CDumpContext& dc) const;
23 #endif
24
25 protected:
26     // Protected Implementations
27     DECLARE_MESSAGE_MAP()
28
29     // Protected Resources
30     CMFCMenuBar       m_wndMenuBar;
31     CMFCToolBar       m_wndToolBar;
32     CMFCStatusBar     m_wndStatusBar;
33     CMFCToolBarImages m_UserImages;
34
35     // Protected Message Handlers
36     afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
37     afx_msg void OnWindowManager();
38     afx_msg void OnViewCustomize();
39     afx_msg LRESULT OnToolbarCreateNew(WPARAM wp, LPARAM lp);
40     afx_msg void OnApplicationLook(UINT id);
41     afx_msg void OnUpdateApplicationLook(CCmdUI* pCmdUI);
42     virtual void OnSetActiveView(CView* pViewNew, BOOL bNotify =
         TRUE);
43
44 private:
45     // Private Initialisers
46     int init_tabs();
47     int init_menubar();
48     int init_toolbar();
49     int init_statusbar();
50     int init_dockingwindows();
51     int init_customisation();
52     int init_userimages();
53     int init_commands();
```

```
54  };
55
```

```cpp

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "pch.h"
#include "framework.h"
#include "DesignArk.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWndEx)

const int  iMaxUserToolbars = 10;
const UINT uiFirstUserToolBarId = AFX_IDW_CONTROLBAR_FIRST + 40;
const UINT uiLastUserToolBarId = uiFirstUserToolBarId +
  iMaxUserToolbars - 1;

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWndEx)
    ON_WM_CREATE()
    ON_COMMAND(ID_WINDOW_MANAGER, &CMainFrame::OnWindowManager)
    ON_COMMAND(ID_VIEW_CUSTOMIZE, &CMainFrame::OnViewCustomize)
    ON_REGISTERED_MESSAGE(AFX_WM_CREATETOOLBAR,
      &CMainFrame::OnToolbarCreateNew)
    ON_COMMAND_RANGE(ID_VIEW_APPLOOK_WIN_2000,
      ID_VIEW_APPLOOK_WINDOWS_7, &CMainFrame::OnApplicationLook)
    ON_UPDATE_COMMAND_UI_RANGE(ID_VIEW_APPLOOK_WIN_2000,
      ID_VIEW_APPLOOK_WINDOWS_7,
      &CMainFrame::OnUpdateApplicationLook)
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM
};

// CMainFrame construction/destruction

CMainFrame::CMainFrame() noexcept
{
    // TODO: add member initialization code here
    theApp.m_nAppLook = theApp.GetInt(_T("ApplicationLook"),
      ID_VIEW_APPLOOK_VS_2008);
}
CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
```

```
51  {
52      if( !CMDIFrameWndEx::PreCreateWindow(cs) )
53          return FALSE;
54      // TODO: Modify the Window class or styles here by modifying
55      //  the CREATESTRUCT cs
56
57      return TRUE;
58  }
59  BOOL CMainFrame::LoadFrame(UINT nIDResource, DWORD dwDefaultStyle,
      CWnd* pParentWnd, CCreateContext* pContext)
60  {
61      // base class does the real work
62
63      if (!CMDIFrameWndEx::LoadFrame(nIDResource, dwDefaultStyle,
          pParentWnd, pContext))
64      {
65          return FALSE;
66      }
67
68
69      // enable customization button for all user toolbars
70      BOOL bNameValid;
71      CString strCustomize;
72      bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
73      ASSERT(bNameValid);
74
75      for (int i = 0; i < iMaxUserToolbars; i ++)
76      {
77          CMFCToolBar* pUserToolbar = GetUserToolBarByIndex(i);
78          if (pUserToolbar != nullptr)
79          {
80              pUserToolbar->EnableCustomizeButton(TRUE,
                  ID_VIEW_CUSTOMIZE, strCustomize);
81          }
82      }
83
84      return TRUE;
85  }
86
87  #ifdef _DEBUG
88  void CMainFrame::AssertValid() const
89  {
90      CMDIFrameWndEx::AssertValid();
91  }
92  void CMainFrame::Dump(CDumpContext& dc) const
93  {
94      CMDIFrameWndEx::Dump(dc);
95  }
96  #endif //_DEBUG
97
98  int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
99  {
100     if (CMDIFrameWndEx::OnCreate(lpCreateStruct) == -1)
101         return -1;
102
103     BOOL bNameValid;
```

```
104     CString strCustomize;
105
106     init_tabs();
107     init_menubar();
108     init_toolbar();
109     init_statusbar();
110     init_dockingwindows();
111     init_customisation();
112     init_userimages();
113     init_commands();
114
115     return 0;
116 }
117 void CMainFrame::OnWindowManager()
118 {
119     ShowWindowsDialog();
120 }
121 void CMainFrame::OnViewCustomize()
122 {
123     CMFCToolBarsCustomizeDialog* pDlgCust = new
        CMFCToolBarsCustomizeDialog(this, TRUE /* scan menus */);
124     pDlgCust->EnableUserDefinedToolbars();
125     pDlgCust->Create();
126 }
127 LRESULT CMainFrame::OnToolbarCreateNew(WPARAM wp,LPARAM lp)
128 {
129     LRESULT lres = CMDIFrameWndEx::OnToolbarCreateNew(wp,lp);
130     if (lres == 0)
131     {
132         return 0;
133     }
134
135     CMFCToolBar* pUserToolbar = (CMFCToolBar*)lres;
136     ASSERT_VALID(pUserToolbar);
137
138     BOOL bNameValid;
139     CString strCustomize;
140     bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
141     ASSERT(bNameValid);
142
143     pUserToolbar->EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE,
        strCustomize);
144     return lres;
145 }
146 void CMainFrame::OnApplicationLook(UINT id)
147 {
148     CWaitCursor wait;
149
150     theApp.m_nAppLook = id;
151
152     switch (theApp.m_nAppLook)
153     {
154     case ID_VIEW_APPLOOK_WIN_2000:
155         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
            (CMFCVisualManager));
156         break;
```

```
157
158     case ID_VIEW_APPLOOK_OFF_XP:
159         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerOfficeXP));
160         break;
161
162     case ID_VIEW_APPLOOK_WIN_XP:
163         CMFCVisualManagerWindows::m_b3DTabsXPTheme = TRUE;
164         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerWindows));
165         break;
166
167     case ID_VIEW_APPLOOK_OFF_2003:
168         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerOffice2003));
169         CDockingManager::SetDockingMode(DT_SMART);
170         break;
171
172     case ID_VIEW_APPLOOK_VS_2005:
173         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerVS2005));
174         CDockingManager::SetDockingMode(DT_SMART);
175         break;
176
177     case ID_VIEW_APPLOOK_VS_2008:
178         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerVS2008));
179         CDockingManager::SetDockingMode(DT_SMART);
180         break;
181
182     case ID_VIEW_APPLOOK_WINDOWS_7:
183         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerWindows7));
184         CDockingManager::SetDockingMode(DT_SMART);
185         break;
186
187     default:
188         switch (theApp.m_nAppLook)
189         {
190         case ID_VIEW_APPLOOK_OFF_2007_BLUE:
191             CMFCVisualManagerOffice2007::SetStyle
                  (CMFCVisualManagerOffice2007::Office2007_LunaBlue);
192             break;
193
194         case ID_VIEW_APPLOOK_OFF_2007_BLACK:
195             CMFCVisualManagerOffice2007::SetStyle
                  (CMFCVisualManagerOffice2007::Office2007_ObsidianBlack
                  );
196             break;
197
198         case ID_VIEW_APPLOOK_OFF_2007_SILVER:
199             CMFCVisualManagerOffice2007::SetStyle
                  (CMFCVisualManagerOffice2007::Office2007_Silver);
200             break;
201
202         case ID_VIEW_APPLOOK_OFF_2007_AQUA:
```

```
203             CMFCVisualManagerOffice2007::SetStyle
                  (CMFCVisualManagerOffice2007::Office2007_Aqua);
204             break;
205         }
206
207         CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS
              (CMFCVisualManagerOffice2007));
208         CDockingManager::SetDockingMode(DT_SMART);
209     }
210
211     RedrawWindow(nullptr, nullptr, RDW_ALLCHILDREN | RDW_INVALIDATE
          | RDW_UPDATENOW | RDW_FRAME | RDW_ERASE);
212
213     theApp.WriteInt(_T("ApplicationLook"), theApp.m_nAppLook);
214 }
215 void CMainFrame::OnUpdateApplicationLook(CCmdUI* pCmdUI)
216 {
217     pCmdUI->SetRadio(theApp.m_nAppLook == pCmdUI->m_nID);
218 }
219
220 void CMainFrame::OnSetActiveView(CView* pViewNew, BOOL bNotify)
221 {
222     CFrameWnd::SetActiveView(pViewNew, bNotify);
223 }
224
225 // Private Initialisers
226 int CMainFrame::init_tabs()
227 {
228     CMDITabInfo mdiTabParams;
229     mdiTabParams.m_style = CMFCTabCtrl::STYLE_3D_ONENOTE;
230     mdiTabParams.m_bActiveTabCloseButton = TRUE;
231     mdiTabParams.m_bDocumentMenu = TRUE;
232     mdiTabParams.m_bEnableTabSwap = TRUE;
233     mdiTabParams.m_bFlatFrame = TRUE;
234     mdiTabParams.m_bTabCloseButton = TRUE;
235     mdiTabParams.m_bTabCustomTooltips = FALSE;
236     mdiTabParams.m_bTabIcons = FALSE;
237     //mdiTabParams.m_nTabBorderSize =
          CMFCVisualManager::GetMDITabsBordersSize;
238     mdiTabParams.m_tabLocation =
          CMFCBaseTabCtrl::Location::LOCATION_TOP;
239     this->EnableMDITabbedGroups(TRUE, mdiTabParams);
240
241     return 0;
242 }
243 int CMainFrame::init_menubar()
244 {
245     if (!this->m_wndMenuBar.Create(this))
246     {
247         TRACE0("Failed to create menubar\n");
248         return -1;      // fail to create
249     }
250
251     this->m_wndMenuBar.SetPaneStyle(this->m_wndMenuBar.GetPaneStyle
          () | CBRS_SIZE_DYNAMIC | CBRS_TOOLTIPS | CBRS_FLYBY);
252
```

```
253     // prevent the menu bar from taking the focus on activation
254     CMFCPopupMenu::SetForceMenuFocus(FALSE);
255
256     return 0;
257 }
258 int CMainFrame::init_toolbar()
259 {
260     BOOL bNameValid;
261     if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
          WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
          CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
262         !m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ?
            IDR_MAINFRAME_256 : IDR_MAINFRAME))
263     {
264         TRACE0("Failed to create toolbar\n");
265         return -1;      // fail to create
266     }
267
268     CString strToolBarName;
269     bNameValid = strToolBarName.LoadString(IDS_TOOLBAR_STANDARD);
270     ASSERT(bNameValid);
271     m_wndToolBar.SetWindowText(strToolBarName);
272
273     CString strCustomize;
274     bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
275     ASSERT(bNameValid);
276     m_wndToolBar.EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE,
          strCustomize);
277
278     // Allow user-defined toolbars operations:
279     InitUserToolbars(nullptr, uiFirstUserToolBarId,
          uiLastUserToolBarId);
280
281     return 0;
282 }
283 int CMainFrame::init_statusbar()
284 {
285     if (!this->m_wndStatusBar.Create(this))
286     {
287         TRACE0("Failed to create status bar\n");
288         return -1;      // fail to create
289     }
290     this->m_wndStatusBar.SetIndicators(indicators, sizeof
          (indicators) / sizeof(UINT));
291
292     return 0;
293 }
294 int CMainFrame::init_dockingwindows()
295 {
296     // TODO: Delete these five lines if you don't want the toolbar
          and menubar to be dockable
297     this->DockPane(&this->m_wndMenuBar);
298     this->DockPane(&this->m_wndToolBar);
299     //this->m_wndMenuBar.EnableDocking(CBRS_ALIGN_ANY);
300     //this->m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
301     //this->EnableDocking(CBRS_ALIGN_ANY);
```

```
302
303     // enable Visual Studio 2005 style docking window behavior
304     CDockingManager::SetDockingMode(DT_SMART);
305     // enable Visual Studio 2005 style docking window auto-hide
          behavior
306     this->EnableAutoHidePanes(CBRS_ALIGN_ANY);
307
308     return 0;
309 }
310 int CMainFrame::init_customisation()
311 {
312     BOOL bNameValid;
313     CString strCustomize;
314     bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
315     ASSERT(bNameValid);
316
317     // set the visual manager and style based on persisted value
318     this->OnApplicationLook(theApp.m_nAppLook);
319
320     // Enable enhanced windows management dialog
321     this->EnableWindowsDialog(ID_WINDOW_MANAGER, ID_WINDOW_MANAGER,
          TRUE);
322
323     // Enable toolbar and docking window menu replacement
324     this->EnablePaneMenu(TRUE, ID_VIEW_CUSTOMIZE, strCustomize,
          ID_VIEW_TOOLBAR);
325
326     // enable quick (Alt+drag) toolbar customization
327     CMFCToolBar::EnableQuickCustomization();
328
329     // Switch the order of document name and application name on the
         window title bar. This
330     // improves the usability of the taskbar because the document
          name is visible with the thumbnail.
331     this->ModifyStyle(0, FWS_PREFIXTITLE);
332
333     return 0;
334 }
335 int CMainFrame::init_userimages()
336 {
337     // Load menu item image (not placed on any standard toolbars):
338     CMFCToolBar::AddToolBarForImageCollection(IDR_MENU_IMAGES,
          theApp.m_bHiColorIcons ? IDB_MENU_IMAGES_24 : 0);
339
340     if (CMFCToolBar::GetUserImages() == nullptr)
341     {
342         // load user-defined toolbar images
343         if (this->m_UserImages.Load(_T(".\\UserImages.bmp")))
344         {
345             CMFCToolBar::SetUserImages(&m_UserImages);
346         }
347     }
348
349     return 0;
350 }
351 int CMainFrame::init_commands()
```

```
352 {
353     CList<UINT, UINT> lstBasicCommands;
354 
355     lstBasicCommands.AddTail(ID_FILE_NEW);
356     lstBasicCommands.AddTail(ID_FILE_OPEN);
357     lstBasicCommands.AddTail(ID_FILE_SAVE);
358     lstBasicCommands.AddTail(ID_FILE_PRINT);
359     lstBasicCommands.AddTail(ID_APP_EXIT);
360     lstBasicCommands.AddTail(ID_EDIT_CUT);
361     lstBasicCommands.AddTail(ID_EDIT_PASTE);
362     lstBasicCommands.AddTail(ID_EDIT_UNDO);
363     lstBasicCommands.AddTail(ID_APP_ABOUT);
364     lstBasicCommands.AddTail(ID_VIEW_STATUS_BAR);
365     lstBasicCommands.AddTail(ID_VIEW_TOOLBAR);
366     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2003);
367     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_VS_2005);
368     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_BLUE);
369     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_SILVER);
370     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_BLACK);
371     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_AQUA);
372     lstBasicCommands.AddTail(ID_VIEW_APPLOOK_WINDOWS_7);
373     lstBasicCommands.AddTail(ID_SORTING_SORTALPHABETIC);
374     lstBasicCommands.AddTail(ID_SORTING_SORTBYTYPE);
375     lstBasicCommands.AddTail(ID_SORTING_SORTBYACCESS);
376     lstBasicCommands.AddTail(ID_SORTING_GROUPBYTYPE);
377 
378     CMFCToolBar::SetBasicCommands(lstBasicCommands);
379 
380     return 0;
381 }
382 
```

```
1
2  // ChildFrm.h : interface of the CChildFrame class
3  //
4
5  #pragma once
6
7  class CChildFrame : public CMDIChildWndEx
8  {
9  public:
10     // Implementation
11     CChildFrame() noexcept;
12     virtual ~CChildFrame();
13     DECLARE_DYNCREATE(CChildFrame)
14
15     // Overrides
16     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
17
18 #ifdef _DEBUG
19     virtual void AssertValid() const;
20     virtual void Dump(CDumpContext& dc) const;
21 #endif
22 protected:
23     // Resources
24     CSplitterWndEx m_wndSplitter;
25
26     // Generated message map functions
27     DECLARE_MESSAGE_MAP()
28 };
29
```

```cpp

// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "pch.h"
#include "framework.h"
#include "DesignArk.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWndEx)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWndEx)
END_MESSAGE_MAP()

// CChildFrame construction/destruction

CChildFrame::CChildFrame() noexcept
{
    // TODO: add member initialization code here
}

CChildFrame::~CChildFrame()
{
}


BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying the ↵
      CREATESTRUCT cs
    if( !CMDIChildWndEx::PreCreateWindow(cs) )
        return FALSE;

    return TRUE;
}

// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWndEx::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWndEx::Dump(dc);
}
#endif //_DEBUG
```

```
56
57 // CChildFrame message handlers
58
```

```
1  //{{NO_DEPENDENCIES}}
2  // Microsoft Visual C++ generated include file.
3  // Used by DesignArk.rc
4  //
5  #define IDP_OLE_INIT_FAILED             100
6  #define IDD_SMARTCOLOUR_TYPES           107
7  #define IDD_SMARTCOLOUR_OPS             108
8  #define IDD_SMARTCOLOUR_BUILTIN         109
9  #define IDD_SMARTCOLOUR_OTHER           110
10 #define IDR_POPUP_EDIT                  119
11 #define ID_STATUSBAR_PANE1              120
12 #define ID_STATUSBAR_PANE2              121
13 #define IDS_STATUS_PANE1                122
14 #define IDS_STATUS_PANE2                123
15 #define IDS_TOOLBAR_STANDARD            124
16 #define IDS_TOOLBAR_CUSTOMIZE           125
17 #define ID_VIEW_CUSTOMIZE               126
18 #define IDR_MAINFRAME                   128
19 #define IDR_MAINFRAME_256               129
20 #define IDR_DesignArkTYPE               130
21 #define ID_WINDOW_MANAGER               131
22 #define ID_VIEW_FILEVIEW                133
23 #define ID_VIEW_CLASSVIEW               134
24 #define ID_PROPERTIES                   135
25 #define ID_OPEN                         136
26 #define ID_OPEN_WITH                    137
27 #define ID_DUMMY_COMPILE                138
28 #define ID_CLASS_ADD_MEMBER_FUNCTION    139
29 #define ID_CLASS_ADD_MEMBER_VARIABLE    140
30 #define ID_CLASS_DEFINITION             141
31 #define ID_CLASS_PROPERTIES             142
32 #define ID_NEW_FOLDER                   143
33 #define ID_SORT_MENU                    144
34 #define ID_SORTING_GROUPBYTYPE          145
35 #define ID_SORTING_SORTALPHABETIC       146
36 #define ID_SORTING_SORTBYTYPE           147
37 #define ID_SORTING_SORTBYACCESS         148
38 #define ID_VIEW_PROPERTIESWND           150
39 #define ID_SORTPROPERTIES               151
40 #define ID_PROPERTIES1                  152
41 #define ID_PROPERTIES2                  153
42 #define ID_EXPAND_ALL                   154
43 #define IDS_FILE_VIEW                   155
44 #define IDS_CLASS_VIEW                  156
45 #define IDS_PROPERTIES_WND              158
46 #define IDI_FILE_VIEW                   161
47 #define IDI_FILE_VIEW_HC                162
48 #define IDI_CLASS_VIEW                  163
49 #define IDI_CLASS_VIEW_HC               164
50 #define IDI_PROPERTIES_WND              167
51 #define IDI_PROPERTIES_WND_HC           168
52 #define IDR_EXPLORER                    169
53 #define IDB_EXPLORER_24                 170
54 #define IDR_SORT                        171
55 #define IDB_SORT_24                     172
56 #define IDR_POPUP_SORT                  173
```

```
 57 #define IDR_POPUP_EXPLORER              174
 58 #define IDB_FILE_VIEW                   175
 59 #define IDB_FILE_VIEW_24                176
 60 #define IDB_CLASS_VIEW                  177
 61 #define IDB_CLASS_VIEW_24               178
 62 #define IDR_MENU_IMAGES                 179
 63 #define IDB_MENU_IMAGES_24              180
 64 #define ID_TOOLS_MACRO                  181
 65 #define IDR_PROPERTIES                  183
 66 #define IDB_PROPERTIES_HC               184
 67 #define IDR_THEME_MENU                  200
 68 #define ID_SET_STYLE                    201
 69 #define ID_VIEW_APPLOOK_WIN_2000        205
 70 #define ID_VIEW_APPLOOK_OFF_XP          206
 71 #define ID_VIEW_APPLOOK_WIN_XP          207
 72 #define ID_VIEW_APPLOOK_OFF_2003        208
 73 #define ID_VIEW_APPLOOK_VS_2005         209
 74 #define ID_VIEW_APPLOOK_VS_2008         210
 75 #define ID_VIEW_APPLOOK_OFF_2007_BLUE   215
 76 #define ID_VIEW_APPLOOK_OFF_2007_BLACK  216
 77 #define ID_VIEW_APPLOOK_OFF_2007_SILVER 217
 78 #define ID_VIEW_APPLOOK_OFF_2007_AQUA   218
 79 #define ID_VIEW_APPLOOK_WINDOWS_7       219
 80 #define IDS_EXPLORER                    305
 81 #define IDS_EDIT_MENU                   306
 82 #define IDB_CARET                       310
 83 #define IDB_PNG1                        329
 84 #define IDB_ARROW_RIGHT_HOVER           334
 85 #define IDB_BITMAP1                     336
 86 #define IDB_ARROW_RIGHT_CLICK           336
 87 #define IDD_APP_ABOUT                   343
 88 #define IDC_MFCVSLISTBOX_TYPES          1051
 89 #define IDC_SPLIT1                      1052
 90 #define IDC_COMBO_COLOUR_TYPES          1053
 91 #define IDC_MFCVSLISTBOX_OPS            1064
 92 #define IDC_COMBO_COLOUR_OPS            1065
 93 #define IDC_MFCVSLISTBOX_BUILTIN_CLASSES 1066
 94 #define IDC_SMARTCOLOUR_OTHERS_NUMBERS_COLOUR 1066
 95 #define IDC_MFCVSLISTBOX_OPS2           1066
 96 #define IDC_MFCVSLISTBOX_BUILTIN_FUNCTIONS 1067
 97 #define IDC_COMBO_COLOUR_OPS2           1067
 98 #define IDC_SMARTCOLOUR_OTHERS_NUMBERS_COLOUR2 1067
 99 #define IDC_COMBO_COLOUR_BUILTIN_FUNCTIONS 1068
100 #define IDC_COMBO_COLOUR_BUILTIN_CLASSES 1069
101 #define IDC_SMARTCOLOUR_OTHERS_COMMENT_COMBO 1070
102 #define IDC_SMARTCOLOUR_OTHERS_COMMENT_COMBO2 1071
103 #define IDC_SMARTCOLOUR_OTHERS_FONT     1071
104 #define IDC_SMARTCOLOUR_OTHER_COMMENTS_COLOUR 1074
105 #define IDC_SMARTCOLOR_OTHER_INDENT     1083
106 #define ID_VIEW_SMARTCOLOURWINDOW       32771
107 #define ID_VIEW_                        32772
108 #define ID_HELP_OPENWEBSITE             32773
109 #define ID_HELP_VIEWUSERDOCSONLINE      32776
110
111 // Next default values for new objects
112 //
```

```
113 #ifdef APSTUDIO_INVOKED
114 #ifndef APSTUDIO_READONLY_SYMBOLS
115 #define _APS_NEXT_RESOURCE_VALUE        345
116 #define _APS_NEXT_COMMAND_VALUE         32777
117 #define _APS_NEXT_CONTROL_VALUE         1084
118 #define _APS_NEXT_SYMED_VALUE           310
119 #endif
120 #endif
121
```

```
1  // pch.h: This is a precompiled header file.
2  // Files listed below are compiled only once, improving build performance for future builds.
3  // This also affects IntelliSense performance, including code completion and many code browsing features.
4  // However, files listed here are ALL re-compiled if any one of them is updated between builds.
5  // Do not add files here that you will be updating frequently as this negates the performance advantage.
6
7  #ifndef PCH_H
8  #define PCH_H
9
10 // add headers that you want to pre-compile here
11 #include "framework.h"
12
13 #include <iostream>
14 #include <fstream>
15 #include <sstream>
16 #include <map>
17 #include <string>
18 #include <vector>
19 #include <memory>
20 #include <math.h>
21 #include <shellapi.h>
22 #include <gdiplus.h>
23 #include <stdio.h>
24
25 #endif //PCH_H
26
27 using namespace std;
28
```

```
1 // pch.cpp: source file corresponding to the pre-compiled header
2
3 #include "pch.h"
4
5 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
6
```

```
1  #pragma once
2
3  #ifndef VC_EXTRALEAN
4  #define VC_EXTRALEAN            // Exclude rarely-used stuff from
     Windows headers
5  #endif
6
7  #include "targetver.h"
8
9  #define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // some CString
     constructors will be explicit
10
11 // turns off MFC's hiding of some common and often safely ignored
     warning messages
12 #define _AFX_ALL_WARNINGS
13
14 #include <afxwin.h>         // MFC core and standard components
15 #include <afxext.h>         // MFC extensions
16
17
18 #include <afxdisp.h>        // MFC Automation classes
19
20
21
22 #ifndef _AFX_NO_OLE_SUPPORT
23 #include <afxdtctl.h>           // MFC support for Internet Explorer
     4 Common Controls
24 #endif
25 #ifndef _AFX_NO_AFXCMN_SUPPORT
26 #include <afxcmn.h>             // MFC support for Windows Common
     Controls
27 #endif // _AFX_NO_AFXCMN_SUPPORT
28
29 #include <afxcontrolbars.h>     // MFC support for ribbons and
     control bars
30
31
32
33
34
35
36
37
38
39 #ifdef _UNICODE
40 #if defined _M_IX86
41 #pragma comment(linker,"/manifestdependency:\"type='win32'
     name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
     processorArchitecture='x86' publicKeyToken='6595b64144ccf1df'
     language='*'\"")
42 #elif defined _M_X64
43 #pragma comment(linker,"/manifestdependency:\"type='win32'
     name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
     processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df'
     language='*'\"")
44 #else
```

```
45 #pragma comment(linker,"/manifestdependency:\"type='win32'
     name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
     processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
     language='*'\"")
46 #endif
47 #endif
48
49
50
```

```
1 #pragma once
2
3 // Including SDKDDKVer.h defines the highest available Windows platform.
4
5 // If you wish to build your application for a previous Windows platform, include WinSDKVer.h and
6 // set the _WIN32_WINNT macro to the platform you wish to support before including SDKDDKVer.h.
7
8 #include <SDKDDKVer.h>
9
```

```
1
2  #pragma once
3  #include "CAppObject.h"
4  #include "CTextHandler.h"
5  #include "CTextEditorObject.h"
6
7  class CDocView
8  {
9  public:
10     // Public constructors
11     CDocView();
12     ~CDocView();
13
14 protected:
15
16     // Protected Commands
17     virtual void updateWindow(BOOL caret = TRUE, BOOL window = TRUE)
         = 0;
18
19     // Protected resources
20     CTextHandler textInput;
21     CFont fFont;
22
23     std::map<CString, CBitmap*> m_pBitmaps;
24
25 };
26
27
```

```cpp

#include "pch.h"
#include "framework.h"

#ifndef SHARED_HANDLERS
#include "DesignArk.h"
#endif

#include "CDocView.h"


// Public constructors
CDocView::CDocView()
{
    this->m_pBitmaps[L"caret"] = new CBitmap();
    this->m_pBitmaps[L"caret"]->LoadBitmap(IDB_CARET);
}
CDocView::~CDocView()
{

    for (auto& it : this->m_pBitmaps) {
        delete it.second;
    }
}

void CDocView::updateWindow(BOOL caret, BOOL window)
{
}

```

```
 1
 2  #pragma once
 3  #include "CDocView.h"
 4  #include "CTextDocument.h"
 5
 6  #define PRINT_SIZE          5
 7
 8  #define INITIALISE          1000 // USED FOR DRAWING ENTIRE VIEW
 9  #define TEXT                1001 // USED FOR TEXT EDITS - NEW TEXT,
       BACKSPACE ON TEXT
10  #define RETURN_BACK         1002 // USED FOR ADDING OR REMOVING
       LINES WITH BACKSPACE OR RETURN
11  #define EDITOR_EDIT         1003 // USED FOR ADDING OR REMOVING
       EDITORS
12  #define HIGHLIGHTING        1004 // USED FOR WHEN HIGHLIGHTING TEXT
13  #define SELECT_LINE         1007 // USED WHEN A DIFFERENT LINE IS
       SELECTED
14  #define REMOVE_HIGHLIGHT    1008 // USED WHEN A HIGHLIGHT IS BEING
       DELETED
15  #define CARET_MOVE          1009 // USED WHEN THE CARET IS MOVED
       WITH THE ARROW KEYS
16  #define ADD_RGN_SIDEBAR_BTN 1010 // USED TO ADD THE AREA OF THE
       SIDEBAR BUTTON TO THE REGION
17  #define ADD_RGN_ACTIVE_LINE 1011
18  #define H_SCROLL            1012
19  #define PASTE               1013
20
21
22  class CTextDocView :
23      public CScrollView, public CDocView
24  {
25  protected:
26      // Public constructors
27      CTextDocView() noexcept;
28      virtual ~CTextDocView();
29
30      DECLARE_DYNCREATE(CTextDocView)
31
32      // Public attributes
33      CTextDocument* GetDocument() const;
34      CSize GetDocSize();
35      int getActiveEditor();
36
37      // Public implementations
38  #ifdef _DEBUG
39      virtual void AssertValid() const;
40      virtual void Dump(CDumpContext& dc) const;
41  #endif
42      void refresh(BOOL caret = TRUE, BOOL window = TRUE);
43
44  protected:
45
46      // Protected implementations
47      virtual void updateWindow(BOOL caret = TRUE, BOOL window =
          TRUE);
48      void ResyncScrollSizes(BOOL docSize = TRUE, BOOL reposition =
```

```
        FALSE);
49
50      // Protected overrides
51      virtual void OnDraw(CDC* pDc);
52      afx_msg BOOL OnEraseBkgnd(CDC* pDC);
53
54      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
55      virtual BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
          message);
56      virtual void OnActivateView(BOOL bActivate, CView*
          pActivateView, CView* pDeactiveView);
57
58      virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
59      virtual void OnBeginPrinting(CDC* pDc, CPrintInfo* pInfo);
60      virtual void OnEndPrinting(CDC* pDc, CPrintInfo* pInfo);
61      virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo =
          (CPrintInfo*)0);
62      virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
63
64      // Protected message handlers
65      afx_msg int OnCreate(LPCREATESTRUCT lpcs);
66      virtual void OnInitialUpdate();
67      afx_msg void OnSize(UINT nType, int cx, int cy);
68
69      afx_msg void OnFilePrintPreview();
70
71      afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
          pScrollBar);
72      afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
73      afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
74      afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
75      afx_msg void OnMouseMove(UINT nFlags, CPoint point);
76      afx_msg BOOL OnMouseWheel(UINT nFlags, short zDelta, CPoint pt);
77      afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
78      afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
79      afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
80      afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
81
82      virtual void OnContextMenu(CWnd* pWnd, CPoint point);
83
84      void OnCopy();
85      void OnPaste();
86      void OnCut();
87
88      DECLARE_MESSAGE_MAP()
89
90  private:
91
92      // Private Methods
93      void expandLineBounds();
94      void selectClippingRgn(int nAction, int type = -1);
95
96      void drawHeader(CDC* pDc);
97
98      // Private Resources
99      int activeEditor;
```

```
100
101     int recentEditor; // Used to store the last editor that was used to change lines
102     int recentLine; // Used to store the last line that was used, to be used in combination with 'recentEditor'
103     int recentBlockLine;
104     BOOL recentHlght;
105     int recentPrintZoom;
106
107     CRgn* rgn;
108     CPrintInfo* pInfo;
109     BOOL printing;
110     int printIterator;
111     int returnNewLines;
112 };
113
114 #ifndef _DEBUG  // debug version in DesignArkView.cpp
115 inline CTextDocument* CTextDocView::GetDocument() const
116 {
117     return reinterpret_cast<CTextDocument*>(m_pDocument);
118 }
119 #endif
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
1
2  // CTextDocument.h : interface of the CTextDocument class
3  //
4
5  #pragma once
6  #include "CTextEditorObject.h"
7
8  class CTextDocument : public CDocument
9  {
10 public:
11     // Public constructors
12     CTextDocument() noexcept;
13     virtual ~CTextDocument();
14
15     DECLARE_DYNCREATE(CTextDocument);
16
17     // Public overrides
18     virtual BOOL OnNewDocument();
19     virtual void Serialize(CArchive& ar);
20
21     virtual void OnFileSave();
22     virtual void OnFileOpen();
23
24 #ifdef SHARED_HANDLERS
25     virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);
26     virtual void InitializeSearchContent();
27 #endif // !SHARED_HANDLERS
28
29     // Public implementations
30 #ifdef _DEBUG
31     virtual void AssertValid() const;
32     virtual void Dump(CDumpContext& dc) const;
33 #endif
34
35     // Document Data
36     CArray<CTextEditorObject*> objects;
37     int defBoxHeight;
38     int lineOffset;
39
40     CString filename;
41
42 protected:
43     // Protected message handlers
44     DECLARE_MESSAGE_MAP()
45
46     // Protected commands
47 #ifdef SHARED_HANDLERS
48     void SetSearchContent(const CString& value);
49 #endif // !SHARED_HANDLERS
50 };
51
```

```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
```

```
1
2  #pragma once
3
4  class CTextHandler
5  {
6  public:
7      // Public constructors
8      CTextHandler();
9      ~CTextHandler();
10
11     // Public commands
12     CString RecieveText();
13     void DestroyText();
14
15     // Public message handlers
16     BOOL OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
17     BOOL OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
18
19 private:
20     // Private resources
21     CString text;
22     BOOL shft;
23 };
24
25
```

```cpp
#include "pch.h"
#include "CTextHandler.h"

// Public constructors
CTextHandler::CTextHandler()
{
    this->shft = FALSE;
}
CTextHandler::~CTextHandler()
{
}

// Public commands
CString CTextHandler::RecieveText()
{
    CString temp = this->text;
    this->text = "";
    return temp;
}
void CTextHandler::DestroyText()
{
    this->text = "";
}

// Public message handlers
BOOL CTextHandler::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    BOOL caps = GetKeyState(VK_CAPITAL);

    switch (nChar) {

    case VK_SPACE:
        this->text.Append(L" ");
        break;

    case VK_SHIFT:
        this->shft = TRUE;
        break;

    case 0x30:
        if (this->shft) {
            this->text.Append(L")");
        }
        else {
            this->text.Append(L"0");
        }
        break;

    case 0x31:
        if (this->shft) {
            this->text.Append(L"!");
        }
        else {
            this->text.Append(L"1");
        }
        break;
```

```
57
58      case 0x32:
59          if (this->shft) {
60              this->text.Append(L"\"");
61          }
62          else {
63              this->text.Append(L"2");
64          }
65          break;
66
67      case 0x33:
68          if (this->shft) {
69              this->text.Append(L"£");
70          }
71          else {
72              this->text.Append(L"3");
73          }
74          break;
75
76      case 0x34:
77          if (this->shft) {
78              this->text.Append(L"$");
79          }
80          else {
81              this->text.Append(L"4");
82          }
83          break;
84
85      case 0x35:
86          if (this->shft) {
87              this->text.Append(L"%");
88          }
89          else {
90              this->text.Append(L"5");
91          }
92          break;
93
94      case 0x36:
95          if (this->shft) {
96              this->text.Append(L"^");
97          }
98          else {
99              this->text.Append(L"6");
100         }
101         break;
102
103     case 0x37:
104         if (this->shft) {
105             this->text.Append(L"&");
106         }
107         else {
108             this->text.Append(L"7");
109         }
110         break;
111
112     case 0x38:
```

```
113             if (this->shft) {
114                 this->text.Append(L"*");
115             }
116             else {
117                 this->text.Append(L"8");
118             }
119             break;
120 
121         case 0x39:
122             if (this->shft) {
123                 this->text.Append(L"(");
124             }
125             else {
126                 this->text.Append(L"9");
127             }
128             break;
129 
130         case 0x41:
131             if (!(this->shft && caps) && (this->shft || caps)) {
132                 this->text.Append(L"A");
133             }
134             else {
135                 this->text.Append(L"a");
136             }
137             break;
138 
139         case 0x42:
140             if (!(this->shft && caps) && (this->shft || caps)) {
141                 this->text.Append(L"B");
142             }
143             else {
144                 this->text.Append(L"b");
145             }
146             break;
147 
148         case 0x43:
149             if (!(this->shft && caps) && (this->shft || caps)) {
150                 this->text.Append(L"C");
151             }
152             else {
153                 this->text.Append(L"c");
154             }
155             break;
156 
157         case 0x44:
158             if (!(this->shft && caps) && (this->shft || caps)) {
159                 this->text.Append(L"D");
160             }
161             else {
162                 this->text.Append(L"d");
163             }
164             break;
165 
166         case 0x45:
167             if (!(this->shft && caps) && (this->shft || caps)) {
168                 this->text.Append(L"E");
```

```
169             }
170             else {
171                 this->text.Append(L"e");
172             }
173             break;
174
175         case 0x46:
176             if (!(this->shft && caps) && (this->shft || caps)) {
177                 this->text.Append(L"F");
178             }
179             else {
180                 this->text.Append(L"f");
181             }
182             break;
183
184         case 0x47:
185             if (!(this->shft && caps) && (this->shft || caps)) {
186                 this->text.Append(L"G");
187             }
188             else {
189                 this->text.Append(L"g");
190             }
191             break;
192
193         case 0x48:
194             if (!(this->shft && caps) && (this->shft || caps)) {
195                 this->text.Append(L"H");
196             }
197             else {
198                 this->text.Append(L"h");
199             }
200             break;
201
202         case 0x49:
203             if (!(this->shft && caps) && (this->shft || caps)) {
204                 this->text.Append(L"I");
205             }
206             else {
207                 this->text.Append(L"i");
208             }
209             break;
210
211         case 0x4A:
212             if (!(this->shft && caps) && (this->shft || caps)) {
213                 this->text.Append(L"J");
214             }
215             else {
216                 this->text.Append(L"j");
217             }
218             break;
219
220         case 0x4B:
221             if (!(this->shft && caps) && (this->shft || caps)) {
222                 this->text.Append(L"K");
223             }
224             else {
```

```
225             this->text.Append(L"k");
226         }
227         break;
228
229     case 0x4C:
230         if (!(this->shft && caps) && (this->shft || caps)) {
231             this->text.Append(L"L");
232         }
233         else {
234             this->text.Append(L"l");
235         }
236         break;
237
238     case 0x4D:
239         if (!(this->shft && caps) && (this->shft || caps)) {
240             this->text.Append(L"M");
241         }
242         else {
243             this->text.Append(L"m");
244         }
245         break;
246
247     case 0x4E:
248         if (!(this->shft && caps) && (this->shft || caps)) {
249             this->text.Append(L"N");
250         }
251         else {
252             this->text.Append(L"n");
253         }
254         break;
255
256     case 0x4F:
257         if (!(this->shft && caps) && (this->shft || caps)) {
258             this->text.Append(L"O");
259         }
260         else {
261             this->text.Append(L"o");
262         }
263         break;
264
265     case 0x50:
266         if (!(this->shft && caps) && (this->shft || caps)) {
267             this->text.Append(L"P");
268         }
269         else {
270             this->text.Append(L"p");
271         }
272         break;
273
274     case 0x51:
275         if (!(this->shft && caps) && (this->shft || caps)) {
276             this->text.Append(L"Q");
277         }
278         else {
279             this->text.Append(L"q");
280         }
```

```
281         break;
282
283     case 0x52:
284         if (!(this->shft && caps) && (this->shft || caps)) {
285             this->text.Append(L"R");
286         }
287         else {
288             this->text.Append(L"r");
289         }
290         break;
291
292     case 0x53:
293         if (!(this->shft && caps) && (this->shft || caps)) {
294             this->text.Append(L"S");
295         }
296         else {
297             this->text.Append(L"s");
298         }
299         break;
300
301     case 0x54:
302         if (!(this->shft && caps) && (this->shft || caps)) {
303             this->text.Append(L"T");
304         }
305         else {
306             this->text.Append(L"t");
307         }
308         break;
309
310     case 0x55:
311         if (!(this->shft && caps) && (this->shft || caps)) {
312             this->text.Append(L"U");
313         }
314         else {
315             this->text.Append(L"u");
316         }
317         break;
318
319     case 0x56:
320         if (!(this->shft && caps) && (this->shft || caps)) {
321             this->text.Append(L"V");
322         }
323         else {
324             this->text.Append(L"v");
325         }
326         break;
327
328     case 0x57:
329         if (!(this->shft && caps) && (this->shft || caps)) {
330             this->text.Append(L"W");
331         }
332         else {
333             this->text.Append(L"w");
334         }
335         break;
336
```

```
337         case 0x58:
338             if (!(this->shft && caps) && (this->shft || caps)) {
339                 this->text.Append(L"X");
340             }
341             else {
342                 this->text.Append(L"x");
343             }
344             break;
345 
346         case 0x59:
347             if (!(this->shft && caps) && (this->shft || caps)) {
348                 this->text.Append(L"Y");
349             }
350             else {
351                 this->text.Append(L"y");
352             }
353             break;
354 
355         case 0x5A:
356             if (!(this->shft && caps) && (this->shft || caps)) {
357                 this->text.Append(L"Z");
358             }
359             else {
360                 this->text.Append(L"z");
361             }
362             break;
363 
364         case VK_MULTIPLY:
365             this->text.Append(L"*");
366             break;
367 
368         case VK_ADD:
369             this->text.Append(L"+");
370             break;
371 
372         case VK_SUBTRACT:
373             this->text.Append(L"-");
374             break;
375 
376         case VK_DIVIDE:
377             this->text.Append(L"/");
378             break;
379 
380         case VK_OEM_1:
381             if (this->shft) {
382                 this->text.Append(L":");
383             }
384             else {
385                 this->text.Append(L";");
386             }
387             break;
388 
389         case VK_OEM_PLUS:
390             if (this->shft) {
391                 this->text.Append(L"+");
392             }
```

```
393             else {
394                 this->text.Append(L"=");
395             }
396             break;
397 
398         case VK_OEM_COMMA:
399             if (this->shft) {
400                 this->text.Append(L"<");
401             }
402             else {
403                 this->text.Append(L",");
404             }
405             break;
406 
407         case VK_OEM_MINUS:
408             if (this->shft) {
409                 this->text.Append(L"_");
410             }
411             else {
412                 this->text.Append(L"-");
413             }
414             break;
415 
416         case VK_OEM_PERIOD:
417             if (this->shft) {
418                 this->text.Append(L">");
419             }
420             else {
421                 this->text.Append(L".");
422             }
423             break;
424 
425         case VK_OEM_2:
426             if (this->shft) {
427                 this->text.Append(L"?");
428             }
429             else {
430                 this->text.Append(L"/");
431             }
432             break;
433 
434         case VK_OEM_3:
435             if (this->shft) {
436                 this->text.Append(L"@");
437             }
438             else {
439                 this->text.Append(L"'");
440             }
441             break;
442 
443         case VK_OEM_4:
444             if (this->shft) {
445                 this->text.Append(L"{");
446             }
447             else {
448                 this->text.Append(L"[");
```

```
449             }
450             break;
451
452         case VK_OEM_5:
453             if (this->shft) {
454                 this->text.Append(L"|");
455             }
456             else {
457                 this->text.Append(L"\\");
458             }
459             break;
460
461         case VK_OEM_6:
462             if (this->shft) {
463                 this->text.Append(L"}");
464             }
465             else {
466                 this->text.Append(L"]");
467             }
468             break;
469
470         default:
471             return FALSE;
472
473         }
474         return TRUE;
475     }
476     BOOL CTextHandler::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
477     {
478         if (nChar == VK_SHIFT) {
479             this->shft = FALSE;
480         }
481         return 0;
482     }
483
484
```

```
1
2  #pragma once
3  #include "DesignArk.h"
4
5  class CAppObject
6  {
7  public:
8      // Public constructors
9      CAppObject(CRect bounds, CString ID, BOOL active = FALSE);
10     ~CAppObject();
11
12     // Public commands
13     virtual void draw(CDC* pDc);
14
15     // Public message handers
16     virtual void OnSize(UINT nType, int cx, int cy);
17
18     virtual BOOL OnLButtonUp(UINT nFlags, CPoint point);
19     virtual BOOL OnLButtonDown(UINT nFlags, CPoint point);
20     virtual void OnLButtonDblClk(UINT nFlags, CPoint point);
21     virtual void OnRButtonUp(UINT nFlags, CPoint point);
22     virtual void OnRButtonDown(UINT nFlags, CPoint point);
23     virtual BOOL OnMouseMove(UINT nFlags, CPoint point);
24
25     virtual int OnRecieveText(CString text);
26     virtual void OnRecieveBackspace();
27     virtual void OnRecieveReturn();
28     virtual void OnRecieveTab();
29
30     virtual void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
31
32     virtual BOOL OnContextMenu(CWnd* pWnd, CPoint point);
33
34     // Public getters, setters and checkers
35     CString getID();
36     void setID(CString ID);
37
38     CRect getBounds();
39     BOOL checkPointInBounds(CPoint point);
40
41     virtual void setBounds(CRect bounds);
42     virtual void setBounds(int x, int y, int cx, int cy);
43     virtual void setPosition(CPoint point);
44     virtual void move(int x, int y);
45
46     BOOL isActive();
47     virtual void setActive(BOOL val);
48
49 protected:
50     CString ID;
51     CRect bounds;
52     BOOL active;
53 };
54
55
```

```cpp
#include "pch.h"
#include "CAppObject.h"

// Public constructors
CAppObject::CAppObject(CRect bounds, CString ID, BOOL active)
{
    this->bounds = bounds;
    this->ID = ID;
    this->active = active;
}
CAppObject::~CAppObject()
{
}

void CAppObject::draw(CDC* pDc)
{
}

// Public message handlers
void CAppObject::OnSize(UINT nType, int cx, int cy)
{
}

BOOL CAppObject::OnLButtonUp(UINT nFlags, CPoint point)
{
    return 0;
}
BOOL CAppObject::OnLButtonDown(UINT nFlags, CPoint point)
{
    return 0;
}
void CAppObject::OnLButtonDblClk(UINT nFlags, CPoint point)
{
}
void CAppObject::OnRButtonUp(UINT nFlags, CPoint point)
{
}
void CAppObject::OnRButtonDown(UINT nFlags, CPoint point)
{
}
BOOL CAppObject::OnMouseMove(UINT nFlags, CPoint point)
{
    return TRUE;
}
int CAppObject::OnRecieveText(CString text)
{
    return 0;
}
void CAppObject::OnRecieveBackspace()
{
}
void CAppObject::OnRecieveReturn()
{
}
void CAppObject::OnRecieveTab()
{
```

```
 57 }
 58
 59 void CAppObject::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
 60 {
 61 }
 62
 63 BOOL CAppObject::OnContextMenu(CWnd* pWnd, CPoint point)
 64 {
 65     return 0;
 66 }
 67
 68 // Public getters, setters and checkers
 69 CString CAppObject::getID()
 70 {
 71     return this->ID;
 72 }
 73 void CAppObject::setID(CString ID)
 74 {
 75     this->ID = ID;
 76 }
 77
 78 CRect CAppObject::getBounds()
 79 {
 80     return this->bounds;
 81 }
 82 BOOL CAppObject::checkPointInBounds(CPoint point)
 83 {
 84     if (this->bounds.left <= point.x && bounds.right >= point.x &&
 85         this->bounds.top <= point.y && this->bounds.bottom >=
               point.y) {
 86         return TRUE;
 87     }
 88
 89     return FALSE;
 90 }
 91
 92 void CAppObject::setBounds(CRect bounds)
 93 {
 94     this->bounds = bounds;
 95 }
 96 void CAppObject::setBounds(int x, int y, int cx, int cy)
 97 {
 98     this->bounds.left = x;
 99     this->bounds.top = y;
100     this->bounds.right = x + cx;
101     this->bounds.bottom = y + cy;
102 }
103 void CAppObject::setPosition(CPoint point)
104 {
105     this->move(point.x - this->bounds.left, point.y - this-
          >bounds.top);
106 }
107 void CAppObject::move(int x, int y)
108 {
109     this->bounds.left += x;
110     this->bounds.right += x;
```

```
111     this->bounds.top += y;
112     this->bounds.bottom += y;
113 }
114
115 BOOL CAppObject::isActive()
116 {
117     return this->active;
118 }
119 void CAppObject::setActive(BOOL val)
120 {
121     this->active = val;
122 }
123
```

```
#pragma once
#include "CTextLineObject.h"


class CTextEditorObject :
    public CAppObject
{
public:
    // Public contructors
    CTextEditorObject(CRect bounds, CString ID, BOOL lineNums, int maxLines = 0, int defBoxHeight = 0, BOOL active = FALSE, std::vector<CString> text = {L""}, std::vector<int> line = {1}, int lineOffset = 0);
    ~CTextEditorObject();

    // Public implementations
    int draw(CDC* pDc, CSize textExtent, int xScrollPosition, int returnNewLines, BOOL printing, CRect printAreaLength = CRect());

    // Public getters, setters and checkers
    int getCaretPos();
    CPoint getCaretPoint(CSize caretSize);
    CSize getTextExtent();
    void setTextExtent(CSize size);

    int getRecentPos();

    BOOL pointHighlighted(CPoint point);
    BOOL hasHighlight();
    void hlghtingOff();
    CRect getHighlightClippingRect();
    CRgn* getHighlightExactRgn(int x_offset, int y_offset);
    int getStartLine();
    BOOL isHlghtMultiline();
    int lineHighlight(int line);
    // RETURNS  0 for none
    //          1 for partially highlighted
    //          2 for line highlighted

    std::vector <int> iGetLineNum(int a = 0);
    CString sGetLineNum(int a = 0);

    void incrementSublines(int subline, int val);
    int getActiveLine();
    int getBlockLine();
    int getPrintLine();
    CString getLineText(int line = 0);
    CRect getLineBounds(int line = 0);
    CString getHighlightedText();
    int getLineTextWidth();
    int getNumLines();
    int getBoxHeight(BOOL default = FALSE);

    virtual void setActive(BOOL active);
    virtual void move(int x, int y);
```

```
52      virtual void setBounds(CRect bounds);
53      std::tuple<CRect, int> getPrintBounds(int returnNewLines, int
          printAreaLength);
54
55      int getCursorArrow();
56
57      void initialise();
58
59      // Public message handlers
60      virtual void OnSize(UINT nType, int x, int y);
61
62      virtual BOOL OnLButtonUp(UINT nFlags, CPoint point);
63      virtual BOOL OnLButtonDown(UINT nFlags, CPoint point);
64      virtual void OnLButtonDblClk(UINT nFlags, CPoint point);
65      virtual void OnRButtonUp(UINT nFlags, CPoint point);
66      virtual void OnRButtonDown(UINT nFlags, CPoint point);
67      virtual BOOL OnMouseMove(UINT nFlags, CPoint point);
68
69      virtual int OnRecieveText(CString text, BOOL open = FALSE);
70      virtual void OnRecieveBackspace();
71      virtual void OnRecieveTab();
72      virtual void OnRecieveReturn(BOOL open = FALSE);
73
74      virtual void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
75      virtual BOOL OnContextMenu(CWnd* pWnd, CPoint point);
76
77  private:
78
79      // Private Implementations
80      void setSidebar();
81      void setHighlighter();
82      void setBracketsNull();
83
84      // Private resources
85
86      // Positioning
87      int activeLine;
88      int mouseLine;
89      int blockLine;
90      int printLine;
91      int caretPos;
92
93      // Sizes
94      int defBoxHeight;
95      CSize textExtent;
96
97      // Highlights
98      BOOL hlght;             // Used to tell the editor that there is
          a highlight
99      BOOL hlghting;          // Used to tell that the user is
          currently highlighting
100     CPoint hlghtStartP;     // Stores the start of the highlight
101     CPoint hlghtEndP;       // Stores the end/current mouse pos for
          the highlighting
102     int startPos;           // Stores the original starting pos in
          terms of character spaces for the highlight
```

```
103     int startLine;          // Stores the original starting line for the highlight
104
105     // Recents
106     BOOL recentHlght;
107     int recentPos;
108     int recentLine;
109
110     // Line stuff
111     std::vector<CTextLineObject*> lines;
112     int maxLines;
113     int lineOffset;
114     BOOL lineNums;
115     BOOL hoverSubLine;
116     BOOL clickSubLine;
117     CRect sidebar;
118
119     // Editing
120     std::vector<std::vector<int>> brackets;
121
122     /*
123
124     Brackets contain the positions and types of all the auto completed brackets currently in the editor.
125     Auto-brackets must always be on the same line and it will only be auto for as long the user is editing that line.
126
127     Each record in brackets contain three integers.
128
129     [x][0] contains the position of the first bracket.
130     [x][1] contains the position of the second bracket.
131     [x][2] contains the type of bracket that has been saved.
132
133     if [x][2] == 0 : type = (
134                  1 : type = {
135                  2 : type = [
136                  3 : type = "
137                  4 : type = '
138
139     */
140
141
142     BOOL bracketContains(int value, int& itPos, int type, int side = 1);
143     void moveBrackets(int val, int index);
144
145     // Stuff for view
146     int cursor_arrow;
147     BOOL lMouseDown;
148 };
149
150
```

```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
```

```
1  #pragma once
2
3  #include "CAppObject.h"
4
5  class CTextLineObject
6      : public CAppObject
7  {
8  public:
9      // Public constructors
10     CTextLineObject(CRect bounds, CString ID, std::vector<int>
         lineNums, CString text = L"", BOOL active = FALSE);
11     ~CTextLineObject();
12
13     //DECLARE_SERIAL(CTextLineObject)
14
15     // Public commands
16     int draw(CDC* pDc, CSize textExtent,
         std::vector<std::vector<int>> brackets, int returnNewLines,
         BOOL printing, int printAreaLength = -1);
17
18     // Public getters & setters
19     CString getText();
20     int getLength();
21     void setText(CString text);
22     void concatenateString(CString text, int position);
23     void backspace(int position);
24
25     BOOL isHighlighting();
26     void setHighlighting(BOOL val);
27     void setHighlighter(int pos1, int pos2);
28     void highlightLine();
29     BOOL isLineHighlighted();
30     int getHStart();
31     int getHEnd();
32     CString getHighlightedText();
33
34     int getNumSubLines();
35     std::vector<int> iGetLineNums();
36     CString sGetLineNums();
37     void addSublines(std::vector<int> subs);
38     void incrementLine(int subline, int val);
39
40     virtual void setBounds(CRect bounds);
41     virtual void move(int x, int y);
42
43 private:
44
45     // Private Functions
46     BOOL onlyNums(CString str);
47     BOOL brkPosContains(std::vector<std::vector<int>> vector, int
         value, int side = 1);
48
49     // Private resources
50     CString text;
51
52     Gdiplus::Rect highlighter;
```

```
53      BOOL highlight;
54      int hStart;
55      int hEnd;
56      BOOL lineHighlight;
57
58      int numSubLines;
59      std::vector<int> lineNums;
60
61      BOOL smartColour;
62  };
63
64
```

```cpp
#include "pch.h"
#include "CTextLineObject.h"

// Public constructors
CTextLineObject::CTextLineObject(CRect bounds, CString ID, std::vector<int> lineNums, CString text, BOOL active)
    : CAppObject(bounds, ID, active)
{
    this->text = text;

    this->highlighter.Y = bounds.top;
    this->highlighter.Height = bounds.Height();
    this->highlight = FALSE;
    hStart = 0;
    hEnd = 0;

    this->numSubLines = lineNums.size();
    this->lineNums = lineNums;

    this->smartColour = TRUE;
}
CTextLineObject::~CTextLineObject()
{
}

// Public commands
int CTextLineObject::draw(CDC* pDc, CSize textExtent, std::vector<vector<int>> brackets, int returnNewLines, BOOL printing, int printAreaLength)
{
    // Local variables
    BOOL autoBrkt = brackets.size() > 0; // If we have any auto brackets (used when not printing)
    int lengthPrint = 0; // Stores the current length of the print on the line that is currently being drawn to (only used when printing)

    // Gdiplus::Bitmap newLineArrow(L"res/newLine.png"); // Taking this out as it does not work and is not needed
    int printMaxCharLength = (printAreaLength - this->bounds.left * 2)/textExtent.cx;

    // If we art using SmartColour
    if (this->smartColour) {

        // Local variables
        CString word = L""; // Stores the word that is being found and is being colour checked
        CString character = L""; // Stores any characters that should seperate words
        int pos = 0; // Stores the position that is being drawn at
        BOOL comment = FALSE; // Stores whether the rest of the line is a comment
        int i = 0; // iterator for the loop, letter position that is being looked at

```

```
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 seperate 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 find if the current word matches one in the list
100                         int it = 0, n = theApp.smartColour_String.size();
101
102                         while (!newColour && it < n) {
103                             // Binary Search
104                             int it2 = 0, n2 = theApp.smartColour_String[it].size();
105                             int min = 0, max = (n2)-1;
106
107
108                             while (!newColour && it2 < n2) {
109
110                                 if (max < min) {
111                                     newColour == TRUE;
112                                 }
113
114                                 int guess = floor((min + max) / 2);
115
116                                 CString tempWord = word;
117                                 CString tempAppWord = theApp.smartColour_String[it][guess];
118
119                                 tempWord.MakeLower();
120                                 tempAppWord.MakeLower();
121
122                                 // If this word on the list matches our searching word
123                                 if (tempWord == tempAppWord) {
124                                     newColour = TRUE;
125                                     pDc->SetTextColor (theApp.smartColour_Colour[it]); // Set the word's colour
126                                 }
127                                 else if (theApp.smartColour_String [it][guess] < word) {
128                                     min = guess + 1;
129                                 }
130                                 else {
131                                     max = guess - 1;
132                                 }
133                                 it2++;
134                             }
135                             it++;
136                         }
137                     }
138
139                     // Drawing the word
140
141                     // Drawing if we have a printer
142                     if (printing) {
143
```

```
144                         // Add the length if the word to the length of our print
145                         lengthPrint += word.GetLength();
146
147                         while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines
148
149                             CString tempWord = L"";
150                             if ((i - word.GetLength()) % printMaxCharLength != 0) {
151                                 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
152                                 word = word.Mid(printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Change the word to be whatever we didn't print and continue the loop
153                             }
154                             else {
155                                 pos++;
156                             }
157
158                             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
159
160                             lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
161
162                             /*pos += tempWord.GetLength() + 2;
163
164                             Gdiplus::Graphics g(pDc->GetSafeHdc());
165                             Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
166                                 textExtent.cx, textExtent.cx);
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 seperation
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                     }
```

```
249                 }
250
251                 else if (character == L"\"" || character == L"'") {
252
253                     pDc->SetTextColor(COLORREF(RGB(255, 128, 0)));
254
255                     int j = i + 1;
256                     BOOL found = FALSE;
257
258                     while (!found && j < this->text.GetLength()) {
259
260                         if (this->text.Mid(j, 1) == character) {
261
262                             word = this->text.Mid(i, j - i + 1);
263
264                             if (printing) {
265
266                                 // Add the length if the word to the length of our print
267                                 lengthPrint += word.GetLength();
268
269                                 while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines
270
271                                     CString tempWord = L"";
272                                     if ((i - word.GetLength()) % printMaxCharLength != 0) {
273                                         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
274                                         word = word.Mid (printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Change the word to be whatever we didn't print and continue the loop
275                                     }
276                                     else {
277                                         pos++;
278                                     }
279
280                                     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
281
282                                     lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
283
284                                     /*pos += tempWord.GetLength() + 2;
```

```
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.cx);
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(),
365                             textExtent.cx, textExtent.cy);
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(RGB(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 if 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->text.Mid(position, this->text.GetLength());
459         }
460         else {
461             this->text = this->text.Mid(0, position) + text + this->text.Mid(position + 1, this->text.GetLength());
462         }
463     }
464     void CTextLineObject::backspace(int position)
465     {
466         if (!this->highlight) {
467             this->text = this->text.Mid(0, position - 1) + this->text.Mid(position, this->text.GetLength());
468         }
469         else {
470             this->text = this->text.Mid(0, min(this->hStart, this->hEnd)) + this->text.Mid(max(this->hStart, this->hEnd), this->text.GetLength());
471             this->setHighlighting(FALSE);
472         }
473     }
474
475     BOOL CTextLineObject::isHighlighting()
476     {
477         return this->highlight;
478     }
479     void CTextLineObject::setHighlighting(BOOL val)
480     {
481         if (!val) {
482             this->highlighter.X = 0;
483             this->highlighter.Width = 0;
484             this->lineHighlight = FALSE;
485         }
486         this->highlight = val;
487     }
```

```
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
```

```
1  #pragma once
2
3  #include "CSmartColourStatements.h"
4  #include "CSmartColourTypes.h"
5  #include "CSmartColourBuiltin.h"
6  #include "CSmartColourOther.h"
7
8  // CSmartColour
9
10 class CSmartColour : public CPropertySheet
11 {
12
13 public:
14     CSmartColour(std::vector<std::vector<CString>> page_strings,
         std::vector<COLORREF> page_colour,
15                      CString commentType, COLORREF commentColour,
                    COLORREF numberColour, COLORREF highlightColour,
                    CString fonts, CString indents,
16                      LPCTSTR pszCaption = L"", CWnd* pParentWnd =
                    nullptr, UINT iSelectPage = 0);
17     virtual ~CSmartColour();
18
19 protected:
20
21     CSmartColourTypes types;
22     CSmartColourStatements statements;
23     CSmartColourBuiltin builtin;
24     CSmartColourOther other;
25
26     DECLARE_DYNAMIC(CSmartColour)
27     DECLARE_MESSAGE_MAP()
28 };
29
30
31
```

```cpp
// CSmartColour.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"

#include "CSmartColour.h"

// CSmartColour

IMPLEMENT_DYNAMIC(CSmartColour, CPropertySheet)

CSmartColour::CSmartColour(std::vector<std::vector<CString>> page_strings, std::vector<COLORREF> page_colour,
                           CString commentType, COLORREF commentColour, COLORREF numberColour, COLORREF highlightColour, CString fonts, CString indents,
                           LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage),
    statements(page_strings[0], page_colour[0], page_strings[1], page_colour[1]),
    types(page_strings[2], page_colour[2]),
    builtin(page_strings[3], page_colour[3], page_strings[4], page_colour[4]),
    other(commentType, commentColour, numberColour, highlightColour, fonts, indents)
{
    this->AddPage(&this->statements);
    this->AddPage(&this->types);
    this->AddPage(&this->builtin);
    this->AddPage(&this->other);
}

CSmartColour::~CSmartColour()
{
}


BEGIN_MESSAGE_MAP(CSmartColour, CPropertySheet)
END_MESSAGE_MAP()


// CSmartColour message handlers

```

```
1  #pragma once
2  #include "afxdialogex.h"
3
4
5  // CSmartColourOps dialog
6
7  class CSmartColourStatements : public CMFCPropertyPage
8  {
9
10 public:
11     CSmartColourStatements(std::vector<CString> syntax_string,
         COLORREF syntax_colour, std::vector<CString> statement_string,
         COLORREF statement_colour, CWnd* pParent = nullptr);   //
         standard constructor
12     virtual ~CSmartColourStatements();
13
14 // Dialog Data
15 #ifdef AFX_DESIGN_TIME
16     enum { IDD = IDD_SMARTCOLOUR_OPS };
17 #endif
18
19 protected:
20     virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
         support
21     virtual BOOL OnInitDialog();
22     virtual BOOL OnKillActive();
23
24     DECLARE_MESSAGE_MAP()
25     DECLARE_DYNAMIC(CSmartColourStatements)
26 public:
27     // Controls
28     CVSListBox m_Listbox_Statements_syntax_control;
29     CMFCColorButton m_Combo_Statements_syntax_control;
30
31     CVSListBox m_Listbox_Statements_statments_control;
32     CMFCColorButton m_Combo_Statements_statments_control;
33
34     // Variables
35     std::vector<CString> m_Strings_Statements_syntax_var;
36     COLORREF m_Colour_Statements_syntax_var;
37
38     std::vector<CString> m_Strings_Statements_statements_var;
39     COLORREF m_Colour_Statements_statements_var;
40
41 };
42
```

```cpp
// CSmartColourOps.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"
#include "afxdialogex.h"
#include "CSmartColourStatements.h"


// CSmartColourOps dialog

IMPLEMENT_DYNAMIC(CSmartColourStatements, CMFCPropertyPage)

CSmartColourStatements::CSmartColourStatements(std::vector<CString> syntax_string, COLORREF syntax_colour, std::vector<CString> statement_string, COLORREF statement_colour, CWnd* pParent/* = nullptr*/)
    : CMFCPropertyPage(IDD_SMARTCOLOUR_OPS /*, pParent*/)
    , m_Listbox_Statements_syntax_control()
    , m_Listbox_Statements_statments_control()
{
    this->m_Strings_Statements_syntax_var = syntax_string;
    this->m_Colour_Statements_syntax_var = syntax_colour;

    this->m_Strings_Statements_statements_var = statement_string;
    this->m_Colour_Statements_statements_var = statement_colour;
}

CSmartColourStatements::~CSmartColourStatements()
{
}

void CSmartColourStatements::DoDataExchange(CDataExchange* pDX)
{
    CMFCPropertyPage::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_MFCVSLISTBOX_OPS, m_Listbox_Statements_statments_control);
    DDX_Control(pDX, IDC_COMBO_COLOUR_OPS, m_Combo_Statements_statments_control);
    DDX_Control(pDX, IDC_MFCVSLISTBOX_OPS2, m_Listbox_Statements_syntax_control);
    DDX_Control(pDX, IDC_COMBO_COLOUR_OPS2, m_Combo_Statements_syntax_control);
}


BOOL CSmartColourStatements::OnInitDialog()
{
    CDialog::OnInitDialog();

    for (auto& it : this->m_Strings_Statements_syntax_var) {
        this->m_Listbox_Statements_syntax_control.AddItem(it);
    }

    this->m_Combo_Statements_syntax_control.SetColor(this->m_Colour_Statements_syntax_var);
```

```
49
50      for (auto& it : this->m_Strings_Statements_statements_var) {
51          this->m_Listbox_Statements_statments_control.AddItem(it);
52      }
53
54      this->m_Combo_Statements_statments_control.SetColor(this->m_Colour_Statements_statements_var);
55
56      return 0;
57  }
58
59  BOOL CSmartColourStatements::OnKillActive()
60  {
61      // Syntax
62      this->m_Strings_Statements_syntax_var.resize(this->m_Listbox_Statements_syntax_control.GetCount());
63
64      for (int i = 0; i < this->m_Listbox_Statements_syntax_control.GetCount(); i++) {
65          this->m_Strings_Statements_syntax_var[i] = this->m_Listbox_Statements_syntax_control.GetItemText(i);
66      }
67
68      // Insertion Sort
69      for (int i = 1; i < this->m_Strings_Statements_syntax_var.size(); i++) {
70          CString currentValue = this->m_Strings_Statements_syntax_var[i];
71          int pos = i;
72
73          while (pos > 0 && this->m_Strings_Statements_syntax_var[pos - 1] > currentValue) {
74              this->m_Strings_Statements_syntax_var[pos] = this->m_Strings_Statements_syntax_var[pos - 1];
75              pos--;
76          }
77          this->m_Strings_Statements_syntax_var[pos] = currentValue;
78      }
79
80      this->m_Colour_Statements_syntax_var = this->m_Combo_Statements_syntax_control.GetColor();
81
82      // Statements
83      this->m_Strings_Statements_statements_var.resize(this->m_Listbox_Statements_statments_control.GetCount());
84
85      for (int i = 0; i < this->m_Listbox_Statements_statments_control.GetCount(); i++) {
86          this->m_Strings_Statements_statements_var[i] = this->m_Listbox_Statements_statments_control.GetItemText(i);
87      }
88
89      // Insertion Sort
90      for (int i = 1; i < this->m_Strings_Statements_statements_var.size(); i++) {
91          CString currentValue = this-
```

```
            >m_Strings_Statements_statements_var[i];
92          int pos = i;
93
94          while (pos > 0 && this->m_Strings_Statements_statements_var
              [pos - 1] > currentValue) {
95            this->m_Strings_Statements_statements_var[pos] = this-
                >m_Strings_Statements_statements_var[pos - 1];
96            pos--;
97          }
98          this->m_Strings_Statements_statements_var[pos] =
              currentValue;
99      }
100
101     this->m_Colour_Statements_statements_var = this-
          >m_Combo_Statements_statments_control.GetColor();
102
103     return CMFCPropertyPage::OnKillActive();
104 }
105
106
107
108 BEGIN_MESSAGE_MAP(CSmartColourStatements, CMFCPropertyPage)
109 END_MESSAGE_MAP()
110
111
112 // CSmartColourOps message handlers
113
```

```cpp
#pragma once
#include "afxdialogex.h"


// CSmartColourTypes dialog

class CSmartColourTypes : public CMFCPropertyPage
{

public:
    CSmartColourTypes(std::vector<CString> types_string, COLORREF colour, CWnd* pParent = nullptr);   // standard constructor
    virtual ~CSmartColourTypes();

// Dialog Data
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_SMARTCOLOUR_TYPES };
#endif#

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnInitDialog();
    virtual BOOL OnKillActive();

    DECLARE_DYNAMIC(CSmartColourTypes)
    DECLARE_MESSAGE_MAP()

public:

    // Controls
    CVSListBox m_Listbox_Types_control;
    CMFCColorButton m_Combo_Types_colour_control;

    // Variables
    COLORREF m_Combo_Types_colour_var;
    std::vector<CString> m_listbox_types_strings;
};

```

```cpp
// CSmartColourTypes.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"
#include "afxdialogex.h"
#include "CSmartColourTypes.h"


// CSmartColourTypes dialog

IMPLEMENT_DYNAMIC(CSmartColourTypes, CMFCPropertyPage)

CSmartColourTypes::CSmartColourTypes(std::vector<CString> types_string, COLORREF colour, CWnd* pParent /*=nullptr*/)
    : CMFCPropertyPage(IDD_SMARTCOLOUR_TYPES/*, pParent */)
    , m_Listbox_Types_control()
{
    this->m_listbox_types_strings = types_string;
    this->m_Combo_Types_colour_var = colour;
}
CSmartColourTypes::~CSmartColourTypes()
{
}

void CSmartColourTypes::DoDataExchange(CDataExchange* pDX)
{
    CMFCPropertyPage::DoDataExchange(pDX);

    DDX_Control(pDX, IDC_MFCVSLISTBOX_TYPES, m_Listbox_Types_control);
    DDX_Control(pDX, IDC_COMBO_COLOUR_TYPES, m_Combo_Types_colour_control);
}

BOOL CSmartColourTypes::OnInitDialog()
{
    CDialog::OnInitDialog();

    for (auto& it : this->m_listbox_types_strings) {
        this->m_Listbox_Types_control.AddItem(it);
    }
    this->m_Combo_Types_colour_control.SetColor(this->m_Combo_Types_colour_var);

    return 0;
}

BOOL CSmartColourTypes::OnKillActive()
{
    this->m_listbox_types_strings.resize(this->m_Listbox_Types_control.GetCount());

    for (int i = 0; i < this->m_Listbox_Types_control.GetCount(); i++) {
        this->m_listbox_types_strings[i] = this-
```

```
            >m_Listbox_Types_control.GetItemText(i);
    }

    // Insertion Sort
    for (int i = 1; i < this->m_listbox_types_strings.size(); i++) {

        CString currentValue = this->m_listbox_types_strings[i];
        int pos = i;

        while (pos > 0 && this->m_listbox_types_strings[pos - 1] > currentValue) {
            this->m_listbox_types_strings[pos] = this->m_listbox_types_strings[pos - 1];
            pos--;
        }
        this->m_listbox_types_strings[pos] = currentValue;
    }

    this->m_Combo_Types_colour_var = this->m_Combo_Types_colour_control.GetColor();

    return CMFCPropertyPage::OnKillActive();
}


BEGIN_MESSAGE_MAP(CSmartColourTypes, CMFCPropertyPage)
END_MESSAGE_MAP()

```

```cpp
#pragma once
#include "afxdialogex.h"


// CSmartColourBuiltin dialog

class CSmartColourBuiltin : public CMFCPropertyPage
{

public:
    CSmartColourBuiltin(std::vector<CString> classes_string, COLORREF classes_colour, std::vector<CString> functions_string, COLORREF functions_colour, CWnd* pParent = nullptr);   // standard constructor
    virtual ~CSmartColourBuiltin();

// Dialog Data
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_SMARTCOLOUR_BUILTIN };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnInitDialog();
    virtual BOOL OnKillActive();

    DECLARE_DYNAMIC(CSmartColourBuiltin)
    DECLARE_MESSAGE_MAP()
public:

    // Controls
    CVSListBox m_Listbox_Builtin_Classes_control;
    CMFCColorButton m_Combo_Builtin_Classes_colour_control;

    CVSListBox m_Listbox_Builtin_Functions_control;
    CMFCColorButton m_Combo_Builtin_Functions_colour_control;

    // Variables
    std::vector<CString> m_Listbox_Builtin_Classes_var;
    COLORREF m_Combo_Builtin_Classes_colour_var;

    std::vector<CString> m_Listbox_Builtin_Functions_var;
    COLORREF m_Combo_Builtin_Functions_colour_var;
};

```

```cpp
// CSmartColourBuiltin.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"
#include "afxdialogex.h"
#include "CSmartColourBuiltin.h"


// CSmartColourBuiltin dialog

IMPLEMENT_DYNAMIC(CSmartColourBuiltin, CMFCPropertyPage)

CSmartColourBuiltin::CSmartColourBuiltin(std::vector<CString> classes_string, COLORREF classes_colour, std::vector<CString> functions_string, COLORREF functions_colour, CWnd* pParent /*=nullptr*/)
    : CMFCPropertyPage(IDD_SMARTCOLOUR_BUILTIN/*, pParent*/)
{
    this->m_Listbox_Builtin_Classes_var = classes_string;
    this->m_Combo_Builtin_Classes_colour_var = classes_colour;

    this->m_Listbox_Builtin_Functions_var = functions_string;
    this->m_Combo_Builtin_Functions_colour_var = functions_colour;
}

CSmartColourBuiltin::~CSmartColourBuiltin()
{
}

void CSmartColourBuiltin::DoDataExchange(CDataExchange* pDX)
{
    CMFCPropertyPage::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_MFCVSLISTBOX_BUILTIN_CLASSES, m_Listbox_Builtin_Classes_control);
    DDX_Control(pDX, IDC_COMBO_COLOUR_BUILTIN_CLASSES, m_Combo_Builtin_Classes_colour_control);
    DDX_Control(pDX, IDC_MFCVSLISTBOX_BUILTIN_FUNCTIONS, m_Listbox_Builtin_Functions_control);
    DDX_Control(pDX, IDC_COMBO_COLOUR_BUILTIN_FUNCTIONS, m_Combo_Builtin_Functions_colour_control);
}

BOOL CSmartColourBuiltin::OnInitDialog()
{
    CDialog::OnInitDialog();

    for (auto& it : this->m_Listbox_Builtin_Classes_var) {
        this->m_Listbox_Builtin_Classes_control.AddItem(it);
    }
    this->m_Combo_Builtin_Classes_colour_control.SetColor(this->m_Combo_Builtin_Classes_colour_var);

    for (auto& it : this->m_Listbox_Builtin_Functions_var) {
        this->m_Listbox_Builtin_Functions_control.AddItem(it);
    }
```

```
49      this->m_Combo_Builtin_Functions_colour_control.SetColor(this->m_Combo_Builtin_Functions_colour_var);
50
51      return 0;
52  }
53
54  BOOL CSmartColourBuiltin::OnKillActive()
55  {
56      // Classes
57      this->m_Listbox_Builtin_Classes_var.resize(this->m_Listbox_Builtin_Classes_control.GetCount());
58
59      for (int i = 0; i < this->m_Listbox_Builtin_Classes_control.GetCount(); i++) {
60          this->m_Listbox_Builtin_Classes_var[i] = this->m_Listbox_Builtin_Classes_control.GetItemText(i);
61      }
62
63      // Insertion Sort
64      for (int i = 1; i < this->m_Listbox_Builtin_Classes_var.size(); i++) {
65          CString currentValue = this->m_Listbox_Builtin_Classes_var[i];
66          int pos = i;
67
68          while (pos > 0 && this->m_Listbox_Builtin_Classes_var[pos - 1] > currentValue) {
69              this->m_Listbox_Builtin_Classes_var[pos] = this->m_Listbox_Builtin_Classes_var[pos - 1];
70              pos--;
71          }
72          this->m_Listbox_Builtin_Classes_var[pos] = currentValue;
73      }
74
75      this->m_Combo_Builtin_Classes_colour_var = this->m_Combo_Builtin_Classes_colour_control.GetColor();
76
77      //Functions
78      this->m_Listbox_Builtin_Functions_var.resize(this->m_Listbox_Builtin_Functions_control.GetCount());
79
80      for (int i = 0; i < this->m_Listbox_Builtin_Functions_control.GetCount(); i++) {
81          this->m_Listbox_Builtin_Functions_var[i] = this->m_Listbox_Builtin_Functions_control.GetItemText(i);
82      }
83
84      // Insertion Sort
85      for (int i = 1; i < this->m_Listbox_Builtin_Functions_var.size(); i++) {
86          CString currentValue = this->m_Listbox_Builtin_Functions_var[i];
87          int pos = i;
88
89          while (pos > 0 && this->m_Listbox_Builtin_Functions_var[pos - 1] > currentValue) {
```

```
90              this->m_Listbox_Builtin_Functions_var[pos] = this-
                  >m_Listbox_Builtin_Functions_var[pos - 1];
91              pos--;
92          }
93          this->m_Listbox_Builtin_Functions_var[pos] = currentValue;
94      }
95
96      this->m_Combo_Builtin_Functions_colour_var = this-
          >m_Combo_Builtin_Functions_colour_control.GetColor();
97
98
99      return CMFCPropertyPage::OnKillActive();
100 }
101
102
103 BEGIN_MESSAGE_MAP(CSmartColourBuiltin, CMFCPropertyPage)
104 END_MESSAGE_MAP()
105
106
107 // CSmartColourBuiltin message handlers
108
```

```h
#pragma once
#include "afxdialogex.h"


// CSmartColourOther dialog

class CSmartColourOther : public CMFCPropertyPage
{

public:
    CSmartColourOther(CString comments_type, COLORREF comments_colour, COLORREF numbers_colour, COLORREF highlight_colour, CString fonts, CString indentSize, CWnd* pParent = nullptr);   // standard constructor
    virtual ~CSmartColourOther();

// Dialog Data
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_SMARTCOLOUR_OTHER };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnInitDialog();
    virtual BOOL OnKillActive();

    DECLARE_DYNAMIC(CSmartColourOther)
    DECLARE_MESSAGE_MAP()
public:

    // Controls
    CComboBox m_pCombo_comments_type_control;
    CMFCColorButton m_pCombo_comments_colour_control;
    CMFCColorButton m_pCombo_numbers_colour_control;
    CMFCColorButton m_pCombo_highlight_colour_control;
    CComboBox m_pCombo_fonts_control;
    CEdit m_pCombo_indent_control;

    // Variables
    CString m_pCombo_comments_type_var;
    COLORREF m_pCombo_comments_colour_var;
    COLORREF m_pCombo_numbers_colour_var;
    COLORREF m_pCombo_highlight_colour_var;
    CString m_pCombo_fonts_var;
    CString m_pCombo_indent_var;
};

```

```cpp
// CSmartColourOther.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"
#include "afxdialogex.h"
#include "CSmartColourOther.h"


// CSmartColourOther dialog

IMPLEMENT_DYNAMIC(CSmartColourOther, CMFCPropertyPage)

CSmartColourOther::CSmartColourOther(CString comments_type, COLORREF comments_colour, COLORREF numbers_colour, COLORREF highlight_colour, CString fonts, CString indentSize, CWnd* pParent /*=nullptr*/)
    : CMFCPropertyPage(IDD_SMARTCOLOUR_OTHER/*, pParent*/)
{
    this->m_pCombo_comments_type_var = comments_type;
    this->m_pCombo_comments_colour_var = comments_colour;
    this->m_pCombo_numbers_colour_var = numbers_colour;
    this->m_pCombo_highlight_colour_var = highlight_colour;
    this->m_pCombo_fonts_var = fonts;
    this->m_pCombo_indent_var = indentSize;
}

CSmartColourOther::~CSmartColourOther()
{
}

void CSmartColourOther::DoDataExchange(CDataExchange* pDX)
{
    CMFCPropertyPage::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_SMARTCOLOUR_OTHERS_COMMENT_COMBO, m_pCombo_comments_type_control);
    DDX_Control(pDX, IDC_SMARTCOLOUR_OTHER_COMMENTS_COLOUR, m_pCombo_comments_colour_control);
    DDX_Control(pDX, IDC_SMARTCOLOUR_OTHERS_NUMBERS_COLOUR, m_pCombo_numbers_colour_control);
    DDX_Control(pDX, IDC_SMARTCOLOUR_OTHERS_NUMBERS_COLOUR2, m_pCombo_highlight_colour_control);
    DDX_Control(pDX, IDC_SMARTCOLOUR_OTHERS_FONT, m_pCombo_fonts_control);
    DDX_Control(pDX, IDC_SMARTCOLOR_OTHER_INDENT, m_pCombo_indent_control);
}
BOOL CSmartColourOther::OnInitDialog()
{
    CDialog::OnInitDialog();

    this->m_pCombo_comments_type_control.SetCurSel(this->m_pCombo_comments_type_control.FindString(-1, this->m_pCombo_comments_type_var));
    this->m_pCombo_comments_colour_control.SetColor(this->m_pCombo_comments_colour_var);
```

```
45
46      this->m_pCombo_numbers_colour_control.SetColor(this->m_pCombo_numbers_colour_var);
47      this->m_pCombo_highlight_colour_control.SetColor(this->m_pCombo_highlight_colour_var);
48
49      this->m_pCombo_fonts_control.SetCurSel(this->m_pCombo_fonts_control.FindString(-1, this->m_pCombo_fonts_var));
50      this->m_pCombo_indent_control.SetWindowTextW(this->m_pCombo_indent_var);
51
52      return 0;
53  }
54  BOOL CSmartColourOther::OnKillActive()
55  {
56      this->m_pCombo_comments_type_control.GetLBText(this->m_pCombo_comments_type_control.GetCurSel(), this->m_pCombo_comments_type_var);
57      this->m_pCombo_comments_colour_var = this->m_pCombo_comments_colour_control.GetColor();
58
59      this->m_pCombo_numbers_colour_var = this->m_pCombo_numbers_colour_control.GetColor();
60      this->m_pCombo_highlight_colour_var = this->m_pCombo_highlight_colour_control.GetColor();
61
62      this->m_pCombo_fonts_control.GetLBText(this->m_pCombo_fonts_control.GetCurSel(), this->m_pCombo_fonts_var);
63      this->m_pCombo_indent_control.GetWindowTextW(this->m_pCombo_indent_var);
64
65      return CMFCPropertyPage::OnKillActive();
66  }
67
68
69  BEGIN_MESSAGE_MAP(CSmartColourOther, CMFCPropertyPage)
70  END_MESSAGE_MAP()
```

```h
#pragma once
#include "afxdialogex.h"


// CAboutDlg dialog

class CAboutDlg : public CDialogEx
{

public:
    CAboutDlg(CWnd* pParent = nullptr);   // standard constructor
    virtual ~CAboutDlg();

// Dialog Data
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_APP_ABOUT };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

    DECLARE_DYNAMIC(CAboutDlg)
    DECLARE_MESSAGE_MAP()
};

```

```cpp
// CAboutDlg.cpp : implementation file
//

#include "pch.h"
#include "DesignArk.h"
#include "afxdialogex.h"
#include "CAboutDlg.h"


// CAboutDlg dialog

IMPLEMENT_DYNAMIC(CAboutDlg, CDialogEx)

CAboutDlg::CAboutDlg(CWnd* pParent /*=nullptr*/)
    : CDialogEx(IDD_APP_ABOUT, pParent)
{

}

CAboutDlg::~CAboutDlg()
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()


// CAboutDlg message handlers

```