Customization update

2023년 5월 27일 토요일 오전 6:09

1. remove EngineDump project
2. remove SumatraPdf-dll project

file	before and after		
	function	before	after

pdf-annot.c	**make the text red and reduce font size to 9**	pdf_set_annot_default_appearance(ctx, annot, "Helv", 12, nelem(black), black);	//float CMYK[] = {0, 0.5, 0.3, 0};
			//pdf_set_annot_default_appearance(ctx, annot, "Helv", 9, 4, CMYK);
			pdf_set_annot_default_appearance(ctx, annot, "Helv", 9, nelem(red), red);

	Prevent Image annot from being cleared	void
		pdf_dirty_annot(fz_context *ctx, pdf_annot *annot)
		{
		pdf_annot_request_resynthesis(ctx, annot);
		}
			void
			pdf_dirty_annot(fz_context *ctx, pdf_annot *annot)
			{
			enum pdf_annot_type ret = pdf_annot_type(ctx, annot);
			if (ret != PDF_ANNOT_IMAGE)
			pdf_annot_request_resynthesis(ctx, annot);
			}

	insert Bbox and image type annotation	const char *
		pdf_string_from_annot_type(fz_context *ctx, enum pdf_annot_type type)
		…
		…
		…
		case PDF_ANNOT_REDACT: return "Redact";
		case PDF_ANNOT_STAMP: return "Stamp";
		case PDF_ANNOT_CARET: return "Caret";
			const char *
			pdf_string_from_annot_type(fz_context *ctx, enum pdf_annot_type type)
			…
			…
			…
			case PDF_ANNOT_REDACT: return "Redact";
			case PDF_ANNOT_BBOX: return "BBox";
			case PDF_ANNOT_STAMP: return "Stamp";
			case PDF_ANNOT_CARET: return "Caret";
			case PDF_ANNOT_IMAGE: return "Image";

	insert Bbox and image type annotation	int
		pdf_annot_type_from_string(fz_context *ctx, const char *subtype)
		{
		…
		…
		…
		if (!strcmp("Redact", subtype)) return PDF_ANNOT_REDACT;
		if (!strcmp("Stamp", subtype)) return PDF_ANNOT_STAMP;
		if (!strcmp("Caret", subtype)) return PDF_ANNOT_CARET;
			int
			pdf_annot_type_from_string(fz_context *ctx, const char *subtype)
			{
			…
			…
			…
			if (!strcmp("Redact", subtype)) return PDF_ANNOT_REDACT;
			if (!strcmp("BBox", subtype)) return PDF_ANNOT_BBOX;
			if (!strcmp("Stamp", subtype)) return PDF_ANNOT_STAMP;
			if (!strcmp("Caret", subtype)) return PDF_ANNOT_CARET;
			if (!strcmp("Image", subtype)) return PDF_ANNOT_IMAGE;

	**1. set rect of image annotation
	2. Change to a transparent border for image object**
		case PDF_ANNOT_CARET:
		{
		fz_rect caret_rect = { 12, 12, 12+18, 12+15 };
		pdf_set_annot_rect(ctx, annot, caret_rect);
		pdf_set_annot_color(ctx, annot, 3, blue);
		}
		break;
			case PDF_ANNOT_CARET:
			{
			fz_rect caret_rect = {12, 12, 12 + 18, 12 + 15};
			pdf_set_annot_rect(ctx, annot, caret_rect);
			pdf_set_annot_color(ctx, annot, 3, blue);
			}
			break;
			case PDF_ANNOT_IMAGE:
			{
			fz_rect image_rect = {12, 12, 12 + 200, 12 + 150};
			pdf_set_annot_rect(ctx, annot, image_rect);
			float transparent[] = {0, 0, 0, 0};
			pdf_set_annot_color(ctx, annot, 4, transparent);
			}
			break;

	set subtype of Bbox and image rect annotation	static pdf_obj *rect_subtypes[] = {
		PDF_NAME(Text),
		PDF_NAME(FreeText),
		PDF_NAME(Square),
		PDF_NAME(Circle),
		PDF_NAME(Redact),
		PDF_NAME(Stamp),
		PDF_NAME(Caret),
		PDF_NAME(Popup),
		PDF_NAME(FileAttachment),
		PDF_NAME(Sound),
		PDF_NAME(Movie),
		PDF_NAME(Widget),
		NULL,
		};
			static pdf_obj *rect_subtypes[] = {
			PDF_NAME(Text),
			PDF_NAME(FreeText),
			PDF_NAME(Square),
			PDF_NAME(Circle),
			PDF_NAME(Redact),
			PDF_NAME(BBox),
			PDF_NAME(Stamp),
			PDF_NAME(Caret),
			PDF_NAME(Image),
			PDF_NAME(Popup),
			PDF_NAME(FileAttachment),
			PDF_NAME(Sound),
			PDF_NAME(Movie),
			PDF_NAME(Widget),
			NULL,
			};

	set subtype of Bbox and image quad point annotation	static pdf_obj *quad_point_subtypes[] = {
		PDF_NAME(Highlight),
		PDF_NAME(Link),
		PDF_NAME(Squiggly),
		PDF_NAME(StrikeOut),
		PDF_NAME(Underline),
		PDF_NAME(Redact),
		NULL,
		};
			static pdf_obj *quad_point_subtypes[] = {
			PDF_NAME(Highlight),
			PDF_NAME(Link),
			PDF_NAME(Squiggly),
			PDF_NAME(StrikeOut),
			PDF_NAME(Underline),
			PDF_NAME(Redact),
			PDF_NAME(BBox),
			NULL,

			};

	set subtype of Bbox and image markup annotation	static pdf_obj *markup_subtypes[] = {
		PDF_NAME(Text),
		PDF_NAME(FreeText),
		PDF_NAME(Line),
		PDF_NAME(Square),
		PDF_NAME(Circle),
		PDF_NAME(Polygon),
		PDF_NAME(PolyLine),
		PDF_NAME(Highlight),
		PDF_NAME(Underline),
		PDF_NAME(Squiggly),
		PDF_NAME(StrikeOut),
		PDF_NAME(Redact),
		PDF_NAME(Stamp),
		PDF_NAME(Caret),
		
		PDF_NAME(Ink),
		PDF_NAME(FileAttachment),
		PDF_NAME(Sound),
		NULL,
		};
			static pdf_obj *markup_subtypes[] = {
			PDF_NAME(Text),
			PDF_NAME(FreeText),
			PDF_NAME(Line),
			PDF_NAME(Square),
			PDF_NAME(Circle),
			PDF_NAME(Polygon),
			PDF_NAME(PolyLine),
			PDF_NAME(Highlight),
			PDF_NAME(Underline),
			PDF_NAME(Squiggly),
			PDF_NAME(StrikeOut),
			PDF_NAME(Redact),
			PDF_NAME(BBox),
			PDF_NAME(Stamp),
			PDF_NAME(Caret),
			PDF_NAME(Image),
			PDF_NAME(Ink),
			PDF_NAME(FileAttachment),
			PDF_NAME(Sound),
			NULL,
			};

EditAnnotation.cpp	function	before	after

	include iostream and fstream	-	#include <iostream>
			#include <fstream>

	1. Force focus to input window when creating a comment
	2. Automatically select entire text
		static void DoContents(EditAnnotationsWindow* ew, Annotation* annot) {
		str::Str s = Contents(annot);
		// TODO: don't replace if already is "\r\n"
		Replace(s, "\n", "\r\n");
		ew->editContents->SetText(s.Get());
		ew->staticContents->SetIsVisible(true);
		ew->editContents->SetIsVisible(true);
		}
			static void DoContents(EditAnnotationsWindow* ew, Annotation* annot) {
			str::Str s = Contents(annot);
			// TODO: don't replace if already is "\r\n"
			Replace(s, "\n", "\r\n");
			ew->editContents->SetText(s.Get());
			keybd_event(VK_CONTROL, 0, 0, 0); // push Ctrl key
			keybd_event('A', 0, 0, 0); // push 'A' key
			keybd_event('A', 0, KEYEVENTF_KEYUP, 0); // release A key
			keybd_event(VK_CONTROL, 0, KEYEVENTF_KEYUP, 0); // release Ctrl key
			ew->staticContents->SetIsVisible(true);
			ew->editContents->SetIsVisible(true);
			SetFocus(ew->editContents->hwnd);
			}

	Remove timer object	static UINT_PTR gMainWindowRerenderTimer = 0;
		static MainWindow* gMainWindowForRender = nullptr;
		
		// TODO: there seems to be a leak
		static void ContentsChanged(EditAnnotationsWindow* ew) {
		auto txt = ew->editContents->GetTextTemp();
		SetContents(ew->annot, txt);
		EnableSaveIfAnnotationsChanged(ew);
		
		MainWindow* win = ew->tab->win;
		if (gMainWindowRerenderTimer != 0) {
		// logf("ContentsChanged: killing existing timer for re-render of MainWindow\n");
		KillTimer(win->hwndCanvas, gMainWindowRerenderTimer);
		gMainWindowRerenderTimer = 0;
		}
		UINT timeoutInMs = 75;
		gMainWindowForRender = win;
		if (MainWindowStillValid(gMainWindowForRender)) {
		gMainWindowRerenderTimer = SetTimer(win->hwndCanvas, 1, timeoutInMs, [](HWND, UINT, UINT_PTR, DWORD) {
		// logf("ContentsChanged: re-rendering MainWindow\n");
		MainWindowRerender(gMainWindowForRender);
		});
		} else {
		// logf("ContentsChanged: NOT re-rendering MainWindow because is not valid anymore\n");
		}
		}
			static MainWindow* gMainWindowForRender = nullptr;
			// TODO: there seems to be a leak
			static void ContentsChanged(EditAnnotationsWindow* ew) {
			auto txt = ew->editContents->GetTextTemp();
			SetContents(ew->annot, txt);
			EnableSaveIfAnnotationsChanged(ew);
			
			MainWindow* win = ew->tab->win;
			gMainWindowForRender = win;
			if (MainWindowStillValid(gMainWindowForRender)) {
			MainWindowRerender(gMainWindowForRender, true);
			}
			}

	Set selection of list box to the last comment after deleting a comment.	void DeleteAnnotationAndUpdateUI(WindowTab* tab, EditAnnotationsWindow* ew, Annotation* annot) {
		annot = FindMatchingAnnotation(ew, annot);
		DeleteAnnotation(annot);
		if (ew != nullptr) {
		// can be null if called from Menu.cpp and annotations window is not visible
		RebuildAnnotations(ew);
		UpdateUIForSelectedAnnotation(ew, 0);
		ew->listBox->SetCurrentSelection(0);
		}
		MainWindowRerender(tab->win);
		ToolbarUpdateStateForWindow(tab->win, false);
		}
			void DeleteAnnotationAndUpdateUI(WindowTab* tab, EditAnnotationsWindow* ew, Annotation* annot) {
			annot = FindMatchingAnnotation(ew, annot);
			DeleteAnnotation(annot);
			if (ew != nullptr) {
			// can be null if called from Menu.cpp and annotations window is not visible
			RebuildAnnotations(ew);
			int iC = ew->listBox->GetCount()-1;
			if (iC>=0) {
			UpdateUIForSelectedAnnotation(ew, iC);
			ew->listBox->SetCurrentSelection(iC);
			}
			}
			MainWindowRerender(tab->win);
			ToolbarUpdateStateForWindow(tab->win, false);

			}

	1. Set default text content as "Text"
	2. Remove free text border
	3. Copy and paste an image file into a PDF page
		Annotation* EngineMupdfCreateAnnotation(EngineBase* engine, AnnotationType typ, int pageNo, PointF pos) {
		    EngineMupdf* epdf = AsEngineMupdf(engine);
		    fz_context* ctx = epdf->ctx;
		
		    auto pageInfo = epdf->GetFzPageInfo(pageNo, true);
		
		    ScopedCritSec cs(epdf->ctxAccess);
		
		    auto page = pdf_page_from_fz_page(ctx, pageInfo->page);
		    enum pdf_annot_type atyp = (enum pdf_annot_type)typ;
		
		    auto annot = pdf_create_annot(ctx, page, atyp);
		
		    pdf_set_annot_modification_date(ctx, annot, time(nullptr));
		    if (pdf_annot_has_author(ctx, annot)) {
		        char* defAuthor = gGlobalPrefs->annotations.defaultAuthor;
		        // if "(none)" we don't set it
		        if (!str::Eq(defAuthor, "(none)")) {
		            const char* author = getuser();
		            if (!str::EmptyOrWhiteSpaceOnly(defAuthor)) {
		                author = defAuthor;
		            }
		            pdf_set_annot_author(ctx, annot, author);
		        }
		    }
		
		    switch (typ) {
		        case AnnotationType::Text:
		        case AnnotationType::FreeText:
		        case AnnotationType::Stamp:
		        case AnnotationType::Caret:
		        case AnnotationType::Square:
		        case AnnotationType::Circle: {
		            fz_rect trect = pdf_annot_rect(ctx, annot);
		            float dx = trect.x1 - trect.x0;
		            trect.x0 = pos.x;
		            trect.x1 = trect.x0 + dx;
		            float dy = trect.y1 - trect.y0;
		            trect.y0 = pos.y;
		            trect.y1 = trect.y0 + dy;
		            pdf_set_annot_rect(ctx, annot, trect);
		        } break;
		        case AnnotationType::Line: {
		            fz_point a{pos.x, pos.y};
		            fz_point b{pos.x + 100, pos.y + 50};
		            pdf_set_annot_line(ctx, annot, a, b);
		        } break;
		    }
		    if (typ == AnnotationType::FreeText) {
		        pdf_set_annot_contents(ctx, annot, "This is a text...");
		        pdf_set_annot_border(ctx, annot, 1);
		    }
		
		    pdf_update_annot(ctx, annot);
		    auto res = MakeAnnotationPdf(epdf, annot, pageNo);
		    if (typ == AnnotationType::Text) {
		        AutoFreeStr iconName = GetAnnotationTextIcon();
		        if (!str::EqI(iconName, "Note")) {
		            SetIconName(res, iconName.Get());
		        }
		        auto col = GetAnnotationTextIconColor();
		        SetColor(res, col);
		    } else if (typ == AnnotationType::Underline) {
		        auto col = GetAnnotationUnderlineColor();
		        SetColor(res, col);
		    } else if (typ == AnnotationType::Highlight) {
		        auto col = GetAnnotationHighlightColor();
		        SetColor(res, col);
		    } else if (typ == AnnotationType::Squiggly) {
		        auto col = GetAnnotationSquigglyColor();
		        SetColor(res, col);
		    } else if (typ == AnnotationType::StrikeOut) {
		        auto col = GetAnnotationStrikeOutColor();
		        SetColor(res, col);
		    }
		    pdf_drop_annot(ctx, annot);
		    return res;
		}
			Annotation* EngineMupdfCreateAnnotation(EngineBase* engine, AnnotationType typ, int pageNo, PointF pos) {
			    if (typ == AnnotationType::Image) {
			        // Open the clipboard, and verify that the image data is there.
			        if (!OpenClipboard(nullptr))
			            return NULL;
			        if (!IsClipboardFormatAvailable(CF_BITMAP)) {
			            CloseClipboard();
			            return NULL;
			        }
			    }
			    EngineMupdf* epdf = AsEngineMupdf(engine);
			    fz_context* ctx = epdf->ctx;
			
			    auto pageInfo = epdf->GetFzPageInfo(pageNo, true);
			
			    ScopedCritSec cs(epdf->ctxAccess);
			
			    auto page = pdf_page_from_fz_page(ctx, pageInfo->page);
			    enum pdf_annot_type atyp = (enum pdf_annot_type)typ;
			
			    auto annot = pdf_create_annot(ctx, page, atyp);
			
			    pdf_set_annot_modification_date(ctx, annot, time(nullptr));
			    if (pdf_annot_has_author(ctx, annot)) {
			        char* defAuthor = gGlobalPrefs->annotations.defaultAuthor;
			        // if "(none)" we don't set it
			        if (!str::Eq(defAuthor, "(none)")) {
			            const char* author = getuser();
			            if (!str::EmptyOrWhiteSpaceOnly(defAuthor)) {
			                author = defAuthor;
			            }
			            pdf_set_annot_author(ctx, annot, author);
			        }
			    }
			
			    switch (typ) {
			        case AnnotationType::Text:
			        case AnnotationType::FreeText:
			            break;
			        case AnnotationType::Stamp:
			        case AnnotationType::Caret:
			        case AnnotationType::Image:
			        case AnnotationType::Square:
			        case AnnotationType::Circle: {
			            fz_rect trect = pdf_annot_rect(ctx, annot);
			            float dx = trect.x1 - trect.x0;
			            trect.x0 = pos.x;
			            trect.x1 = trect.x0 + dx;
			            float dy = trect.y1 - trect.y0;
			            trect.y0 = pos.y;
			            trect.y1 = trect.y0 + dy;
			            pdf_set_annot_rect(ctx, annot, trect);
			        } break;
			        case AnnotationType::Line: {
			            fz_point a{pos.x, pos.y};
			            fz_point b{pos.x + 100, pos.y + 50};
			            pdf_set_annot_line(ctx, annot, a, b);
			        } break;
			    }
			    if (typ == AnnotationType::FreeText) {
			        pdf_set_annot_contents(ctx, annot, "Text");
			        pdf_set_annot_border(ctx, annot, 0);
			        fz_rect trect = pdf_annot_rect(ctx, annot);
			        trect.x0 = pos.x;
			        trect.y0 = pos.y + 10;
			        trect.x1 = pos.x;
			        trect.y1 = pos.y + 10;
			        pdf_set_annot_rect(ctx, annot, trect);
			    }
			
			    pdf_update_annot(ctx, annot);
			    auto res = MakeAnnotationPdf(epdf, annot, pageNo);
			    if (typ == AnnotationType::Text) {
			        AutoFreeStr iconName = GetAnnotationTextIcon();
			        if (!str::EqI(iconName, "Note")) {
			            SetIconName(res, iconName.Get());
			        }
			        auto col = GetAnnotationTextIconColor();
			        SetColor(res, col);
			    } else if (typ == AnnotationType::Underline) {
			        auto col = GetAnnotationUnderlineColor();
			        SetColor(res, col);
			    } else if (typ == AnnotationType::Highlight) {
			        auto col = GetAnnotationHighlightColor();
			        SetColor(res, col);
			    } else if (typ == AnnotationType::Squiggly) {
			        auto col = GetAnnotationSquigglyColor();
			        SetColor(res, col);

			} else if (typ == AnnotationType::StrikeOut) {
			auto col = GetAnnotationStrikeOutColor();
			SetColor(res, col);
			}
			pdf_drop_annot(ctx, annot);
			if (typ == AnnotationType::Image) {
			// Retrieve the bitmap handle from the clipboard.
			HBITMAP hBitmap = static_cast<HBITMAP>(GetClipboardData(CF_BITMAP));
			if (hBitmap == nullptr) {
			CloseClipboard();
			return NULL;
			}
			// Extract DIB data from a bitmap handle.
			BITMAP bm;
			GetObject(hBitmap, sizeof(BITMAP), &bm);
			int size = bm.bmWidthBytes * bm.bmHeight;
			unsigned char* data = new unsigned char[size];
			GetBitmapBits(hBitmap, size, data);
			
			// Write the extracted DIB data to a file.
			std::ofstream file("clipboard_image.bmp", std::ios::binary);
			if (!file.is_open()) {
			delete[] data;
			CloseClipboard();
			return NULL;
			}
			BITMAPFILEHEADER bmfh = {0};
			bmfh.bfType = 0x4d42; // "BM"
			bmfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
			bmfh.bfSize = bmfh.bfOffBits + size;
			file.write(reinterpret_cast<const char*>(&bmfh), sizeof(bmfh));
			BITMAPINFOHEADER bmih = {0};
			bmih.biSize = sizeof(BITMAPINFOHEADER);
			bmih.biWidth = bm.bmWidth;
			bmih.biHeight = bm.bmHeight; // Save top-down method
			bmih.biPlanes = 1;
			bmih.biBitCount = bm.bmBitsPixel;
			bmih.biCompression = BI_RGB;
			bmih.biSizeImage = size;
			file.write(reinterpret_cast<const char*>(&bmih), sizeof(bmih));
			for (int y = bm.bmHeight - 1; y >= 0; --y) {
			file.write(reinterpret_cast<const char*>(data + y * bm.bmWidthBytes), bm.bmWidthBytes);
			}
			file.close();
			// Clean up unused handles and data.
			delete[] data;
			CloseClipboard();
			// Attaches a clipboard image to the stamp. Stamp functionality implemented in Image
			fz_image *img = fz_new_image_from_file(ctx, "clipboard_image.bmp");
			pdf_set_annot_stamp_image(ctx, annot, img);
			fz_drop_image(ctx, img);
			}
			return res;
			}

	add image to annotation type	static AnnotationType gAnnotsWithColor[] = {
		AnnotationType::Stamp, AnnotationType::Text, AnnotationType::FileAttachment,
		AnnotationType::Sound, AnnotationType::Caret, AnnotationType::FreeText,
		AnnotationType::Ink, AnnotationType::Line, AnnotationType::Square,
		AnnotationType::Circle, AnnotationType::Polygon, AnnotationType::PolyLine,
		AnnotationType::Highlight, AnnotationType::Underline, AnnotationType::StrikeOut,
		AnnotationType::Squiggly,
		};
			static AnnotationType gAnnotsWithColor[] = {
			AnnotationType::Stamp, AnnotationType::Text, AnnotationType::FileAttachment,
			AnnotationType::Sound, AnnotationType::Caret, AnnotationType::Image, AnnotationType::FreeText,
			AnnotationType::Ink, AnnotationType::Line, AnnotationType::Square,
			AnnotationType::Circle, AnnotationType::Polygon, AnnotationType::PolyLine,
			AnnotationType::Highlight, AnnotationType::Underline, AnnotationType::StrikeOut,
			AnnotationType::Squiggly,
			};

	Declaring clipboard image Trackbar and Track Position Objects	struct EditAnnotationsWindow : Wnd {
		void OnSize(UINT msg, UINT type, SIZE size) override;
		void OnClose() override;
		
		WindowTab* tab = nullptr;
		LayoutBase* mainLayout = nullptr;
		
		ListBox* listBox = nullptr;
		Static* staticRect = nullptr;
		Static* staticAuthor = nullptr;
		Static* staticModificationDate = nullptr;
		Static* staticPopup = nullptr;
		Static* staticContents = nullptr;
		Edit* editContents = nullptr;
		Static* staticTextAlignment = nullptr;
		DropDown* dropDownTextAlignment = nullptr;
		Static* staticTextFont = nullptr;
		DropDown* dropDownTextFont = nullptr;
			struct EditAnnotationsWindow : Wnd {
			void OnSize(UINT msg, UINT type, SIZE size) override;
			void OnClose() override;
			
			WindowTab* tab = nullptr;
			LayoutBase* mainLayout = nullptr;
			
			ListBox* listBox = nullptr;
			Static* staticRect = nullptr;
			Static* staticAuthor = nullptr;
			Static* staticModificationDate = nullptr;
			Static* staticPopup = nullptr;
			Static* staticContents = nullptr;
			Edit* editContents = nullptr;
			Static* staticTextAlignment = nullptr;
			DropDown* dropDownTextAlignment = nullptr;
			Static* staticTextFont = nullptr;
			DropDown* dropDownTextFont = nullptr;

	Static* staticTextSize = nullptr;
	Trackbar* trackbarTextSize = nullptr;
		Static* staticTextSize = nullptr;
		Trackbar* trackbarTextSize = nullptr;
		Static* staticImageSize = nullptr;
		Trackbar* trackbarImageSize = nullptr;

Make clipboard image trackbar and track position objects visible	static void HidePerAnnotControls(EditAnnotationsWindow* ew) {
	ew->staticRect->SetIsVisible(false);
	ew->staticAuthor->SetIsVisible(false);
	ew->staticModificationDate->SetIsVisible(false);
	ew->staticPopup->SetIsVisible(false);
	ew->staticContents->SetIsVisible(false);
	ew->editContents->SetIsVisible(false);
	ew->staticTextAlignment->SetIsVisible(false);
	ew->dropDownTextAlignment->SetIsVisible(false);
	ew->staticTextFont->SetIsVisible(false);
	ew->dropDownTextFont->SetIsVisible(false);
	ew->staticTextSize->SetIsVisible(false);
	ew->trackbarTextSize->SetIsVisible(false);
		static void HidePerAnnotControls(EditAnnotationsWindow* ew) {
		ew->staticRect->SetIsVisible(false);
		ew->staticAuthor->SetIsVisible(false);
		ew->staticModificationDate->SetIsVisible(false);
		ew->staticPopup->SetIsVisible(false);
		ew->staticContents->SetIsVisible(false);
		ew->editContents->SetIsVisible(false);
		ew->staticTextAlignment->SetIsVisible(false);
		ew->dropDownTextAlignment->SetIsVisible(false);
		ew->staticTextFont->SetIsVisible(false);
		ew->dropDownTextFont->SetIsVisible(false);
		ew->staticTextSize->SetIsVisible(false);
		ew->trackbarTextSize->SetIsVisible(false);
		ew->staticImageSize->SetIsVisible(false);
		ew->trackbarImageSize->SetIsVisible(false);

Initialize cliboard image Trackbar command	HidePerAnnotControls(ew);
	if (ew->annot) {
	DoRect(ew, ew->annot);
	DoAuthor(ew, ew->annot);
	DoModificationDate(ew, ew->annot);
	DoPopup(ew, ew->annot);
	DoContents(ew, ew->annot);
	
	DoTextAlignment(ew, ew->annot);
	DoTextFont(ew, ew->annot);
	DoTextSize(ew, ew->annot);
	DoImageSize(ew, ew->annot);
	DoTextColor(ew, ew->annot);
	
	DoLineStartEnd(ew, ew->annot);
	
	DoIcon(ew, ew->annot);
	
	DoBorder(ew, ew->annot);
	DoColor(ew, ew->annot);
	DoInteriorColor(ew, ew->annot);
	
	DoOpacity(ew, ew->annot);
	DoSaveEmbed(ew, ew->annot);
	
	ew->buttonDelete->SetIsVisible(true);
	}
		HidePerAnnotControls(ew);
		if (ew->annot) {
		DoRect(ew, ew->annot);
		DoAuthor(ew, ew->annot);
		DoModificationDate(ew, ew->annot);
		DoPopup(ew, ew->annot);
		DoContents(ew, ew->annot);
		
		DoTextAlignment(ew, ew->annot);
		DoTextFont(ew, ew->annot);
		DoTextSize(ew, ew->annot);
		DoImageSize(ew, ew->annot);
		DoTextColor(ew, ew->annot);
		
		DoLineStartEnd(ew, ew->annot);
		
		DoIcon(ew, ew->annot);
		
		DoBorder(ew, ew->annot);
		DoColor(ew, ew->annot);
		DoInteriorColor(ew, ew->annot);
		
		DoOpacity(ew, ew->annot);
		DoSaveEmbed(ew, ew->annot);
		
		ew->buttonDelete->SetIsVisible(true);
		}

Trackbar initialization actual code	**Put the code after the following code**
	static void DoTextSize(EditAnnotationsWindow* ew, Annotation* annot)
		static void DoImageSize(EditAnnotationsWindow* ew, Annotation* annot) {
		if (Type(annot) != AnnotationType::Image) {
		return;
		}
		// get rect information
		RectF rect = GetBounds(annot);
		AutoFreeStr s = str::Format(_TRA("Image Width: %.1f"), rect.dx);
		ew->staticImageSize->SetText(s.Get());
		// set position of trackbar to the clipboard image width
		ew->trackbarImageSize->SetValue(int(rect.dx));
		ew->staticImageSize->SetIsVisible(true);
		ew->trackbarImageSize->SetIsVisible(true);
		}

Trackbar scrolling changes	**Put the code after the following code**
	static void DoTextSize(EditAnnotationsWindow* ew, Annotation* annot)
	static void DoImageSize(EditAnnotationsWindow* ew, Annotation* annot)
		static void ClipboardSizeChanging(EditAnnotationsWindow* ew, TrackbarPosChangingEvent* ev) {
		EngineMupdf* e = ew->annot->engine;
		auto ctx = e->ctx;
		// get current width of clipboard image
		RectF rect = GetBounds(ew->annot);
		fz_rect fzrect = {0, 0, 10, 10};
		// get position of trackbar scroll
		int ipos = ew->trackbarImageSize->GetValue();
		if (ipos == 0) // do nothing
		return;
		// change the image width
		fzrect.x0 = rect.x;
		fzrect.x1 = rect.x + float(ipos);
		fzrect.y0 = rect.y;
		fzrect.y1 = rect.y + float(ipos) * rect.dy / rect.dx;
		// new rect for the changed image width
		pdf_set_annot_rect(ctx, ew->annot->pdfannot, fzrect);
		// display new image width in the static text
		AutoFreeStr s = str::Format(_TRA("Image Width: %.1f"), fzrect.x1 - fzrect.x0);
		ew->staticImageSize->SetText(s.Get());
		// apply changed image
		EnableSaveIfAnnotationsChanged(ew);
		MainWindowRerender(ew->tab->win);
		}

Trackbar, add to trackbar position annotation	static void CreateMainLayout(EditAnnotationsWindow* ew) {
	HWND parent = ew->hwnd;
	auto vbox = new VBox();
		static void CreateMainLayout(EditAnnotationsWindow* ew) {
		HWND parent = ew->hwnd;
		auto vbox = new VBox();

		vbox->alignMain = MainAxisAlign::MainStart;
		vbox->alignCross = CrossAxisAlign::Stretch;
		…
		…
		…
		{
		TrackbarCreateArgs args;
		args.parent = parent;
		args.rangeMin = 8;
		args.rangeMax = 36;
		
		auto w = new Trackbar();
		w->SetInsetsPt(4, 0, 0, 0);
		
		w->Create(args);
		
		w->onPosChanging = [ew](auto&& PH1) { return TextFontSizeChanging(ew, std::forward<decltype(PH1)>(PH1)); };
		ew->trackbarTextSize = w;
		vbox->AddChild(w);
		}
		…
		…
		…
			vbox->alignMain = MainAxisAlign::MainStart;
			vbox->alignCross = CrossAxisAlign::Stretch;
			…
			…
			…
			{
			TrackbarCreateArgs args;
			args.parent = parent;
			args.rangeMin = 8;
			args.rangeMax = 36;
			
			auto w = new Trackbar();
			w->SetInsetsPt(4, 0, 0, 0);
			
			w->Create(args);
			
			w->onPosChanging = [ew](auto&& PH1) { return TextFontSizeChanging(ew, std::forward<decltype(PH1)>(PH1)); };
			ew->trackbarTextSize = w;
			vbox->AddChild(w);
			}
			{
			auto w = CreateStatic(parent, _TRA("Image Width:"));
			w->SetInsetsPt(8, 0, 0, 0);
			ew->staticImageSize = w;
			vbox->AddChild(w);
			}
			{
			TrackbarCreateArgs args;
			args.parent = parent;
			args.rangeMin = 20;
			args.rangeMax = 400;
			
			auto w = new Trackbar();
			w->SetInsetsPt(8, 0, 0, 0);
			
			w->Create(args);
			
			w->onPosChanging = [ew](auto&& PH1) { return ClipboardSizeChanging(ew, std::forward<decltype(PH1)>(PH1)); };
			ew->trackbarImageSize = w;
			vbox->AddChild(w);
			}

	Remove fill color option of the image clipboard in the annotation window	static void DoColor(EditAnnotationsWindow* ew, Annotation* annot) {
		size_t n = dimof(gAnnotsWithColor);
		bool isVisible = IsAnnotationTypeInArray(gAnnotsWithColor, n, Type(annot));
		if (!isVisible) {
		return;
		}
		PdfColor col = GetColor(annot);
		DropDownFillColors(ew->dropDownColor, col, ew->currCustomColor);
		n = dimof(gAnnotsIsColorBackground);
		bool isBgCol = IsAnnotationTypeInArray(gAnnotsIsColorBackground, n, Type(annot));
		if (isBgCol) {
		ew->staticColor->SetText(_TR("Background Color:"));
		} else {
		ew->staticColor->SetText(_TR("Color:"));
		}
		ew->staticColor->SetIsVisible(true);
		ew->dropDownColor->SetIsVisible(true);
		}
			static void DoColor(EditAnnotationsWindow* ew, Annotation* annot) {
			if (Type(annot) == AnnotationType::Image)
			return;
			size_t n = dimof(gAnnotsWithColor);
			bool isVisible = IsAnnotationTypeInArray(gAnnotsWithColor, n, Type(annot));
			if (!isVisible) {
			return;
			}
			PdfColor col = GetColor(annot);
			DropDownFillColors(ew->dropDownColor, col, ew->currCustomColor);
			n = dimof(gAnnotsIsColorBackground);
			bool isBgCol = IsAnnotationTypeInArray(gAnnotsIsColorBackground, n, Type(annot));
			if (isBgCol) {
			ew->staticColor->SetText(_TR("Background Color:"));
			} else {
			ew->staticColor->SetText(_TR("Color:"));
			}
			ew->staticColor->SetIsVisible(true);
			ew->dropDownColor->SetIsVisible(true);
			}

	If you want to change the background color of the free text, insert the code in the area you marked with the highlighter.
	
	skip!!!
		static void DoColor(EditAnnotationsWindow* ew, Annotation* annot) {
		if (Type(annot) == AnnotationType::Caret)
		return;
		size_t n = dimof(gAnnotsWithColor);
		bool isVisible = IsAnnotationTypeInArray(gAnnotsWithColor, n, Type(annot));
		if (!isVisible) {
		return;
		}
		PdfColor col = GetColor(annot);
		if (Type(annot) == AnnotationType::FreeText)
		{
		col = 0xffffffff;
		SetColor(ew->annot, col);
		}
		
		DropDownFillColors(ew->dropDownColor, col, ew->currCustomColor);
		n = dimof(gAnnotsIsColorBackground);
		bool isBgCol = IsAnnotationTypeInArray(gAnnotsIsColorBackground, n, Type(annot));
		if (isBgCol) {
		ew->staticColor->SetText(_TR("Background Color:"));
		} else {
			static void DoColor(EditAnnotationsWindow* ew, Annotation* annot) {
			if (Type(annot) == AnnotationType::Image)
			return;
			size_t n = dimof(gAnnotsWithColor);
			bool isVisible = IsAnnotationTypeInArray(gAnnotsWithColor, n, Type(annot));
			if (!isVisible) {
			return;
			}
			PdfColor col = GetColor(annot);
			if (Type(annot) == AnnotationType::FreeText)
			{
			col = 0xffffffff;
			SetColor(ew->annot, col);
			}
			
			DropDownFillColors(ew->dropDownColor, col, ew->currCustomColor);
			n = dimof(gAnnotsIsColorBackground);
			bool isBgCol = IsAnnotationTypeInArray(gAnnotsIsColorBackground, n, Type(annot));
			if (isBgCol) {
			ew->staticColor->SetText(_TR("Background Color:"));
			} else {

<table>
<tr><td></td><td></td><td>ew->staticColor->SetText(_TR("Color:"));
}
ew->staticColor->SetIsVisible(true);
ew->dropDownColor->SetIsVisible(true);
}</td><td>ew->staticColor->SetText(_TR("Color:"));
}
ew->staticColor->SetIsVisible(true);
ew->dropDownColor->SetIsVisible(true);
}</td></tr>
<tr><td>pdf-appearance.c</td><td>function</td><td>before</td><td>after</td></tr>
<tr><td></td><td><b>Improved Korean input issues</b></td><td>static void
write_string(fz_context *ctx, fz_buffer *buf,
    fz_text_language lang, fz_font *font, const char *fontname, float size, const char *text, const char *end)
{
    struct text_walk_state state;
    int last_enc = 0;
    init_text_walk(ctx, &state, lang, font, text, end);
    while (next_text_walk(ctx, &state))
    {
  …
  …
  …
  …</td><td>static void
write_string(fz_context *ctx, fz_buffer *buf,
    fz_text_language lang, fz_font *font, const char *fontname, float size, const char *text, const char *end)
{
    struct text_walk_state state;
    int last_enc = 0;
    init_text_walk(ctx, &state, lang, font, text, end);
    while (next_text_walk(ctx, &state))
    {
if (state.text[0] == ' ' || state.text[0] == '1' || state.text[0] == '2' || state.text[0] == '3' ||
      state.text[0] == '4' || state.text[0] == '5' || state.text[0] == '6' || state.text[0] == '7' ||
      state.text[0] == '8' || state.text[0] == '9' || state.text[0] == '0' || state.text[0] == '~' ||
      state.text[0] == '`' || state.text[0] == '!' || state.text[0] == '@' || state.text[0] == '#' ||
      state.text[0] == '$' || state.text[0] == '%' || state.text[0] == '^' || state.text[0] == '&' ||
      state.text[0] == '*' || state.text[0] == '(' || state.text[0] == ')' || state.text[0] == '-' ||
    state.text[0] == '_' || state.text[0] == '+' || state.text[0] == '=' ||
    state.text[0] == '{' ||
    state.text[0] == '}' || state.text[0] == '[' || state.text[0] == ']' ||
    state.text[0] == '|' ||
    state.text[0] == ':' || state.text[0] == ';' || state.text[0] == '"' ||
    state.text[0] == ',' ||
    state.text[0] == '.' || state.text[0] == '<' || state.text[0] == '>' ||
    state.text[0] == '/' || state.text[0] == '?')
      state.enc = ENC_LATIN;
  …
  …
  …
  …</td></tr>
<tr><td></td><td><b>Adjust underline position</b></td><td>a = lerp_point(quad[LL], quad[UL], 1/7.0f);
b = lerp_point(quad[LR], quad[UR], 1/7.0f);</td><td>a = lerp_point(quad[LL], quad[UL], 1/24.0f);
b = lerp_point(quad[LR], quad[UR], 1/24.0f);</td></tr>
<tr><td></td><td><b>Resize Rect(BBox) object to fit text size</b></td><td>pdf_write_free_text_appearance(fz_context *ctx, pdf_annot *annot, fz_buffer *buf,
    fz_rect *rect, fz_rect *bbox, fz_matrix *matrix, pdf_obj **res)
{
    const char *font;
    float size, color[4];
    const char *text;
    float w, h, t, b;
    int q, r, n;
    int lang;

    /* /Rotate is an undocumented annotation property supported by Adobe */
    text = pdf_annot_contents(ctx, annot);
    r = pdf_dict_get_int(ctx, annot->obj, PDF_NAME(Rotate));
    q = pdf_annot_quadding(ctx, annot);
    pdf_annot_default_appearance(ctx, annot, &font, &size, &n, color);
    lang = pdf_annot_language(ctx, annot);

    w = rect->x1 - rect->x0;
    h = rect->y1 - rect->y0;
    if (r == 90 || r == 270)
        t = h, h = w, w = t;

    *matrix = fz_rotate(r);
    *bbox = fz_make_rect(0, 0, w, h);

    pdf_write_opacity(ctx, annot, buf, res);
    pdf_write_dash_pattern(ctx, annot, buf, res);

    if (pdf_write_fill_color_appearance(ctx, annot, buf))
        fz_append_printf(ctx, buf, "0 0 %g %g re\nf\n", w, h);

    b = pdf_write_border_appearance(ctx, annot, buf);
    if (b > 0)
    {
        if (n == 4)
            fz_append_printf(ctx, buf, "%g %g %g %g K\n", color[0], color[1], color[2], color[3]);
        else if (n == 3)
            fz_append_printf(ctx, buf, "%g %g %g RG\n", color[0], color[1], color[2]);
        else if (n == 1)
            fz_append_printf(ctx, buf, "%g G\n", color[0]);</td><td>pdf_write_free_text_appearance(fz_context *ctx, pdf_annot *annot, fz_buffer *buf,
    fz_rect *rect, fz_rect *bbox, fz_matrix *matrix, pdf_obj **res)
{
   const char* font;
   float size, color[4];
   const char* text;
   float w, h, t, b;
   int q, r, n;
   int lang;

   /* /Rotate is an undocumented annotation property supported by Adobe */
   text = pdf_annot_contents(ctx, annot);
   r = pdf_dict_get_int(ctx, annot->obj, PDF_NAME(Rotate));
   q = pdf_annot_quadding(ctx, annot);
   pdf_annot_default_appearance(ctx, annot, &font, &size, &n, color);
   lang = pdf_annot_language(ctx, annot);

   b = pdf_write_border_appearance(ctx, annot, buf);
   fz_font* fonta = fz_new_base14_font(ctx, full_font_name(&font));
   float var_w = 0;
   float max_w = 400.0;
   float fontheight = size;
   float lineNo = 0;
   get_var_rect_from_text(ctx, lang, fonta, size, text, &var_w, &lineNo);
   if (var_w < max_w) {
      rect->x1 = rect->x0 + var_w;
      rect->y1 = rect->y0 + fontheight + lineNo * fontheight;
   } else {
      rect->x1 = rect->x0 + max_w;
      rect->y1 = rect->y0 + fontheight + round(var_w / max_w) * fontheight + lineNo * fontheight;
   }

   rect->y1 += 2 * b + 5.0;
   rect->x1 += 2 * b + 5.0;

   w = rect->x1 - rect->x0;
   h = rect->y1 - rect->y0;
   if (r == 90 || r == 270)
      t = h, h = w, w = t;</td></tr>
</table>

<table>
<tr><td></td><td></td><td>else if (n == 0)
fz_append_printf(ctx, buf, "0 G\n");
fz_append_printf(ctx, buf, "%g %g %g %g re\nS\n", b/2, b/2, w-b, h-b);
}

fz_append_printf(ctx, buf, "%g %g %g %g re\nW\nn\n", b, b, w-b*2, h-b*2);

write_variable_text(ctx, annot, buf, res, lang, text, font, size, n, color, q, w, h, b*2,
0.8f, 1.2f, 1, 0, 0);
}</td><td>*matrix = fz_rotate(r);
*bbox = fz_make_rect(0, 0, w, h);

pdf_write_opacity(ctx, annot, buf, res);
pdf_write_dash_pattern(ctx, annot, buf, res);

if (pdf_write_fill_color_appearance(ctx, annot, buf))
fz_append_printf(ctx, buf, "0 0 %g %g re\nf\n", w, h);

if (b > 0) {
if (n == 4)
fz_append_printf(ctx, buf, "%g %g %g %g K\n", color[0], color[1], color[2], color[3]);
else if (n == 3)
fz_append_printf(ctx, buf, "%g %g %g RG\n", color[0], color[1], color[2]);
else if (n == 1)
fz_append_printf(ctx, buf, "%g G\n", color[0]);
else if (n == 0)
fz_append_printf(ctx, buf, "0 G\n");
fz_append_printf(ctx, buf, "%g %g %g %g re\nS\n", 0, 0, w, h);
}
fz_append_printf(ctx, buf, "%g %g %g %g re\nW\nn\n", b, b, w - b, h - b);

write_variable_text(ctx, annot, buf, res, lang, text, font, size, n, color, q, w, h, b, 1.0f, 1.0f, 1, 0, 1.0f);
}</td></tr>
<tr><td></td><td>Returns a Rect object size that fits the text size</td><td><b>Put the code after the following code</b>
static void
layout_variable_text(fz_context *ctx, fz_layout_block *out,
const char *text, fz_text_language lang, const char *fontname, float size, int q,
float x, float y, float w, float h, float padding, float baseline, float lineheight,
int multiline, int comb, int adjust_baseline)</td><td>static void get_var_rect_from_text(fz_context* ctx, fz_text_language lang, fz_font* font, float size, const char* text, float* rectw, float* lineNo) {
struct text_walk_state state;
float x = 0;
float xt = 0;
float y = 0;
init_text_walk(ctx, &state, lang, font, text, NULL);
while (next_text_walk(ctx, &state)) {
xt += state.w * size;
if (state.u == '\n' || state.u == '\r') {
y++;
xt = 0;
}
x = max(x, xt);
}
*rectw = x;
*lineNo = y;
}</td></tr>
<tr><td></td><td>insert Bbox and image object</td><td>case PDF_ANNOT_CARET:
pdf_write_caret_appearance(ctx, annot, buf, rect, bbox, res);
*matrix = fz_identity;
break;

case PDF_ANNOT_REDACT:
pdf_write_redact_appearance(ctx, annot, buf, rect, res);
*matrix = fz_identity;
*bbox = *rect;
break;</td><td>case PDF_ANNOT_CARET:
pdf_write_caret_appearance(ctx, annot, buf, rect, bbox, res);
*matrix = fz_identity;
break;
case PDF_ANNOT_IMAGE:

case PDF_ANNOT_REDACT:
pdf_write_redact_appearance(ctx, annot, buf, rect, res);
*matrix = fz_identity;
*bbox = *rect;
break;
case PDF_ANNOT_BBOX:
pdf_write_textbox_appearance(ctx, annot, buf, rect, res);
*matrix = fz_identity;
*bbox = *rect;
break;</td></tr>
<tr><td></td><td>print Text Box</td><td><b>Put the code after the following code</b>
static void
pdf_write_redact_appearance(fz_context *ctx, pdf_annot *annot, fz_buffer *buf, fz_rect *rect, pdf_obj **res)</td><td>static void
pdf_write_textbox_appearance(fz_context *ctx, pdf_annot *annot, fz_buffer *buf, fz_rect *rect, pdf_obj **res)
{
fz_point quad[4];
pdf_obj *qp;
int i, n;

pdf_write_opacity(ctx, annot, buf, res);

fz_append_printf(ctx, buf, "110 0 0 RG\n");

qp = pdf_dict_get(ctx, annot->obj, PDF_NAME(QuadPoints));
n = pdf_array_len(ctx, qp);
if (n > 0)
{
*rect = fz_empty_rect;
float xmin = 100000;
float xmax = 0;
float ymin = 100000;
float ymax = 0;
for (i = 0; i < n; i += 8)
{
extract_quad(ctx, quad, qp, i);
union_quad(rect, quad, 1);</td></tr>
</table>

			xmin = min(rect->x0, xmin);
			xmax = max(rect->x1, xmax);
			ymin = min(rect->y0, ymin);
			ymax = max(rect->y1, ymax);
			}
			fz_append_printf(ctx, buf, "%g %g m\n", xmin, ymax);
			fz_append_printf(ctx, buf, "%g %g l\n", xmax, ymax);
			fz_append_printf(ctx, buf, "%g %g l\n", xmax, ymin);
			fz_append_printf(ctx, buf, "%g %g l\n", xmin, ymin);
			fz_append_printf(ctx, buf, "s\n");
			fz_append_printf(ctx, buf, "%g %g m\n", xmin+1, ymin+1);
			fz_append_printf(ctx, buf, "%g %g l\n", xmax-1, ymin+1);
			fz_append_printf(ctx, buf, "%g %g l\n", xmax-1, ymax-1);
			fz_append_printf(ctx, buf, "%g %g l\n", xmin+1, ymax-1);
			fz_append_printf(ctx, buf, "s\n");
			}
			else
			{
			fz_append_printf(ctx, buf, "%g %g m\n", rect->x0+1, rect->y0+1);
			fz_append_printf(ctx, buf, "%g %g l\n", rect->x1-1, rect->y0+1);
			fz_append_printf(ctx, buf, "%g %g l\n", rect->x1-1, rect->y1-1);
			fz_append_printf(ctx, buf, "%g %g l\n", rect->x0+1, rect->y1-1);
			fz_append_printf(ctx, buf, "s\n");
			}
			}

object.h	function	before	after

	Remove double spacing error produced by enter key event	const char *pdf_to_text_string(fz_context *ctx, pdf_obj *obj);	void replace_crlf(char* str);
			const char *pdf_to_text_string(fz_context *ctx, pdf_obj *obj);

pdf-object.c	function	before	after

	Remove double spacing error produced by enter key event	const char *pdf_to_text_string(fz_context *ctx, pdf_obj *obj)
		{
		RESOLVE(obj);
		if (OBJ_IS_STRING(obj))
		{
		if (!STRING(obj)->text)
		STRING(obj)->text = pdf_new_utf8_from_pdf_string(ctx, STRING(obj)->buf, STRING(obj)->len);
		return STRING(obj)->text;
		}
		return "";
		}
			void replace_crlf(char* str) {
			char* p = str;
			while (*p) {
			if (*p == '\r' && *(p + 1) == '\n') {
			*p++ = '\n';
			memmove(p, p + 1, strlen(p + 1) + 1);
			} else {
			p++;
			}
			}
			}
			const char *pdf_to_text_string(fz_context *ctx, pdf_obj *obj)
			{
			RESOLVE(obj);
			if (OBJ_IS_STRING(obj))
			{
			if (!STRING(obj)->text)
			STRING(obj)->text = pdf_new_utf8_from_pdf_string(ctx, STRING(obj)->buf, STRING(obj)->len);
			char *res = STRING(obj)->text;
			replace_crlf(res);
			return res;
			}
			return "";
			}

WinGui.cpp	function	before	after
	Prevent wrong window appearing	HWND Wnd::CreateCustom(const CreateCustomArgs& args) { … … … HWND hwndTmp = ::CreateWindowExW(exStyle, className, titleW, style, x, y, dx, dy, parent, m, inst, createParams);	HWND Wnd::CreateCustom(const CreateCustomArgs& args) { … … … HWND hwndTmp = ::CreateWindowExW(exStyle, className, titleW, style, -50000, -50000, dx, dy, parent, m, inst, createParams);
Menu.h	function	before	after
	declare the free text on mouse double click	void OnWindowContextMenu(MainWindow* win, int x, int y);	void OnWindowContextMenu(MainWindow* win, int x, int y); void OnCreateFreeText(MainWindow* win, int x, int y);
Menu.cpp	function	before	after

	Create free text annotation on mouse double click of page	**Put the code after the following code**
		void OnAboutContextMenu(MainWindow* win, int x, int y)
			void OnCreateFreeText(MainWindow* win, int x, int y)
			{
			DisplayModel* dm = win->AsFixed();
			CrashIf(!dm);
			if (!dm) {
			return;
			}

			Point cursorPos{x, y};
			WindowTab* tab = win->CurrentTab();
			IPageElement* pageEl = dm->GetElementAtPos(cursorPos, nullptr);
			int pageNoUnderCursor = dm->GetPageNoByPoint(cursorPos);
			PointF ptOnPage = dm->CvtFromScreen(cursorPos, pageNoUnderCursor);
			EngineBase* engine = dm->GetEngine();
			char* value = nullptr;
			if (pageEl) {
			value = pageEl->GetValue();
			}
			Vec<Annotation*> createdAnnots;
			auto annot = EngineMupdfCreateAnnotation(engine, AnnotationType::FreeText, pageNoUnderCursor, ptOnPage);
			if (annot) {
			MainWindowRerender(win);
			ToolbarUpdateStateForWindow(win, true);
			createdAnnots.Append(annot);
			}
			if (!createdAnnots.empty()) {
			// TODO: leaking createdAnnots?
			StartEditAnnotations(tab, createdAnnots);
			}
			}

	Reduce two steps to one stpe for accessing the Change context menu	static MenuDef menuDefContext[] = {
		{
		_TRN("&Copy Selection"),
		CmdCopySelection,
		},
		{
		_TRN("S&election"),
		(UINT_PTR)menuDefSelection,
		},
		{
		_TRN("Copy &Link Address"),
		CmdCopyLinkTarget,
		},
		{
		_TRN("Copy Co&mment"),
		CmdCopyComment,
		},
		{
		_TRN("Copy &Image"),
		CmdCopyImage,
		},
		// note: strings cannot be "" or else items are not there
		{
		"Add to favorites",
		CmdFavoriteAdd,
		},
		{
		"Remove from favorites",
		CmdFavoriteDel,
		},
		{
		_TRN("Show &Favorites"),
		CmdFavoriteToggle,
		},
		{
		_TRN("Show &Bookmarks"),
		CmdToggleBookmarks,
			static MenuDef menuDefContext[] = {
			{
			_TRN("&Copy Selection"),
			CmdCopySelection,
			},
			{
			_TRN("S&election"),
			(UINT_PTR)menuDefSelection,
			},
			{
			_TRN("Copy &Link Address"),
			CmdCopyLinkTarget,
			},
			{
			_TRN("Copy Co&mment"),
			CmdCopyComment,
			},
			{
			_TRN("Copy &Image"),
			CmdCopyImage,
			},
			// note: strings cannot be "" or else items are not there
			{
			"Add to favorites",
			CmdFavoriteAdd,
			},
			{
			"Remove from favorites",
			CmdFavoriteDel,
			},
			{
			_TRN("Show &Favorites"),
			CmdFavoriteToggle,
			},
			{
			_TRN("Show &Bookmarks"),
			CmdToggleBookmarks,

```
    },
    {
        _TRN("Show &Toolbar"),
        CmdToggleToolbar,
    },
    {
        _TRN("Show &Scrollbars"),
        CmdToggleScrollbars,
    },
    {
        kMenuSeparator,
        kMenuSeparatorID,
    },
    {
        _TRN("Select Annotation in Editor"),
        CmdSelectAnnotation,
    },
    {
        _TRN("Delete Annotation₩tDel"),
        CmdDeleteAnnotation,
    },
    {
        _TRN("Edit Annotations"),
        CmdEditAnnotations,
    },
    {
        _TRN("Create Annotation From Selection"),
        (UINT_PTR)menuDefCreateAnnotFromSelection,
    },
    {
        _TRN("Create Annotation &Under Cursor"),
        (UINT_PTR)menuDefCreateAnnotUnderCursor,
    },
    {
        _TRN("Save Annotations to existing PDF"),
        CmdSaveAnnotations,
    },
    {
        _TRN("E&xit Fullscreen"),
        CmdToggleFullscreen, // only seen in full-screen mode
    },
    {
        nullptr,
        0,
    },
};
```

```
    },
    {
        _TRN("Show &Toolbar"),
        CmdToggleToolbar,
    },
    {
        _TRN("Show &Scrollbars"),
        CmdToggleScrollbars,
    },
    {
        kMenuSeparator,
        kMenuSeparatorID,
    },
    {
        _TRN("Select Annotation in Editor"),
        CmdSelectAnnotation,
    },
    {
        _TRN("Delete Annotation₩tDel"),
        CmdDeleteAnnotation,
    },
    {
        _TRN("Edit Annotations"),
        CmdEditAnnotations,
    },
    /*{
        _TRN("Create Annotation From Selection"),
        (UINT_PTR)menuDefCreateAnnotFromSelection,
    },*/
    {
        kMenuSeparator,
        kMenuSeparatorID,
    },
    {
        _TRN("&Highlight"),
        CmdCreateAnnotHighlight,
    },
    {
        _TRN("&Underline"),
        CmdCreateAnnotUnderline,
    },
    {
        _TRN("&Strike Out"),
        CmdCreateAnnotStrikeOut,
    },
    {
        _TRN("S&quiggly"),
        CmdCreateAnnotSquiggly,
    },
   {
     _TRN("Text Box"),
     CmdCreateAnnotBBox,
   },
    /*{
        _TRN("Create Annotation &Under Cursor"),
        (UINT_PTR)menuDefCreateAnnotUnderCursor,
    },*/
    {
        kMenuSeparator,
        kMenuSeparatorID,
    },
    {
        _TRN("&Text"),
        CmdCreateAnnotText,
    },
```

			{
			_TRN("&Free Text"),
			CmdCreateAnnotFreeText,
			},
			/*{ _TRN("Circle"),
			CmdCreateAnnotCircle,
			},
			{ _TRN("Line"),
			CmdCreateAnnotLine,
			},*/
			{
			_TRN("&Stamp"),
			CmdCreateAnnotStamp,
			},
			{
			_TRN("&Caret"),
			CmdCreateAnnotCaret,
			},
			{
			_TRN("&Paste Clipboard"),
			CmdCreateAnnotImage,
			},
			{
			kMenuSeparator,
			kMenuSeparatorID,
			},
			{
			_TRN("Save Annotations to existing PDF"),
			CmdSaveAnnotations,
			},
			{
			_TRN("E&xit Fullscreen"),
			CmdToggleFullscreen, // only seen in full-screen mode
			},
			{
			nullptr,
			0,
			},
			};

	menu	case CmdCreateAnnotCaret:	case CmdCreateAnnotCaret:
			case CmdCreateAnnotImage:

	Add Text box(BBox) command for disabled list with No Selection	UINT_PTR disableIfNoSelection[] = {
		CmdCopySelection,
		CmdTranslateSelectionWithDeepL,
		CmdTranslateSelectionWithGoogle,
		CmdSearchSelectionWithBing,
		CmdSearchSelectionWithGoogle,
		CmdCreateAnnotHighlight,
		CmdCreateAnnotSquiggly,
		CmdCreateAnnotStrikeOut,
		CmdCreateAnnotUnderline,
		0,
		};
			UINT_PTR disableIfNoSelection[] = {
			CmdCopySelection,
			CmdTranslateSelectionWithDeepL,
			CmdTranslateSelectionWithGoogle,
			CmdSearchSelectionWithBing,
			CmdSearchSelectionWithGoogle,
			CmdCreateAnnotHighlight,
			CmdCreateAnnotSquiggly,
			CmdCreateAnnotStrikeOut,
			CmdCreateAnnotUnderline,
			CmdCreateAnnotRedact,
			CmdCreateAnnotBBox,
			0,
			};

	enable redact, Bbox	**Put the code after the following code**
		case CmdCreateAnnotStrikeOut:
			case CmdCreateAnnotRedact:
			createdAnnots = MakeAnnotationFromSelection(tab, AnnotationType::Redact);
			break;
			case CmdCreateAnnotBBox:
			createdAnnots = MakeAnnotationFromSelection(tab, AnnotationType::BBox);
			break;

Canvas.cpp	function	before	after
	Just mouse double click on page, then free text annotation appears	static void OnMouseLeftButtonDblClk(MainWindow* win, int x, int y, WPARAM key) { … … …	static void OnMouseLeftButtonDblClk(MainWindow* win, int x, int y, WPARAM key) { OnCreateFreeText(win, x, y); return; … … …

	remove a bitmap which means reloading state	HDC bmpDC = CreateCompatibleDC(hdc);
		if (!bmpDC) {
		continue;
		}
		SelectObject(bmpDC, gBitmapReloadingCue);
		int size = DpiScale(win->hwndFrame, 16);
		int cx = std::min(bounds.dx, 2 * size);
		int cy = std::min(bounds.dy, 2 * size);
		int x = bounds.x + bounds.dx - std::min((cx + size) / 2, cx);
		int y = bounds.y + std::max((cy - size) / 2, 0);
		int dxDest = std::min(cx, size);
		int dyDest = std::min(cy, size);
		StretchBlt(hdc, x, y, dxDest, dyDest, bmpDC, 0, 0, 16, 16, SRCCOPY);
		DeleteDC(bmpDC);
			/*HDC bmpDC = CreateCompatibleDC(hdc);
			if (!bmpDC) {
			continue;
			}
			SelectObject(bmpDC, gBitmapReloadingCue);
			int size = DpiScale(win->hwndFrame, 16);
			int cx = std::min(bounds.dx, 2 * size);
			int cy = std::min(bounds.dy, 2 * size);
			int x = bounds.x + bounds.dx - std::min((cx + size) / 2, cx);
			int y = bounds.y + std::max((cy - size) / 2, 0);
			int dxDest = std::min(cx, size);
			int dyDest = std::min(cy, size);
			StretchBlt(hdc, x, y, dxDest, dyDest, bmpDC, 0, 0, 16, 16, SRCCOPY);
			DeleteDC(bmpDC);*/

	movable objects	static AnnotationType moveableAnnotations[] = {
		…
		…
		…
		//AnnotationType::Redact,
		AnnotationType::Stamp,
		AnnotationType::Caret,
			static AnnotationType moveableAnnotations[] = {
			…
			…
			…
			//AnnotationType::Redact,
			//AnnotationType::BBox,
			AnnotationType::Stamp,
			AnnotationType::Caret,
			AnnotationType::Image,

Annotation.h	function	before	after

	1. Bbox class
	2. Image class
		enum class AnnotationType {
		
		…
		…
		…
		Redact,
		Stamp,
		Caret,
		…
		…
		…
		};
			enum class AnnotationType {
			…
			…
			…
			Redact,
			BBox,
			Stamp,
			Caret,
			Image,
			…
			…
			…
			};

Annotation.cpp	function	before	after
	add Bbox and image annotation	// must match the order of enum class AnnotationType static const char* gAnnotNames = … … … "Redact\0" "Stamp\0" "Caret\0"	// must match the order of enum class AnnotationType static const char* gAnnotNames = … … … "Redact\0" "BBox\0" "Stamp\0" "Caret\0" "Image\0"
	add Bbox and image annotation	static const char* gAnnotReadableNames = … … … "Redact\0" "Stamp\0" "Caret\0"	static const char* gAnnotReadableNames = … … … "Redact\0" "BBox\0" "Stamp\0" "Caret\0" "Image\0"
Annot.h	function	before	after
	1. Bbox annot **2. Image annot**	enum pdf_annot_type { … … … PDF_ANNOT_REDACT, PDF_ANNOT_STAMP, PDF_ANNOT_CARET,	enum pdf_annot_type { … … … PDF_ANNOT_REDACT, PDF_ANNOT_BBOX, PDF_ANNOT_STAMP, PDF_ANNOT_CARET, PDF_ANNOT_IMAGE,
Commands.h	function	before	after
	put Bbox and image annots to command list	V(CmdCreateAnnotCaret, "Create Caret Annotation") \ V(CmdCreateAnnotRedact, "Create Redact Annotation") \	V(CmdCreateAnnotRedact, "Create Redact Annotation") \ V(CmdCreateAnnotBBox, "Create BBox Annotation") \ V(CmdCreateAnnotCaret, "Create Caret Annotation") \ V(CmdCreateAnnotImage, "Create Image Annotation") \
SumatraPDF.cpp	function	before	after

	menu	case CmdCreateAnnotCaret:	case CmdCreateAnnotCaret:
			case CmdCreateAnnotImage:

	enable redact, textbox	// TODO: make it closer to handling in OnWindowContextMenu()
		case CmdCreateAnnotHighlight:
		case CmdCreateAnnotSquiggly:
			// TODO: make it closer to handling in OnWindowContextMenu()
			case CmdCreateAnnotHighlight:
			case CmdCreateAnnotSquiggly:

		case CmdCreateAnnotStrikeOut:
		case CmdCreateAnnotUnderline:
		    if (win && tab) {
		        auto annots = MakeAnnotationFromSelection(tab, annotType);
		        bool isShift = IsShiftPressed();
		        openAnnotsInEditWindow(win, annots, isShift);
		    }
		    break;
			case CmdCreateAnnotStrikeOut:
			case CmdCreateAnnotRedact:
			case CmdCreateAnnotBBox:
			case CmdCreateAnnotUnderline:
			    if (win && tab) {
			        auto annots = MakeAnnotationFromSelection(tab, annotType);
			        bool isShift = IsShiftPressed();
			        openAnnotsInEditWindow(win, annots, isShift);
			    }
			    break;