Summary of customization (SumatraPDF)

2023년 5월 16일 화요일 오후 12:08

File	Changes (The highlighted area is different from the original)	Note

Annot.h	enum pdf_annot_type
	{
	PDF_ANNOT_TEXT,
	PDF_ANNOT_LINK,
	PDF_ANNOT_FREE_TEXT,
	PDF_ANNOT_LINE,
	PDF_ANNOT_SQUARE,
	PDF_ANNOT_CIRCLE,
	PDF_ANNOT_POLYGON,
	PDF_ANNOT_POLY_LINE,
	PDF_ANNOT_HIGHLIGHT,
	PDF_ANNOT_UNDERLINE,
	PDF_ANNOT_SQUIGGLY,
	PDF_ANNOT_STRIKE_OUT,
	PDF_ANNOT_REDACT,
	PDF_ANNOT_STAMP,
	PDF_ANNOT_CARET,
	PDF_ANNOT_IMAGE,
	PDF_ANNOT_INK,
	PDF_ANNOT_POPUP,
	PDF_ANNOT_FILE_ATTACHMENT,
	PDF_ANNOT_SOUND,
	PDF_ANNOT_MOVIE,
	PDF_ANNOT_RICH_MEDIA,
	PDF_ANNOT_WIDGET,
	PDF_ANNOT_SCREEN,
	PDF_ANNOT_PRINTER_MARK,
	PDF_ANNOT_TRAP_NET,
	PDF_ANNOT_WATERMARK,
	PDF_ANNOT_3D,
	PDF_ANNOT_PROJECTION,
	PDF_ANNOT_UNKNOWN = -1
	};	

Annotation.cpp	#if 0 // must match the order of enum class AnnotationType static const char* gAnnotNames = "Text₩0" "Link₩0" "FreeText₩0" "Line₩0" "Square₩0" "Circle₩0" "Polygon₩0" "PolyLine₩0" "Highlight₩0" "Underline₩0" "Squiggly₩0" "StrikeOut₩0" "Redact₩0" "Stamp₩0" "Caret₩0" "Image₩0" "Ink₩0" "Popup₩0" "FileAttachment₩0" "Sound₩0"	

	"Movie₩0" "RichMedia₩0" "Widget₩0" "Screen₩0" "PrinterMark₩0" "TrapNet₩0" "Watermark₩0" "3D₩0" "Projection₩0"; #endif static const char* gAnnotReadableNames = "Text₩0" "Link₩0" "Free Text₩0" "Line₩0" "Square₩0" "Circle₩0" "Polygon₩0" "Poly Line₩0" "Highlight₩0" "Underline₩0" "Squiggly₩0" "StrikeOut₩0" "Redact₩0" "Stamp₩0" "Caret₩0" "Image₩0" "Ink₩0" "Popup₩0" "File Attachment₩0" "Sound₩0" "Movie₩0" "RichMedia₩0" "Widget₩0" "Screen₩0" "Printer Mark₩0" "Trap Net₩0" "Watermark₩0" "3D₩0" "Projection₩0"; // clang format-on	

Annotation.h	// for fast conversions, must match the order of pdf_annot_type enum in annot.h
	enum class AnnotationType {
	Text,
	Link,
	FreeText,
	Line,
	Square,
	Circle,
	Polygon,
	PolyLine,
	Highlight,
	Underline,
	Squiggly,
	StrikeOut,
	Redact,
	Stamp,
	Caret,
	Image,
	Ink,
	Popup,	

	FileAttachment, Sound, Movie, RichMedia, Widget, Screen, PrinterMark, TrapNet, Watermark, ThreeD, Projection, Unknown = -1 };	

Canvas.cpp	// clang-format off
	static AnnotationType moveableAnnotations[] = {
	AnnotationType::Text,
	AnnotationType::Link,
	AnnotationType::FreeText,
	AnnotationType::Line,
	AnnotationType::Square,
	AnnotationType::Circle,
	AnnotationType::Polygon,
	AnnotationType::PolyLine,
	//AnnotationType::Highlight,
	//AnnotationType::Underline,
	//AnnotationType::Squiggly,
	//AnnotationType::StrikeOut,
	//AnnotationType::Redact,
	AnnotationType::Stamp,
	AnnotationType::Caret,
	AnnotationType::Image,
	AnnotationType::Ink,
	AnnotationType::Popup,
	AnnotationType::FileAttachment,
	AnnotationType::Sound,
	AnnotationType::Movie,
	//AnnotationType::Widget, // TODO: maybe moveble?
	AnnotationType::Screen,
	AnnotationType::PrinterMark,
	AnnotationType::TrapNet,
	AnnotationType::Watermark,
	AnnotationType::ThreeD,
	AnnotationType::Unknown,
	};	

Commands.h	V(CmdCreateAnnotStamp, "Create Stamp Annotation") ₩ V(CmdCreateAnnotCaret, "Create Caret Annotation") ₩ V(CmdCreateAnnotImage, "Create Image Annotation") ₩ V(CmdCreateAnnotInk, "Create Ink Annotation") ₩ V(CmdCreateAnnotPopup, "Create Popup Annotation") ₩ V(CmdCreateAnnotFileAttachment, "Create File Attachment Annotation") ₩	

EditAnnotaions.cpp	#include "utils/Log.h"
	#include <iostream>
	#include <fstream>
	
	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,
	};	

```
    Trackbar* trackbarTextSize = nullptr;
    Static* staticImageSize = nullptr;
    Trackbar* trackbarImageSize = nullptr;
    Static* staticTextColor = nullptr;
    DropDown* dropDownTextColor = nullptr;


    ew->trackbarTextSize->SetIsVisible(false);
    ew->staticImageSize->SetIsVisible(false);
    ew->trackbarImageSize->SetIsVisible(false);
    ew->staticTextColor->SetIsVisible(false);
    ew->dropDownTextColor->SetIsVisible(false);


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);
    SetFocus(ew->editContents->hwnd);
}

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);
}

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);
}
```

```
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 (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);
    }

    {
        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);
    }

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:
        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, "");
        pdf_set_annot_border(ctx, annot, 0);
    }

    pdf_update_annot(ctx, annot);

    auto res = MakeAnnotationPdf(epdf, annot, pageNo);
    if (typ == AnnotationType::Text) {
        AutoFreeStr iconName = GetAnnotationTextIcon();
        if (!str::Eql(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);
        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;

}

//[ ACCESSKEY_GROUP Context Menu (Create annot under cursor)
static MenuDef menuDefCreateAnnotUnderCursor[] = {
    {
        _TRN("&Text"),
        CmdCreateAnnotText,
    },
    {
        _TRN("&Free Text"),
        CmdCreateAnnotFreeText,
    },
    {
        _TRN("&Stamp"),
        CmdCreateAnnotStamp,
    },
    {
        _TRN("&Caret"),
        CmdCreateAnnotCaret,
    },
    {
        _TRN("&Paste Clipboard"),
        CmdCreateAnnotImage,
    },
    //{ _TRN("Ink"), CmdCreateAnnotInk, },
    //{ _TRN("Square"), CmdCreateAnnotSquare, },
    //{ _TRN("Circle"), CmdCreateAnnotCircle, },
    //{ _TRN("Line"), CmdCreateAnnotLine, },
    //{ _TRN("Polygon"), CmdCreateAnnotPolygon, },
    //{ _TRN("Poly Line"), CmdCreateAnnotPolyLine, },
    //{ _TRN("File Attachment"), CmdCreateAnnotFileAttachment, },
    {
        nullptr,
        0,
    },
};

        // Note: duplicated in OnWindowContextMenu because slightly different handling
        case CmdCreateAnnotText:
        case CmdCreateAnnotFreeText:
        case CmdCreateAnnotStamp:
        case CmdCreateAnnotCaret:
        case CmdCreateAnnotImage:
        case CmdCreateAnnotSquare:
```

```
        case CmdCreateAnnotLine:
        case CmdCreateAnnotCircle: {
            auto annot = EngineMupdfCreateAnnotation(engine, annotType, pageNoUnderCursor, ptOnPage);
            if (annot) {
                MainWindowRerender(win);
                ToolbarUpdateStateForWindow(win, true);
                createdAnnots.Append(annot);
            }
        } break;
```

object.h

```
const char *pdf_to_name(fz_context *ctx, pdf_obj *obj);
void replace_crlf(char* str);
const char *pdf_to_text_string(fz_context *ctx, pdf_obj *obj);
```

pdf-annot.c

```
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);
}

const char *
pdf_string_from_annot_type(fz_context *ctx, enum pdf_annot_type type)
{
        switch (type)
        {
        case PDF_ANNOT_TEXT: return "Text";
        case PDF_ANNOT_LINK: return "Link";
        case PDF_ANNOT_FREE_TEXT: return "FreeText";
        case PDF_ANNOT_LINE: return "Line";
        case PDF_ANNOT_SQUARE: return "Square";
        case PDF_ANNOT_CIRCLE: return "Circle";
        case PDF_ANNOT_POLYGON: return "Polygon";
        case PDF_ANNOT_POLY_LINE: return "PolyLine";
        case PDF_ANNOT_HIGHLIGHT: return "Highlight";
        case PDF_ANNOT_UNDERLINE: return "Underline";
        case PDF_ANNOT_SQUIGGLY: return "Squiggly";
        case PDF_ANNOT_STRIKE_OUT: return "StrikeOut";
        case PDF_ANNOT_REDACT: return "Redact";
        case PDF_ANNOT_STAMP: return "Stamp";
        case PDF_ANNOT_CARET: return "Caret";
        case PDF_ANNOT_IMAGE: return "Image";
        case PDF_ANNOT_INK: return "Ink";
        case PDF_ANNOT_POPUP: return "Popup";
        case PDF_ANNOT_FILE_ATTACHMENT: return "FileAttachment";
        case PDF_ANNOT_SOUND: return "Sound";
        case PDF_ANNOT_MOVIE: return "Movie";
        case PDF_ANNOT_RICH_MEDIA: return "RichMedia";
        case PDF_ANNOT_WIDGET: return "Widget";
        case PDF_ANNOT_SCREEN: return "Screen";
        case PDF_ANNOT_PRINTER_MARK: return "PrinterMark";
        case PDF_ANNOT_TRAP_NET: return "TrapNet";
        case PDF_ANNOT_WATERMARK: return "Watermark";
        case PDF_ANNOT_3D: return "3D";
        case PDF_ANNOT_PROJECTION: return "Projection";
        default: return "UNKNOWN";
        }
}

int
```

```
pdf_annot_type_from_string(fz_context *ctx, const char *subtype)
{
	if (!strcmp("Text", subtype)) return PDF_ANNOT_TEXT;
	if (!strcmp("Link", subtype)) return PDF_ANNOT_LINK;
	if (!strcmp("FreeText", subtype)) return PDF_ANNOT_FREE_TEXT;
	if (!strcmp("Line", subtype)) return PDF_ANNOT_LINE;
	if (!strcmp("Square", subtype)) return PDF_ANNOT_SQUARE;
	if (!strcmp("Circle", subtype)) return PDF_ANNOT_CIRCLE;
	if (!strcmp("Polygon", subtype)) return PDF_ANNOT_POLYGON;
	if (!strcmp("PolyLine", subtype)) return PDF_ANNOT_POLY_LINE;
	if (!strcmp("Highlight", subtype)) return PDF_ANNOT_HIGHLIGHT;
	if (!strcmp("Underline", subtype)) return PDF_ANNOT_UNDERLINE;
	if (!strcmp("Squiggly", subtype)) return PDF_ANNOT_SQUIGGLY;
	if (!strcmp("StrikeOut", subtype)) return PDF_ANNOT_STRIKE_OUT;
	if (!strcmp("Redact", subtype)) return PDF_ANNOT_REDACT;
	if (!strcmp("Stamp", subtype)) return PDF_ANNOT_STAMP;
	if (!strcmp("Caret", subtype)) return PDF_ANNOT_CARET;
	if (!strcmp("Image", subtype)) return PDF_ANNOT_IMAGE;
	if (!strcmp("Ink", subtype)) return PDF_ANNOT_INK;
	if (!strcmp("Popup", subtype)) return PDF_ANNOT_POPUP;
	if (!strcmp("FileAttachment", subtype)) return PDF_ANNOT_FILE_ATTACHMENT;
	if (!strcmp("Sound", subtype)) return PDF_ANNOT_SOUND;
	if (!strcmp("Movie", subtype)) return PDF_ANNOT_MOVIE;
	if (!strcmp("RichMedia", subtype)) return PDF_ANNOT_RICH_MEDIA;
	if (!strcmp("Widget", subtype)) return PDF_ANNOT_WIDGET;
	if (!strcmp("Screen", subtype)) return PDF_ANNOT_SCREEN;
	if (!strcmp("PrinterMark", subtype)) return PDF_ANNOT_PRINTER_MARK;
	if (!strcmp("TrapNet", subtype)) return PDF_ANNOT_TRAP_NET;
	if (!strcmp("Watermark", subtype)) return PDF_ANNOT_WATERMARK;
	if (!strcmp("3D", subtype)) return PDF_ANNOT_3D;
	if (!strcmp("Projection", subtype)) return PDF_ANNOT_PROJECTION;
	return PDF_ANNOT_UNKNOWN;
}

case PDF_ANNOT_FREE_TEXT:
	{
		fz_rect text_rect = { 12, 12, 12+300, 12+30 };

	/* Use undocumented Adobe property to match page rotation. */
	int rot = pdf_to_int(ctx, pdf_dict_get_inheritable(ctx, page->obj, PDF_NAME(Rotate)));
	if (rot != 0)
		pdf_dict_put_int(ctx, annot->obj, PDF_NAME(Rotate), rot);

	pdf_set_annot_rect(ctx, annot, text_rect);
	pdf_set_annot_border(ctx, annot, 0);
		pdf_set_annot_default_appearance(ctx, annot, "Helv", 9, nelem(red), red);
	}
	break;

case PDF_ANNOT_STAMP:
	{
		fz_rect stamp_rect = { 12, 12, 12+190, 12+50 };
		pdf_set_annot_rect(ctx, annot, stamp_rect);
		pdf_set_annot_color(ctx, annot, 3, red);
		pdf_set_annot_icon_name(ctx, annot, "Draft");
	}
	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;

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(Image),
        PDF_NAME(Popup),
        PDF_NAME(FileAttachment),
        PDF_NAME(Sound),
        PDF_NAME(Movie),
        PDF_NAME(Widget),
        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(Stamp),
        PDF_NAME(Caret),
        PDF_NAME(Image),
        PDF_NAME(Ink),
        PDF_NAME(FileAttachment),
        PDF_NAME(Sound),
        NULL,
};

if (n > 0)
{
        for (i = 0; i < n; i += 8)
        {
                /* Acrobat draws the line at 1/7 of the box width from the bottom
                 * of the box and 1/16 thick of the box width. */

        h = extract_quad(ctx, quad, qp, i);
        a = lerp_point(quad[LL], quad[UL], 1/37.0f);
        b = lerp_point(quad[LR], quad[UR], 1/37.0f);ddddddddd

        fz_append_printf(ctx, buf, "%g w₩n", h/16);
        fz_append_printf(ctx, buf, "%g %g m₩n", a.x, a.y);
```

```
        fz_append_printf(ctx, buf, "%g %g l₩n", b.x, b.y);
        fz_append_printf(ctx, buf, "S₩n");

        union_quad(rect, quad, h/16);
        }
}

while (x < w)
        {
                x += h/7;
                a = lerp_point(quad[LL], quad[LR], x/w-0.01f);
                if (up)
                {
                        b = lerp_point(quad[UL], quad[UR], x/w-0.01f);
                        c = lerp_point(a, b, 1/17.0f);
                        fz_append_printf(ctx, buf, "%g %g l₩n", c.x, c.y);
                }
                else
                        fz_append_printf(ctx, buf, "%g %g l₩n", a.x, a.y);
                up = !up;
        }
        fz_append_printf(ctx, buf, "S₩n");

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;

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;
}

static void 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;

    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);

    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);
}

    case PDF_ANNOT_IMAGE:
     case PDF_ANNOT_TEXT:
     case PDF_ANNOT_FILE_ATTACHMENT:
```

pdf-font-add.c	case FZ_ADOBE_KOREA: basefont = serif ? "Dotum" : "Batang"; encoding = wmode ? "UniKS-UTF16-V" : "UniKS-UTF16-H"; ordering = "Korea1"; supplement = 2; break;	

pdf-object.c	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 "";
	}	

SumatraPDF.cpp	// Note: duplicated in OnWindowContextMenu because slightly different handling case CmdCreateAnnotText: case CmdCreateAnnotFreeText: case CmdCreateAnnotStamp: case CmdCreateAnnotCaret: case CmdCreateAnnotImage: case CmdCreateAnnotSquare: case CmdCreateAnnotLine:	