JetsonPDF.Tiff

API Reference

Built from scratch against ISO 32000-2 (PDF 2.0)

This document was authored by JetsonPDF.Writer. The cover gradient, the chapter headings, the two-column tables, the navigable bookmarks and the page-number footer are all produced by the same writer the chapters describe. Use the Bookmarks pane to jump between sections.

Generated	2026-06-13 00:10 UTC
Producer	JetsonPDF.Writer
Format	PDF 1.7 with object & cross-reference streams

Contents

One-line entries per feature area. Click an entry in the Bookmarks pane for direct navigation; this page is a flat human-readable index.

1. Decoding TIFFs (TiffImage / TiffFrame) 3
2. Encoding TIFFs (TiffWriter) 4
3. Rasterising PDFs on Windows (PdfToTiffConverter) 5
4. Rasterising PDFs in the browser (PdfToTiffBrowserConverter) 7
5. Progress reporting 9

Decoding TIFFs (TiffImage / TiffFrame)

The codec ships in JetsonPDF.Tiff (netstandard2.0) and has no external dependencies. Decode bytes or a stream into a TiffImage; each frame is RGBA8888 or a JPEG passthrough that re-encodes to PNG on demand for browsers that can't render TIFF.

Type	Description
`JetsonPDF.Tiff.TiffImage`	Eager whole-file decode; exposes Frames as a read-only list.
`TiffImage.Decode(byte[])`	Parse a TIFF buffer.
`TiffImage.Decode(Stream)`	Read a stream into memory and decode.
`image.Frames`	IReadOnlyList<TiffFrame> in IFD order.
`JetsonPDF.Tiff.TiffFrame`	One decoded page: Width/Height, BitsPerSample, Photometric.
`frame.Rgba8888`	Row-major RGBA buffer, or null when this is a JPEG passthrough.
`frame.PassthroughJpeg`	Raw JPEG bytes for single-strip Compression=7 frames; else null.
`frame.EncodeImage()`	Returns PNG bytes (or the original JPEG bytes for passthroughs).
`frame.EncodePng()`	Forces PNG output; throws when the frame is a JPEG passthrough.
`frame.ToDataUri()`	... — drop-in <img src>.

Example

```
using var fs = File.OpenRead("scan.tiff");
var image = TiffImage.Decode(fs);

for (int i = 0; i < image.Frames.Count; i++)
{
    var f = image.Frames[i];
    Console.WriteLine($"page {i + 1}: {f.Width}x{f.Height} " +
                      $"{f.BitsPerSample}-bit {f.Photometric}");
    File.WriteAllBytes($"page{i + 1}.png", f.EncodePng());
}

// Browser-friendly: hand the data URI to an <img>:
string dataUri = image.Frames[0].ToDataUri();
```

Encoding TIFFs (TiffWriter)

TiffWriter takes an enumerable of TiffEncodeFrames (each one width + height + RGBA8888) and chains the IFDs to produce a multi-frame TIFF. Compression and pixel format are controlled by TiffWriteOptions. The writer is fully managed - no GDI+ or System.Drawing.

Type	Description
`JetsonPDF.Tiff.TiffWriter`	Static encoder. Three Encode overloads.
`TiffWriter.Encode(frames, options)`	Returns byte[]; one IFD per frame.
`TiffWriter.Encode(output, frames, options)`	Writes to a Stream; stream is not closed.
`TiffWriter.Encode(w, h, rgba, options)`	Convenience single-frame overload.
`JetsonPDF.Tiff.TiffEncodeFrame`	Input frame DTO: Width, Height, Rgba8888.
`TiffEncodeFrame.From(TiffFrame)`	Re-encode a decoded frame (e.g. recompress a passthrough).
`JetsonPDF.Tiff.TiffWriteOptions`	Compression, PixelFormat, BlackAndWhiteThreshold, DpiX/Y, RowsPerStrip.
`JetsonPDF.Tiff.TiffCompression`	None, PackBits, Deflate, AdobeDeflate (writer-supported).
`JetsonPDF.Tiff.TiffEncodePixelFormat`	Rgb, Rgba, Grayscale, BlackAndWhite.

Example

```
var page1 = new TiffEncodeFrame(width: 850, height: 1100, rgba8888: bufA);
var page2 = new TiffEncodeFrame(width: 850, height: 1100, rgba8888: bufB);

byte[] tiff = TiffWriter.Encode(new[] { page1, page2 },
    new TiffWriteOptions
    {
        Compression  = TiffCompression.Deflate,
        PixelFormat  = TiffEncodePixelFormat.Rgb,
        DpiX = 200, DpiY = 200,
        RowsPerStrip = 64,
    });

File.WriteAllBytes("output.tiff", tiff);
```

Rasterising PDFs on Windows (PdfToTiffConverter)

JetsonPDF.PdfToTiffConverter (net8.0-windows) drives the WPF render pipeline: each page is converted to XAML via PdfToXamlConverter, parsed, measured/arranged at the page's MediaBox in DIPs, captured to a RenderTargetBitmap at the chosen DPI, and the RGBA bytes are fed to the managed TiffWriter. STA-only.

Type	Description
`JetsonPDF.Tiff.PdfToTiffConverter`	Static; Windows-only (STA). Four entry points.
`ConvertToFile(pdfPath, tiffPath, options?, progress?)`	Synchronous; loads the PDF then writes the TIFF.
`ConvertToFileAsync(pdfPath, tiffPath, ...)`	Async variant - same behaviour.
`ConvertAsync(ReadDocument, Stream, options?, progress?)`	Encodes a parsed document to any writable stream.
`RenderPageAsync(ReadPage, options?)`	Returns a Pbgra32 BitmapSource for one page (no TIFF involved).
`JetsonPDF.Tiff.PdfToTiffOptions`	Init-only options: Dpi, Compression, ColorMode, Pages, ...
`opts.Dpi (default 150)`	Output raster resolution.
`opts.Compression`	TiffCompression - defaults to Deflate.
`opts.ColorMode + BlackAndWhiteThreshold`	TiffColorMode (Color / Grayscale / BlackAndWhite) + 0..255 threshold.
`opts.MultipageStrategy`	SingleMultiPage (default) or OneTiffPerPage.
`opts.Pages`	1-based page numbers to include. Null = all.
`opts.BackgroundArgb`	ARGB hex (default "#FFFFFFFF"); null disables the fill.
`opts.RowsPerStrip / opts.XamlOptions`	Strip layout knob; forwarded PdfToXamlOptions.
`PdfToTiffOptions.ParsePageRange("1-3,5,7-9")`	Static helper; expands a spec to a 1-based int[].
`JetsonPDF.Tiff.TiffColorMode`	Color, Grayscale, BlackAndWhite.
`JetsonPDF.Tiff.TiffMultipageStrategy`	SingleMultiPage, OneTiffPerPage (file pattern: '-pNNN' insertion).

Rasterising PDFs on Windows (PdfToTiffConverter) (

Example

```
// Synchronous (call from an STA thread, e.g. WPF UI):
PdfToTiffConverter.ConvertToFile(
    pdfPath:  "report.pdf",
    tiffPath: "report.tiff",
    options: new PdfToTiffOptions
    {
        Dpi          = 200,
        ColorMode    = TiffColorMode.Grayscale,
        Compression  = TiffCompression.Deflate,
        Pages        = PdfToTiffOptions.ParsePageRange("1-3,5"),
    });

// Async to a Stream (already have a parsed document):
using var fs = File.Create("report.tiff");
await PdfToTiffConverter.ConvertAsync(doc, fs);

// Render one page to a WPF BitmapSource (no TIFF involved):
BitmapSource bmp = await PdfToTiffConverter.RenderPageAsync(doc.Pages[0]);
```

Rasterising PDFs in the browser (PdfToTiffBrowserCo

JetsonPDF.OpenSilver ships a browser-side rasteriser that mounts each PDF page into a live OpenSilver host panel, snapshots the resulting DOM via html2canvas (JS interop), reads RGBA back through CanvasRenderingContext2D.getImageData, and composes a multi-page TIFF via the managed TiffWriter. No server round-trip, no native image library.

Type	Description
`JetsonPDF.OpenSilver.PdfToTiffBrowserConverter`	Static; ConvertAsync is the single entry point.
`ConvertAsync(byte[] pdfBytes, Panel host, options?, progress?)`	Task<byte[]> — host must already be parented in the live visual tree.
`Panel host`	Live, initially empty Panel used as scratch container; must be in Application.Current.
`options : TiffWriteOptions`	Same options DTO as the codec writer (Compression / PixelFormat / DPI / ...).
`progress : IProgress<TiffConversionProgress>`	Stage-weighted progress; ConvertingXaml ? Rendering ? Encoding ? Completed.
`JetsonPDF.OpenSilver.TiffViewer`	UserControl that decodes a TIFF and displays each frame as a base64-PNG <Image>.
`viewer.Source`	byte[] or Stream of TIFF bytes.
`viewer.Page`	int? — single frame index, or null to stack every page.
`viewer.FitToWidth / viewer.PageCount`	Layout toggle + readonly decoded frame count.
`JetsonPDF.OpenSilver.TiffViewerSource`	Static helpers: ToBytes(object?), TryDecode(bytes, out image, out error).

Rasterising PDFs in the browser (PdfToTiffBrowserC

Example

```
// Caller XAML: <Grid x:Name="ScratchHost" Visibility="Collapsed"/>

var pdfBytes = await File.ReadAllBytesAsync("report.pdf");
var progress = new Progress<TiffConversionProgress>(p =>
    statusText.Text = p.Description);

byte[] tiff = await PdfToTiffBrowserConverter.ConvertAsync(
    pdfBytes,
    host: ScratchHost,
    options: new TiffWriteOptions
    {
        Compression = TiffCompression.Deflate,
        PixelFormat = TiffEncodePixelFormat.Rgb,
    },
    progress: progress);

viewer.Source = tiff;          // <jp:TiffViewer x:Name="viewer"/>
```

Progress reporting

Both converters report progress through the same TiffConversionProgress payload, so a single Progress<T> sink works for either pipeline. Fraction is stage-weighted (xaml conversion is fastest, encoding is cheap, rendering dominates) and Description is a human-readable status string bindable to a label.

Type	Description
`JetsonPDF.Tiff.TiffConversionProgress`	readonly struct payload.
`new(completed, total, stage, currentPage)`	Full constructor.
`new(completed, total)`	Legacy two-arg ctor; stage inferred.
`p.CompletedPages / TotalPages / CurrentPage`	Whole-page counters.
`p.Stage`	Phase enum: Starting / ConvertingXaml / Rendering / Encoding / Completed.
`p.Fraction`	Stage-weighted [0,1] for progress bars.
`p.Description`	Bindable status text (e.g. "Rendering page 3 of 12").
`JetsonPDF.Tiff.TiffConversionStage`	Starting, ConvertingXaml, Rendering, Encoding, Completed.

Example

```
var progress = new Progress<TiffConversionProgress>(p =>
{
    progressBar.Value      = p.Fraction;
    progressLabel.Content  = p.Description;
});

await PdfToTiffConverter.ConvertToFileAsync(
    "in.pdf", "out.tiff",
    options: null,
    progress: progress);
```