JetsonPDF.OpenSilver

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.

Reading: PdfToXamlConverter (OpenSilver)

Same shared converter as the WPF integration, compiled under OPENSILVER into the JetsonPDF.OpenSilver namespace and targeting netstandard2.0. Produces the same XAML string; OpenSilver's WebAssembly runtime renders it through the browser DOM. Standard WPF controls and Path/TextBlock/Image work; the small set of OpenSilver-unsupported primitives is intercepted by the emitter.

Type	Description
`JetsonPDF.OpenSilver.PdfToXamlConverter`	Static, identical shape to the WPF surface.
`ConvertAsync(ReadDocument, options?)`	Task<string>; <StackPanel> of <Canvas> pages.
`ConvertPageAsync(ReadPage, options?)`	Task<string>; one standalone <Canvas>.
`JetsonPDF.OpenSilver.PdfToXamlOptions`	Same FontSubstitution / ShowFormFields / HideContentUnderWidgets surface.
`JetsonPDF.OpenSilver.Base64ImageExtension`	{jetsonpdf:Base64Image Data='...'} — wraps payload in data: URI for BitmapImage.
`JetsonPDF.OpenSilver.StandardFontMap`	Shared Standard-14 ? FontFamily map.

Example

```
var doc  = JetsonPDF.Reading.Reader.Load(pdfBytes);
string xaml = await PdfToXamlConverter.ConvertAsync(doc);

var root = (FrameworkElement)XamlReader.Load(xaml);
pageHost.Content = root;
```

Authoring: XamlToPdfConverter (OpenSilver, async)

JetsonPDF.OpenSilver.Authoring runs the same authoring pipeline as the WPF converter, but async — OpenSilver loads images via the browser asynchronously so the converter awaits before walking the tree. Drives the same PaginatedTable re-parse loop for overflow pagination. Requires a hosted Panel for accurate layout in the Wasm runtime.

Type	Description
`JetsonPDF.OpenSilver.Authoring.XamlToPdfConverter`	Static. Two ConvertAsync overloads.
`ConvertAsync(string xaml, options?)`	Task<byte[]>; the rendered PDF bytes.
`ConvertAsync(string xaml, Stream, options?)`	Writes the PDF bytes to a Stream.
`options : JetsonPDF.XamlAuthoring.XamlToPdfConverterOptions`	Shared converter options (DefaultPageSize, Title, Author, fonts).
`JetsonPDF.OpenSilver.Authoring.OpenSilverTreeWalker`	IXamlTreeWalker implementation; set HostContainer for live Wasm hosting.
`OpenSilverTreeWalker.HostContainer`	Static Panel slot for production hosts; null in detached/test scenarios.
`JetsonPDF.OpenSilver.Authoring.XamlDocument / XamlPage`	Same shared authoring types as WPF, namespaced under OpenSilver.
`JetsonPDF.OpenSilver.Authoring.PaginatedTable`	Identical XAML conformance as WPF's authoring dialect.
`JetsonPDF.OpenSilver.Authoring.Form`	Attached props: Form.FieldName / MaxLength / IsMultiline / IsPassword / Action.
`JetsonPDF.OpenSilver.Authoring.PageNumberExtension`	{jsonpdf:PageNumber} / {jsonpdf:PageCount} resolved per page.

Example

```
// host XAML: <Canvas x:Name="AuthoringHost" Visibility="Collapsed"/>
OpenSilverTreeWalker.HostContainer = AuthoringHost;

// xaml: the authoring document markup as a string -
// typically loaded from disk or an embedded resource.
string xaml = await ResourceLoader.LoadXamlAsync("hello.xaml");

byte[] pdf = await XamlToPdfConverter.ConvertAsync(xaml);
```

Browser TIFF viewer + converter

OpenSilver ships a managed TIFF stack for the browser: the codec from JetsonPDF.Tiff decodes/encodes via netstandard2.0 (no native libs), TiffViewer is a UserControl that presents decoded frames as base64-PNG <Image>s, and PdfToTiffBrowserConverter rasterises a PDF in the browser via html2canvas. End-to-end PDF ? TIFF in Wasm.

Type	Description
`JetsonPDF.OpenSilver.TiffViewer`	UserControl. Wire Source = byte[] | Stream.
`viewer.Source / Page / FitToWidth / PageCount`	DPs: source bytes, optional single-page index, fit toggle, decoded count.
`JetsonPDF.OpenSilver.TiffViewerSource`	Static helpers: ToBytes(object?), TryDecode(bytes, out image, out error).
`JetsonPDF.OpenSilver.PdfToTiffBrowserConverter`	Static. ConvertAsync(byte[], Panel host, options?, progress?) ? Task<byte[]>.
`TiffWriteOptions / TiffCompression / TiffEncodePixelFormat`	Shared with the codec; same DTO.
`IProgress<TiffConversionProgress>`	Stage-weighted progress for the multi-page pipeline.

Example

```
// XAML:
//   <jp:TiffViewer x:Name="Viewer"
//                  Source="{Binding TiffBytes}"
//                  FitToWidth="True"/>
//   <Grid x:Name="RasterHost" Visibility="Collapsed"/>

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

Viewer.Source = tiff;
```

JPEG / JPX decoder hooks

OpenSilver has no built-in JPEG 2000 decoder and a JPEG decoder that can't expose pixels for SMask compositing. Implementers plug in their own pixel decoders via two small interfaces - typically wrapping ImageSharp or SkiaSharp. Without them the page still renders, but JPEG+SMask alpha is dropped and JPX images fall back to passthrough.

Type	Description
`JetsonPDF.OpenSilver.ImageDecoders`	Static slots set on app startup.
`ImageDecoders.Jpeg`	IJpegPixelDecoder? — defaults to DefaultBrowserJpegDecoder (createImageBitmap-backed).
`ImageDecoders.Jpx`	IJpxPixelDecoder? — null until consumer wires one in.
`JetsonPDF.OpenSilver.IJpegPixelDecoder`	TryDecodeRgb24Async(byte[], CancellationToken) ? Task<JpegPixelResult?>.
`JetsonPDF.OpenSilver.IJpxPixelDecoder`	Same shape for JPEG 2000.
`JetsonPDF.OpenSilver.JpegPixelResult`	readonly struct: Rgb (byte[] RGB24), Width, Height.
`DefaultBrowserJpegDecoder`	Built-in JPEG decoder bridging to browser createImageBitmap via JS interop.

Example

```
// App startup (Wasm host or Simulator):
ImageDecoders.Jpeg = new ManagedJpegDecoder(); // your wrapper
ImageDecoders.Jpx  = new ManagedJpxDecoder();  // your wrapper

public sealed class ManagedJpegDecoder : IJpegPixelDecoder
{
    public Task<JpegPixelResult?> TryDecodeRgb24Async(byte[] bytes,
        CancellationToken ct = default)
    {
        using var ms = new MemoryStream(bytes);
        var img = Image.Load<Rgb24>(ms);    // ImageSharp
        var buf = new byte[img.Width * img.Height * 3];
        img.CopyPixelDataTo(buf);
        return Task.FromResult<JpegPixelResult?>(
            new JpegPixelResult(buf, img.Width, img.Height));
    }
}
```

Widget actions (browser dispatch)

JetsonPDF.OpenSilver.WidgetActions is the browser-targeted twin of the WPF dispatcher. Same attached-property surface (Action / Enabled / ActionInvokedEvent), but the defaults bridge to the browser: URI actions call window.open, Named('Print') calls window.print, ResetForm walks the subtree and clears input controls.

Type	Description
`JetsonPDF.OpenSilver.WidgetActions`	Attached-property dispatcher; mirror of the WPF surface.
`WidgetActions.ActionProperty`	string payload serialised by the emitter.
`WidgetActions.EnabledProperty`	Set True on an ancestor to walk descendants and bind Click handlers.
`WidgetActions.ActionInvokedEvent`	RoutedEvent raised before default handling - set Handled=true to suppress.
`Default URI behaviour`	window.open via OpenSilver Interop (popup blockers may intervene).
`Default Named("Print") behaviour`	window.print via OpenSilver Interop.
`Default ResetFormAction behaviour`	Clears descendant text / checkbox / combo / list values.

Example

```
var root = (FrameworkElement)XamlReader.Load(xaml);
WidgetActions.SetEnabled(root, true);

// Customise dispatch (e.g. integrate with an app-level toast):
root.AddHandler(WidgetActions.ActionInvokedEvent,
    new RoutedEventHandler((s, e) =>
    {
        if (e.OriginalSource is FrameworkElement fe)
        {
            string? payload = WidgetActions.GetAction(fe);
            // SubmitForm, JavaScript, GoTo - your handler:
            HandleAction(payload);
            e.Handled = true;     // suppress default
        }
    }));
```