JetsonPDF.Wpf

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

JetsonPDF.Wpf turns a parsed PDF into a XAML string a WPF host can drop into any Canvas/StackPanel slot. The converter is filesystem-free: images are inlined as base64 and resolved by the JetsonPDF Base64Image markup extension; text becomes vector paths (no font cache, no .ttf disk lookups). Async for parity with the OpenSilver pipeline.

Type	Description
`JetsonPDF.Wpf.PdfToXamlConverter`	Static. ConvertAsync / ConvertPageAsync.
`ConvertAsync(ReadDocument, options?)`	Task<string>; returns a <StackPanel> of per-page <Canvas>es.
`ConvertPageAsync(ReadPage, options?)`	Task<string>; returns a single standalone <Canvas>.
`JetsonPDF.Wpf.PdfToXamlOptions`	FontSubstitution / ShowFormFields / HideContentUnderWidgets.
`opts.FontSubstitution`	Dictionary<string,string> from PDF /BaseFont to WPF FontFamily.
`opts.ShowFormFields (default true)`	Emit AcroForm widgets as live WPF input controls.
`opts.HideContentUnderWidgets (default true)`	Clip baked widget chrome under live widgets to avoid double-borders.
`JetsonPDF.Wpf.StandardFontMap`	PDF Standard-14 to WPF FontFamily map used by both converters.
`JetsonPDF.Wpf.Base64ImageExtension`	{jetsonpdf:Base64Image Data='...'} - inline image, no I/O.

Example

```
var doc = JetsonPDF.Reading.Reader.Load("invoice.pdf");
string xaml = await PdfToXamlConverter.ConvertAsync(doc,
    new PdfToXamlOptions
    {
        ShowFormFields           = true,
        HideContentUnderWidgets  = true,
        FontSubstitution         = { ["Helvetica"] = "Segoe UI" },
    });

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

Authoring: XamlToPdfConverter (XAML to PDF)

JetsonPDF.Wpf.Authoring is the inverse pipeline: a XamlReader-parsed authoring tree is Measure/Arrange'd by WPF, then VisualTreeEmitter walks the arranged visuals and emits PDF drawing operators. Grid/StackPanel/DockPanel layout, Brushes, fonts, transforms all come for free from WPF. STA-required.

Type	Description
`JetsonPDF.Wpf.Authoring.XamlToPdfConverter`	Static. Two Convert overloads.
`Convert(string xaml, options?)`	Returns byte[] of the rendered PDF.
`Convert(string xaml, Stream, options?)`	Writes the PDF to a Stream.
`JetsonPDF.Wpf.Authoring.XamlToPdfOptions`	DefaultPageSize, Title, Author, font defaults, RasterizeEffectsDpi.
`opts.DefaultPageSize`	Falls back to Letter; ignored when the root sets explicit Width/Height.
`opts.Title / opts.Author`	Stamped into /Info /Title and /Info /Author.
`opts.DefaultFontFamily / DefaultFontSizeDip`	Used when a <TextBlock> omits the attributes. 16 DIP = 12 pt.
`opts.RasterizeEffectsDpi (default 192)`	Rasterisation density for subtrees with UIElement.Effect (no PDF analogue).
`JetsonPDF.Wpf.Authoring.VisualTreeEmitter`	The walker that turns the arranged visual tree into PDF operators. Public for extension.

Example

```
// xaml: the literal XAML markup of an authoring document
// (typically loaded from disk, an embedded resource, or a
// string constant); a <jetsonpdf:XamlDocument> root with
// a single panel child is the common shape.
string xaml = File.ReadAllText("hello.xaml");

byte[] pdf = XamlToPdfConverter.Convert(xaml,
    new XamlToPdfOptions { Title = "Hello", Author = "Acme" });
File.WriteAllBytes("hello.pdf", pdf);
```

XamlDocument, XamlPage, page setup

XamlDocument is a ContentControl; drop one panel into its content for single-page documents, or populate Pages with XamlPages for multi-page. Per-page PageSize/Landscape/PageWidth/PageHeight override the document defaults. The converter sets JetsonPageContext on each page's DataContext so {jetsonpdf:PageNumber} / {jetsonpdf:PageCount} resolve to the right values.

Type	Description
`JetsonPDF.Wpf.Authoring.XamlDocument`	Root <jetsonpdf:XamlDocument>; ContentControl with PDF metadata.
`doc.PageSize`	JetsonPageSize enum: Letter (default), Legal, A3/A4/A5, Tabloid.
`doc.Landscape`	Swaps width and height. Ignored under a custom size.
`doc.PageWidth / doc.PageHeight`	Both NaN by default; set both to override PageSize/Landscape.
`doc.Title / doc.Author`	/Info entries; also surfaced via XamlToPdfOptions.
`doc.Pages`	Collection<XamlPage> for explicit multi-page mode.
`doc.Outline / doc.NamedDestinations / doc.PageLabels / doc.Layers`	Catalog-level collections (outline tree, named dests, page labels, OCGs).
`doc.Conformance`	PDF/A or PDF/UA conformance flags forwarded to the writer.
`JetsonPDF.Wpf.Authoring.XamlPage`	<jetsonpdf:XamlPage> with optional per-page size overrides.
`page.PageSize / Landscape / PageWidth / PageHeight`	Per-page overrides; unset = inherit from the document.
`JetsonPDF.Wpf.Authoring.JetsonPageSize`	Letter, Legal, Tabloid, A3, A4, A5.
`JetsonPDF.Wpf.Authoring.JetsonPageContext`	DataContext wrapper exposing PageNumber / PageCount per page.

XamlDocument, XamlPage, page setup (continued)

Example

```
<jetsonpdf:XamlDocument PageSize="Letter" Author="Acme">
  <jetsonpdf:XamlDocument.Pages>
    <jetsonpdf:XamlPage>
      <StackPanel Margin="36">
        <TextBlock FontSize="28" Text="Cover" />
      </StackPanel>
    </jetsonpdf:XamlPage>
    <jetsonpdf:XamlPage Landscape="True">
      <Grid Margin="36">
        <TextBlock FontSize="14"
                   Text="{Binding PageNumber, StringFormat='Page {0}'}"/>
      </Grid>
    </jetsonpdf:XamlPage>
  </jetsonpdf:XamlDocument.Pages>
</jetsonpdf:XamlDocument>
```

PaginatedTable

PaginatedTable is a FrameworkElement that draws a header row plus N data rows bound to ItemsSource. When the row count exceeds the arranged slot, the converter re-parses the XAML once per overflow slice and sets SliceStart/SliceCount so the header repeats and rows distribute - so {jetsonpdf:PageNumber} bindings stay live on every page.

Type	Description
`JetsonPDF.Wpf.Authoring.PaginatedTable`	<jetsonpdf:PaginatedTable>; row-flowing tabular control.
`table.ItemsSource`	IEnumerable? of data rows.
`table.Columns`	Collection<PaginatedColumn>; per-column width + header text + binding path.
`table.HeaderHeight / RowHeight`	Doubles; defaults 24 and 20.
`table.HeaderBackground / HeaderBorderBrush / HeaderBorderThickness`	Header styling.
`table.CellBorderBrush / CellBorderThickness / CellPadding`	Cell styling.
`table.AlternatingRowBackground`	Optional even-row fill.
`JetsonPDF.Wpf.Authoring.PaginatedColumn`	Column definition (header, width, content binding).
`JetsonPDF.Wpf.Authoring.Pagination`	Attached prop: Pagination.HideOnOverflow="True" hides chrome on slices >0.

Example

```
<jetsonpdf:PaginatedTable ItemsSource="{Binding LineItems}"
                          RowHeight="22" HeaderHeight="26">
  <jetsonpdf:PaginatedTable.Columns>
    <jetsonpdf:PaginatedColumn Header="Item"  Width="*"  Binding="{Binding Name}"/>
    <jetsonpdf:PaginatedColumn Header="Qty"   Width="60" Binding="{Binding Qty}"/>
    <jetsonpdf:PaginatedColumn Header="Price" Width="80" Binding="{Binding Price, StringFormat=c}"/>
  </jetsonpdf:PaginatedTable.Columns>
</jetsonpdf:PaginatedTable>
```

AcroForm widgets, page numbers & navigation

Authoring XAML opts into AcroForm output via jetsonpdf:Form.* attached properties on standard input controls. {jetsonpdf:PageNumber}/{PageCount} markup extensions resolve per page (one-way bindings to the per-page JetsonPageContext). Document-level outline, named destinations and page labels live on XamlDocument.

Type	Description
`JetsonPDF.Wpf.Authoring.Form`	Attached props: FieldName, MaxLength, IsMultiline, IsPassword, Action.
`jetsonpdf:Form.FieldName="name" on TextBox`	Becomes an AcroForm /Tx widget at the control's arranged bounds.
`jetsonpdf:Form.FieldName on CheckBox / ComboBox / ListBox / Button`	Becomes /Btn (check / push), /Ch (combo / list).
`JetsonPDF.Wpf.Authoring.PageNumberExtension`	{jetsonpdf:PageNumber} - one-way binding to PageContext.PageNumber.
`JetsonPDF.Wpf.Authoring.PageCountExtension`	{jetsonpdf:PageCount} - one-way binding to PageContext.PageCount.
`doc.Outline + XamlOutlineItem`	Bookmark tree; each item carries Title + PageIndex or NamedDestination + Bold/Italic/IsExpanded.
`doc.NamedDestinations + NamedDestination`	Name + PageIndex + Mode (FitPage / FitHorizontal / FitVertical / Xyz) + Left/Top/Zoom.
`doc.PageLabels + XamlPageLabel`	Roman / alpha / decimal label ranges with optional prefix.
`doc.Layers + XamlLayer`	Optional content groups (OCGs) declared up-front; reference by name.

Example

```
<Grid Margin="36">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>

  <TextBox  Grid.Row="0" Width="260" Height="22"
            jetsonpdf:Form.FieldName="name"
            jetsonpdf:Form.MaxLength="64"/>

  <TextBlock Grid.Row="1" HorizontalAlignment="Right"
             Text="{jetsonpdf:PageNumber}" />
</Grid>
```

Display infrastructure (Base64ImageExtension, Widge

Base64ImageExtension materialises inline base64 image payloads into a BitmapImage at parse time - no data: URI plumbing needed. WidgetActions is an attached behaviour that dispatches PDF widget actions emitted by PdfToXamlConverter: URI / Named('Print') / ResetForm work with the defaults; SubmitForm / JavaScript bubble for consumer code.

Type	Description
`JetsonPDF.Wpf.Base64ImageExtension`	{jetsonpdf:Base64Image Data='...'} ? frozen BitmapImage.
`ext.Data`	Raw base-64 (no data: prefix, no MIME).
`JetsonPDF.Wpf.WidgetActions`	Attached-property dispatcher for emitted widgets.
`WidgetActions.ActionProperty`	string payload set by the emitter (serialised WidgetAction).
`WidgetActions.EnabledProperty`	Set to True on an ancestor to enable Click dispatch.
`WidgetActions.ActionInvokedEvent`	RoutedEvent raised before default handling - mark Handled=true to suppress.
`JetsonPDF.Wpf.ResourceCache`	Internal scratch used by the emitter (image dedup, glyph path cache). Public surface limited.

Example

```
// 1) Inline-image markup extension:
// <Image Source="{jetsonpdf:Base64Image Data='iVBORw0KG...'}" />

// 2) Enable widget dispatch on the host control:
var root = (FrameworkElement)XamlReader.Parse(xaml);
WidgetActions.SetEnabled(root, true);

// 3) Override the default for a custom action:
root.AddHandler(WidgetActions.ActionInvokedEvent,
    new RoutedEventHandler((s, e) =>
    {
        // e.OriginalSource = the clicked element
        // Read WidgetActions.GetAction(element) for the payload
        // Set e.Handled = true to suppress the default behaviour
    }));
```