

JetsonPDF.Writer

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 2026-04-30 17:10 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. Document model & writing 3
- 2. Text & fonts 4
- 3. Graphics & paths 5
- 4. Color spaces 6
- 5. Images 7
- 6. Forms (AcroForm widgets) 8
- 7. Annotations 9
- 8. Encryption & signatures 10
- 9. Conformance: PDF/A & PDF/UA 11
- 10. Structure tree & accessibility 12
- 11. Measurements & viewports 13
- 12. Incremental updates, linearization & associated files 14

Document model & writing

Everything starts with a PdfDocument: configure metadata, page layout, optional content groups and conformance flags, then call Save once content is laid down. Object streams, linearization and incremental updates are first-class.

| Type | Description |
|--------------------------------|--|
| JetsonPDF.PdfDocument | Top-level document. Add pages, set Title/Author/PdfVersion. |
| JetsonPDF.PdfPage | One page: drawing surface plus structure tags and resources. |
| JetsonPDF.PageSize | Letter/Legal/Tabloid/A3/A4/A5 plus arbitrary (w,h) records. |
| JetsonPDF.PdfPageMode | Catalog /PageMode hint: UseOutlines, UseThumbs, FullScreen. |
| JetsonPDF.PdfViewerPreferences | Initial viewer state: DisplayDocTitle, CenterWindow, ... |
| doc.UseObjectStreams = true | Cross-reference and object streams (PDF 1.5+ compression). |
| doc.AddPage(PageSize) | Append a page; returns the PdfPage drawing surface. |
| doc.Save(path stream) | Serialize the document. Throws if zero pages. |

Example

```
var doc = new PdfDocument
{
    Title = "Hello",
    PdfVersion = "2.0",
    UseObjectStreams = true,
    PageMode = PdfPageMode.UseOutlines,
};

var page = doc.AddPage(PageSize.Letter);
page.DrawText("Hello, PDF!",
    new PdfFont(FontFamily.Helvetica, 22), x: 36, y: 720);

doc.Save("hello.pdf");
```

Text & fonts

Standard 14 fonts cost no bytes and need no embedding. Embedded TrueType ships as Identity-H composites with subsetting; OpenType-CFF fonts are CharString-pruned. Type 3 lets you paint glyphs as content streams.

| Type | Description |
|---|--|
| <code>JetsonPDF.PdfFont</code> | Wrapper for Standard 14 or embedded face + size + style. |
| <code>JetsonPDF.FontFamily</code> | Helvetica, TimesRoman, Courier, Symbol, ZapfDingbats. |
| <code>JetsonPDF.FontStyle</code> | Regular Bold Italic flags (combinable). |
| <code>JetsonPDF.EmbeddedFontFace</code> | Load a TTF/OTF from disk (or bytes) for embedding. |
| <code>JetsonPDF.PdfType3Font</code> | Procedural font; each glyph is a content stream. |
| <code>page.DrawText(...)</code> | Single-line text at (x,y) with optional colour. |
| <code>page.DrawTextBlock(...)</code> | Word-wrapped paragraph; returns line count. |
| <code>page.DrawTextWithActualText(...)</code> | Marked-content /Span with /ActualText for ligatures. |

Example

```
var heading = new PdfFont(FontFamily.Helvetica, 18, FontStyle.Bold);
var body    = new PdfFont(FontFamily.TimesRoman, 11);

page.DrawText("Section title", heading, x: 36, y: 720);
page.DrawTextBlock("Lorem ipsum dolor sit amet ...", body,
    x: 36, y: 700, width: 540, lineHeight: 14);

var arial = EmbeddedFontFace.FromFile(@"C:\Windows\Fonts\arial.ttf");
page.DrawText("Embedded TrueType",
    new PdfFont(arial, 14), x: 36, y: 660);
```

Graphics & paths

Vector primitives plus arbitrary PdfPath build-up. Shadings, tiling patterns and transparency groups round out the §8 surface. The chapter below renders a live axial shading drawn by the writer on this page.

Live demo: an axial shading drawn by the writer on this page.



| Type | Description |
|------------------------------------|--|
| JetsonPDF.PdfPath | Move/Line/Curve/Rect builder with NonZero or EvenOdd fill. |
| JetsonPDF.PdfShading.Axial(...) | Two- or multi-stop linear gradient via Type 3 stitching. |
| JetsonPDF.PdfShading.Radial(...) | Concentric-circle gradient (focal-point capable). |
| JetsonPDF.PdfTilingPattern | Coloured (PaintType 1) or uncoloured (Type 2) pattern. |
| JetsonPDF.PdfTransparencyGroup | Page /Group with isolation, knockout and blend mode. |
| page.DrawRectangle(...) | Axis-aligned rect with optional fill / stroke / width. |
| page.DrawPath(path, fill, stroke) | Render a built PdfPath. |
| page.FillRectangleWithShading(...) | Apply a PdfShading inside a rect (sh operator). |

Example

```
page.DrawRectangle(36, 700, 200, 40,
    fill: PdfColor.Rgb(0.13, 0.31, 0.52));

var triangle = new PdfPath()
    .MoveTo(60, 660).LineTo(160, 660).LineTo(110, 580).Close();
page.DrawPath(triangle, fill: PdfColor.Rgb(0.85, 0.30, 0.20));

var grad = PdfShading.Axial(36, 0, 380, 0,
    PdfColor.Rgb(0.13, 0.31, 0.52),
    PdfColor.Rgb(0.85, 0.30, 0.20));
page.FillRectangleWithShading(36, 540, 344, 24, grad);
```

Color spaces

Beyond the device families, JetsonPDF speaks every §8.6 colour space: CIE-based Cal/Lab, ICC-tagged custom profiles, Separation spot inks and DeviceN multi-ink compositions. Build a PdfColorSpace once, then reference it from any PdfColor.

| Type | Description |
|-------------------------------|---|
| JetsonPDF.PdfColor | Gray/RGB/CMYK + ICC + Separation/DeviceN value type. |
| PdfColor.Rgb / Gray / Cmyk | Static factories for the device families. |
| JetsonPDF.PdfColorSpace | Device*, Cal*, Lab, ICC, Separation, DeviceN, NChannel. |
| JetsonPDF.PdfIccProfile | Embed an ICC v2/v4 profile as a colour space. |
| PdfColorSpace.Separation(...) | Single-ink spot colour with a tint-transform function. |
| PdfColorSpace.DeviceN(...) | Multi-component process + spot space (NChannel ready). |

Example

```
var srgb = PdfIccProfile.FromFile("sRGB.icc");
var srgbSpace = PdfColorSpace.Icc(srgb);

var brand = PdfColor.FromColorSpace(srgbSpace, 0.13, 0.31, 0.52);
page.DrawRectangle(36, 700, 220, 40, fill: brand);

var pantone = PdfColorSpace.Separation("PANTONE 286 C",
    PdfColor.Rgb(0.13, 0.31, 0.52));
page.DrawRectangle(36, 640, 220, 40,
    fill: PdfColor.FromColorSpace(pantone, 1.0));
```

Images

JPEG passthrough, full-bit-depth PNG (1/2/4/8/16-bit + Adam7), CCITT G3 and G4, JBIG2 with arithmetic and Huffman paths, JPEG 2000 with JP2 metadata extraction. ICC profiles and image masks are first-class citizens.

| Type | Description |
|---|--|
| <code>JetsonPDF.PdfImage.FromJpeg(...)</code> | JPEG with native /DCTDecode passthrough. |
| <code>JetsonPDF.PdfImage.FromPng(...)</code> | PNG decoder honouring tRNS, alpha and Adam7. |
| <code>JetsonPDF.PdfImage.FromFile(...)</code> | Sniffs the format and dispatches to FromJpeg/FromPng. |
| <code>JetsonPDF.PdfImageMask</code> | 1-bpp stencil for hard-edge transparency. |
| <code>JetsonPDF.PdfFormXObject</code> | Reusable vector content - draw on many pages, deduped. |
| <code>page.DrawImage(img, x, y, w, h)</code> | Paint at arranged rectangle with cm matrix. |

Example

```
var jpeg = PdfImage.FromFile("photo.jpg");
double aspect = (double)jpeg.PixelHeight / jpeg.PixelWidth;
page.DrawImage(jpeg, x: 36, y: 540,
    width: 220, height: 220 * aspect);

byte[] pngBytes = File.ReadAllBytes("logo.png");
var png = PdfImage.FromPng(pngBytes);
page.DrawImage(png, x: 296, y: 540, width: 130, height: 130);
```

Forms (AcroForm widgets)

Every interactive widget §12.7 specifies, with a baked /AP appearance and the modern /AA action grab-bag. Signature fields support /SV seed values and /Lock restrictions; barcode fields ride a private appearance pipeline.

| Type | Description |
|-----------------------------------|--|
| JetsonPDF.Forms.PdfTextField | Plain or rich text input; multiline, password, comb. |
| JetsonPDF.Forms.PdfCheckBox | On/Off toggle with named appearance state. |
| JetsonPDF.Forms.PdfRadioGroup | Mutually exclusive group of PdfRadioButton kids. |
| JetsonPDF.Forms.PdfComboBox | Drop-down (editable or fixed) backed by /Opt array. |
| JetsonPDF.Forms.PdfListBox | Multi-select list with /TI top-index scroll. |
| JetsonPDF.Forms.PdfPushButton | Action-trigger button; URI/Named/JS/Submit/Reset. |
| JetsonPDF.Forms.PdfSignatureField | Digital sig field with /SV + /Lock + DocMDP. |
| JetsonPDF.Forms.PdfBarcodeField | Matrix barcode baked into the widget appearance. |

Example

```
var fnt = new PdfFont(FontFamily.Helvetica, 10);

var name = page.AddTextField("name", 36, 700, 260, 22, fnt);
name.Value = "Jane Q. Tester";
name.MaxLength = 64;

var subscribe = page.AddCheckBox("subscribe", 320, 700, 14, 14);
subscribe.IsChecked = true;

var print = page.AddPushButton("btn_print", 320, 660, 80, 24, fnt, "Print");
print.Action = new PdfNamedAction("Print");

page.AddSignatureField("sig", 320, 600, 220, 24);
```

Annotations

All §12.5.6 markup, geometric, file and media subtypes ship end-to-end. Each one round-trips through the reader and is paintable by the WPF viewer.

| Type | Description |
|--|--|
| <code>JetsonPDF.Annotations.PdfAnnotation</code> | Abstract base; common <code>/Rect</code> , <code>/Contents</code> , <code>/F</code> flags. |
| <code>JetsonPDF.Annotations.PdfLinkAnnotation</code> | URI or <code>GoTo</code> link with <code>/QuadPoints</code> regions. |
| <code>JetsonPDF.Annotations.PdfTextMarkupAnnotation</code> | Highlight, Underline, StrikeOut, Squiggly. |
| <code>JetsonPDF.Annotations.PdfFreeTextAnnotation</code> | Floating <code>/DA</code> -styled text box on the page. |
| <code>JetsonPDF.Annotations.PdfStampAnnotation</code> | Bitmap or vector stamp with rotation. |
| <code>JetsonPDF.Annotations.PdfInkAnnotation</code> | Free-hand strokes as Bezier polylines. |
| <code>JetsonPDF.Annotations.PdfFileAttachmentAnnotation</code> | Embedded file with <code>/F</code> + <code>/UF</code> Filespec. |
| <code>JetsonPDF.Annotations.PdfRedactionAnnotation</code> | Redact mark plus burn-in support. |
| <code>JetsonPDF.Annotations.PdfPolygonAnnotation</code> | Closed polygonal region with <code>/Measure</code> option. |
| <code>JetsonPDF.Annotations.PdfLineEnding</code> | <code>OpenArrow</code> / <code>ClosedArrow</code> / <code>Diamond</code> / ... endings. |

Example

```
var helv = new PdfFont(FontFamily.Helvetica, 11);

page.AddLink(36, 600, 380, 16, "https://www.iso.org/standard/75839.html");

page.AddAnnotation(new PdfStampAnnotation(290, 600, 120, 40, "Approved")
{
    Contents = "Approved on " + DateTime.Now.ToString("yyyy-MM-dd"),
});

page.AddAnnotation(new PdfLineAnnotation(60, 480, 240, 480)
{
    StrokeColor = PdfColor.Rgb(0.1, 0.3, 0.7),
    BorderWidth = 2,
    StartEnding = PdfLineEnding.OpenArrow,
    EndEnding = PdfLineEnding.ClosedArrow,
});
```

Encryption & signatures

Standard Security Handler with AES-128 and AES-256, owner/user passwords and the full §7.6 permission flag set. Invisible PKCS#7 detached signatures, RFC 3161 DocTimeStamps and a writable Document Security Store enable PAdES B-LTA.

| Type | Description |
|---------------------------------------|--|
| JetsonPDF.PdfEncryptionOptions | Password + permission + cipher choice. |
| JetsonPDF.PdfSigner | PKCS#7 detached sig via incremental update. |
| JetsonPDF.PdfTimestamper | RFC 3161 DocTimeStamp via TSA delegate. |
| JetsonPDF.PdfDss | Document Security Store with /VRI cert/CRL/OCSP. |
| JetsonPDF.Forms.PdfSignatureSeedValue | /SV constraints: filter, digest, MDP, reasons. |
| JetsonPDF.Forms.PdfFieldLock | All / Include / Exclude post-sign change rules. |
| JetsonPDF.PdfDocMdpPermission | DocMDP catalog /Perms - 1, 2 or 3. |
| JetsonPDF.PdfFieldMdpRestriction | FieldMDP /Reference for cosigned fields. |

Example

```
doc.Encryption = new PdfEncryptionOptions
{
    Algorithm      = PdfEncryptionAlgorithm.Aes256,
    UserPassword  = "user",
    OwnerPassword = "owner",
    Permissions   = PdfPermissions.Print | PdfPermissions.CopyContent,
};
doc.Save("encrypted.pdf");

var signed = PdfSigner.Sign(File.ReadAllBytes("encrypted.pdf"),
    new PdfSignatureOptions(signingCert)
    {
        Reason      = "Approved for release",
        FieldName   = "ApprovalSig",
    });
File.WriteAllBytes("signed.pdf", signed);
```

Conformance: PDF/A & PDF/UA

Conformance is opt-in via PdfDocument.Conformance plus the ThrowOnConformanceError save flag. Writer emits the correct XMP, /OutputIntents and catalog plumbing; the validator enforces the rules before save.

| Type | Description |
|-----------------------------------|--|
| JetsonPDF.PdfConformance | A1b/A2b/A2a/A2u/A3b/A3a/A3u, UA1, UA2. |
| JetsonPDF.PdfConformanceValidator | Twelve-plus rules with severity and code. |
| JetsonPDF.PdfConformanceIssue | Single validation finding (code, message, severity). |
| JetsonPDF.PdfOutputIntent | Required ICC output-intent for PDF/A. |
| doc.ThrowOnConformanceError | Run the validator on Save and throw on Error. |
| doc.MarkAsTagged = true | Emit /MarkInfo and tag the structure tree. |
| doc.Language = "en-US" | Set /Lang on the catalog (required for PDF/UA). |

Example

```
doc.Conformance = PdfConformance.PdfA2b | PdfConformance.PdfUA1;
doc.MarkAsTagged = true;
doc.Language = "en-US";
doc.OutputIntent = new PdfOutputIntent
{
    Subtype = "GTS_PDFA1",
    OutputConditionIdentifier = "sRGB IEC61966-2.1",
    OutputCondition = "sRGB",
    RegistryName = "http://www.color.org",
};

doc.ThrowOnConformanceError = true;
doc.Save("conformant.pdf");
```

Structure tree & accessibility

Tagged PDF is built bottom-up: BeginTag pushes a structure element on the stack; the using-block pops it. Structure elements carry Lang/Title/Alt/ActualText, attribute bundles and an OBJR back-reference to annotations.

| Type | Description |
|---|--|
| <code>page.BeginTag("H1")</code> | Push an element on the marked-content stack. |
| <code>JetsonPDF.PdfStructureElement</code> | Tree node with Title, ID, Alt, ActualText, Lang. |
| <code>JetsonPDF.PdfStructureAttributes</code> | Attribute owner + bag (Layout, Table, List, ...). |
| <code>doc.RoleMap["Custom"]="Note"</code> | Map private element types onto standard ones. |
| <code>doc.ClassMap["FigCenter"]=...</code> | Shared attribute classes referenced from elements. |
| <code>element.AnnotationRefs.Add(...)</code> | /OBJR child connecting structure to annotations. |

Example

```
using (var sect = page.BeginTag("Sect"))
{
    sect.Title = "Introduction";

    using (page.BeginTag("H1"))
        page.DrawText("Introduction", h1Fnt, x: 36, y: 720);

    using (page.BeginTag("P"))
        page.DrawTextBlock("Body text ...", bodyFnt,
            x: 36, y: 700, width: 540, lineHeight: 14);
}

doc.RoleMap["NoteBox"] = "Note";
```

Measurements & viewports

PdfMeasure (§12.9 rectilinear, §12.10 geospatial) tells viewers how content-stream coordinates map to real-world distances and coordinates - the toolbar shows '1 in = 10 ft' for engineering drawings or lat/lon read-outs for maps.

| Type | Description |
|---------------------------------|---|
| JetsonPDF.PdfMeasure | Abstract base for /RL and /GEO measurements. |
| JetsonPDF.PdfRectilinearMeasure | Engineering / floor-plan scale: ScaleRatio, X/Y/Distance/Area chains. |
| JetsonPDF.PdfGeospatialMeasure | Geographic mapping with GCS/PCS coordinate systems. |
| JetsonPDF.PdfNumberFormat | One step in a unit-conversion chain (ft -> in, deg -> min). |
| JetsonPDF.PdfCoordinateSystem | EPSG code or WKT definition for geospatial measures. |
| JetsonPDF.PdfViewport | Sub-rectangle of a page that pins a measure to its area. |
| page.Viewports.Add(...) | Attach a viewport to the page (§12.9.4 /VP entry). |

Example

```
var measure = new PdfRectilinearMeasure { ScaleRatio = "1 in = 10 ft" };
measure.XAxis.Add(new PdfNumberFormat
{
    Unit = "ft", ConversionFactor = 120, Precision = 100,
});
measure.Distance.Add(measure.XAxis[0]);
measure.Area.Add(new PdfNumberFormat { Unit = "sq ft", ConversionFactor = 14400 });

page.Viewports.Add(new PdfViewport
{
    BBox    = (36, 36, 576, 756),
    Name    = "Floor 1",
    Measure = measure,
});
```

Incremental updates, linearization & associated files

Append-only edits preserve every byte of the original; linearization rewrites a PDF for fast-web-view; associated files attach machine-readable payloads (e.g. Factur-X invoice XML) without breaking PDF/A.

| Type | Description |
|---|--|
| <code>JetsonPDF.PdfIncrementalUpdater</code> | Append-only updates over an existing PDF (metadata, widgets). |
| <code>PdfIncrementalUpdater.Open(path)</code> | Load a file and start an incremental session. |
| <code>updater.Title / Author / Subject</code> | Mutate /Info dict entries; <code>.Save()</code> returns full bytes. |
| <code>JetsonPDF.PdfLinearizer</code> | Annex F fast-web-view layout with hint stream and two xrefs. |
| <code>PdfLinearizer.Linearize(byte[])</code> | One-shot rewrite; returns the linearized bytes. |
| <code>JetsonPDF.PdfAssociatedFile</code> | AF entry: ties an embedded file to a relationship. |
| <code>JetsonPDF.PdfAfRelationship</code> | Source / Data / Alternative / Supplement / EncryptedPayload / FormData / Schema. |
| <code>JetsonPDF.PdfEmbeddedFile</code> | MIME-typed payload backing an <code>/EmbeddedFile</code> or <code>/Filespec</code> . |

Example

```
// Re-title an existing PDF without rewriting it.
var updater = PdfIncrementalUpdater.Open("input.pdf");
updater.Title    = "Quarterly Report (final)";
updater.Producer = "JetsonPDF";
File.WriteAllBytes("output.pdf", updater.Save());

// Linearize for fast-web-view.
byte[] fast = PdfLinearizer.Linearize(File.ReadAllBytes("output.pdf"));

// Attach Factur-X XML (PDF/A-3 compliant).
var xml = new PdfEmbeddedFile("factur-x.xml",
    File.ReadAllBytes("factur-x.xml"))
{
    MimeType = "application/xml",
};
doc.AssociatedFiles.Add(new PdfAssociatedFile(xml,
    PdfAfRelationship.Alternative));
```