JetsonPDF.Reader

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.

Reader entry point

PdfReader.Load returns an immutable ReadPdfDocument tree of DTOs that mirrors the writer surface. Every page item, annotation, form widget and metadata block has a Read* counterpart. Encrypted files use the password overload.

Type	Description
`JetsonPDF.Reading.PdfReader.Load(path)`	Open a PDF from disk; throws on parse error.
`PdfReader.Load(stream | byte[])`	In-memory variants - no filesystem access required.
`PdfReader.Load(path, password)`	Decrypts using the standard security handler.
`JetsonPDF.Reading.ReadPdfDocument`	Catalog DTO: Pages, Outlines, Conformance, Language, ...
`doc.Pages`	Read-only IList<ReadPdfPage> in catalog order.
`doc.NamedDestinations`	Map<string, PdfDestination> for cross-page navigation.
`doc.Conformance`	PdfConformance flags as advertised by the input file.

Example

```
using var fs = File.OpenRead("invoice.pdf");
ReadPdfDocument doc = PdfReader.Load(fs);

Console.WriteLine($"Title:       {doc.Title}");
Console.WriteLine($"Author:      {doc.Author}");
Console.WriteLine($"Pages:       {doc.Pages.Count}");
Console.WriteLine($"Conformance: {doc.Conformance}");

// Encrypted file:
ReadPdfDocument secured = PdfReader.Load("secret.pdf", "user-pwd");
```

Pages & content items

Each ReadPdfPage exposes its content as a flat IList<PageItem>. Pattern-match on the concrete subtype to walk text, vector paths, raster images, inline images and shading fills - the interpreter has already resolved colour spaces and the CTM.

Type	Description
`JetsonPDF.Reading.ReadPdfPage`	One page DTO: MediaBox, CropBox, Items, Annotations, Widgets.
`JetsonPDF.Reading.PageItem`	Abstract base of every drawn item.
`JetsonPDF.Reading.PageTextItem`	Decoded text run with font, size, Unicode and quads.
`JetsonPDF.Reading.PagePathItem`	Vector path with stroke/fill colours and pattern.
`JetsonPDF.Reading.PageImageItem`	Resolved image XObject + draw matrix.
`JetsonPDF.Reading.PageInlineImageItem`	Inline (BI/ID/EI) image content.
`JetsonPDF.Reading.PageShadingItem`	sh-painted shading rectangle.

Example

```
ReadPdfDocument doc = PdfReader.Load("sample.pdf");

foreach (ReadPdfPage page in doc.Pages)
{
    foreach (PageItem item in page.Items)
    {
        switch (item)
        {
            case PageTextItem  t: Console.WriteLine($"text:  {t.Text}"); break;
            case PageImageItem i: Console.WriteLine($"image: {i.PixelWidth}x{i.PixelHeight}"); break;
            case PagePathItem  p: Console.WriteLine($"path:  {p.PathData}"); break;
            case PageShadingItem s: Console.WriteLine("shading"); break;
        }
    }
}
```

Annotations

PageAnnotation hangs off ReadPdfPage.Annotations. The hierarchy mirrors §12.5.6: TextMarkup family, geometric shapes, free text, stamps, ink, file attachments, carets, popups, sounds and redactions. Every subtype round-trips from the writer.

Type	Description
`JetsonPDF.Reading.PageAnnotation`	Abstract base; rect, contents, flags.
`JetsonPDF.Reading.TextMarkupAnnotation`	Highlight / Underline / StrikeOut / Squiggly with TextMarkupKind.
`JetsonPDF.Reading.FreeTextAnnotation`	Floating /DA-styled text.
`JetsonPDF.Reading.StampAnnotation`	Stamp with rotation and contents.
`JetsonPDF.Reading.GeometricAnnotation`	Base for Line/Square/Circle/Polygon/PolyLine/Ink.
`JetsonPDF.Reading.LinkAnnotation`	URI or destination link with quads.
`JetsonPDF.Reading.FileAttachmentAnnotation`	Embedded file with /F + /UF Filespec.
`JetsonPDF.Reading.CaretAnnotation`	Caret marker (insertion point).
`JetsonPDF.Reading.PopupAnnotation`	Popup window paired with a parent annotation.
`JetsonPDF.Reading.SoundAnnotation`	Inline sound clip.
`JetsonPDF.Reading.RedactAnnotation`	Redact mark + burn-in metadata.

Example

```
ReadPdfPage page = PdfReader.Load("commented.pdf").Pages[0];

foreach (PageAnnotation a in page.Annotations)
{
    switch (a)
    {
        case TextMarkupAnnotation m:
            Console.WriteLine($"{m.Kind}: '{m.MarkedText}'");
            break;
        case FreeTextAnnotation ft:
            Console.WriteLine($"FreeText: {ft.Contents}");
            break;
        case LinkAnnotation link:
            Console.WriteLine($"Link -> {link.Url ?? link.InternalDestination?.ToString()}");
            break;
    }
}
```

Widgets & form actions

WidgetAnnotation surfaces every AcroForm field type with its computed appearance stream, value, options and action chain. PdfWidgetAction is the base of a tagged union the host pattern-matches when the user clicks the widget.

Type	Description
`JetsonPDF.Reading.WidgetAnnotation`	Form field DTO: FieldType, FieldFlags, Value, Options, Action.
`JetsonPDF.PdfWidgetAction`	Abstract action base parsed from /A.
`JetsonPDF.PdfUriAction`	External URL on activation.
`JetsonPDF.PdfNamedAction`	Standard viewer command (Print, NextPage, ...).
`JetsonPDF.PdfResetFormAction`	Reset listed fields (or every field if empty).
`JetsonPDF.PdfSubmitFormAction`	POST form data to a URL.
`JetsonPDF.Reading.ReadAdditionalActions`	/AA dictionary: Keystroke, Format, Validate, Calculate, mouse triggers.
`JetsonPDF.Reading.BarcodeMetadata`	Symbology + module bits when the widget is a barcode.

Example

```
foreach (var widget in page.Widgets)
{
    Console.WriteLine($"{widget.FieldName}: {widget.FieldType} = {widget.Value}");

    switch (widget.Action)
    {
        case PdfUriAction uri:    Console.WriteLine($"  -> {uri.Uri}"); break;
        case PdfNamedAction nm:   Console.WriteLine($"  named: {nm.Name}"); break;
        case PdfResetFormAction:  Console.WriteLine("  resets form"); break;
        case PdfSubmitFormAction submit: Console.WriteLine($"  submit -> {submit.Url}"); break;
    }
}
```

Document navigation

Outline tree (bookmarks), named destinations, and page labels. ReadOutlineItem is a recursive DTO; named destinations indirect through the catalog so cross-document references resolve uniformly.

Type	Description
`doc.Outlines`	IReadOnlyList<ReadOutlineItem> rooted at the catalog /Outlines.
`JetsonPDF.Reading.ReadOutlineItem`	Bookmark node: Title, Destination, Children, styling flags.
`JetsonPDF.PdfDestination`	Page + zoom hint (FitEntire, Xyz, FitH, FitR, ...).
`doc.NamedDestinations`	Map<string, PdfDestination> from catalog /Names tree.
`page.PageLabel`	Label for this page (roman numerals, A-prefix, ...).
`JetsonPDF.PdfPageLabelRange`	One range from the document /PageLabels number tree.

Example

```
static void Walk(ReadOutlineItem item, int depth)
{
    Console.WriteLine($"{new string(' ', depth * 2)}{item.Title} -> {item.Destination}");
    foreach (var child in item.Children) Walk(child, depth + 1);
}

foreach (var root in doc.Outlines) Walk(root, 0);

if (doc.NamedDestinations.TryGetValue("intro", out var dest))
    Console.WriteLine($"Jump 'intro' lives on page {dest.PageIndex + 1}");
```

Tagged PDF (structure tree)

Reader.ReadStructureTree mirrors the writer's bottom-up tag stack. Walk the tree via Children; resolve OBJR back-references to annotations through the optional ObjrResolver delegate. Attribute bundles surface as ReadStructureAttributes.

Type	Description
`doc.StructureTree`	Root ReadStructureTree (null for untagged PDFs).
`JetsonPDF.Reading.ReadStructureTree`	Document role/class maps + Children.
`JetsonPDF.Reading.ReadStructureElement`	Tree node: Type, Lang, Title, Alt, ActualText, ID, Attributes, Children.
`JetsonPDF.Reading.ReadStructureAttributes`	One attribute bundle: Owner + named values.
`element.AnnotationRefs`	/OBJR pointers - resolved if ObjrResolver is supplied.

Example

```
static void Dump(ReadStructureElement e, int depth)
{
    string lang = e.Lang is null ? "" : $" lang='{e.Lang}'";
    string actual = e.ActualText is null ? "" : $" actual='{e.ActualText}'";
    Console.WriteLine($"{new string(' ', depth * 2)}{e.Type}{lang}{actual}");
    foreach (var c in e.Children) Dump(c, depth + 1);
}

if (doc.StructureTree is { } tree)
    foreach (var root in tree.Children) Dump(root, 0);
```

Security & signatures

ReadPdfDocument exposes the encryption posture and every signature in /AcroForm. ReadDocumentSecurityStore surfaces DSS / VRI for long-term-validation trust, and ReadSignatureSeedValue / ReadFieldLock describe pre-sign constraints.

Type	Description
`doc.IsEncrypted`	True when the input was encrypted (decrypted in memory).
`doc.EncryptionAlgorithm`	PdfEncryptionAlgorithm in effect on the input.
`doc.Signatures`	All signature widgets parsed from the form.
`JetsonPDF.Reading.ReadSignature`	ByteRange + cert chain + reason + DocMDP/FieldMDP.
`JetsonPDF.Reading.ReadDocumentSecurityStore`	Top-level DSS: Certs, Crls, Ocsps + Vri map.
`JetsonPDF.Reading.ReadVri`	Per-signature validation-related info entry.
`JetsonPDF.Reading.ReadSignatureSeedValue`	/SV constraints attached to a signature widget.
`JetsonPDF.Reading.ReadFieldLock`	/Lock action: All / Include / Exclude.
`JetsonPDF.Reading.ReadFieldMdpRestriction`	FieldMDP transform: which fields are locked at sign time.

Example

```
Console.WriteLine($"Encrypted? {doc.IsEncrypted}  Algo: {doc.EncryptionAlgorithm}");

foreach (var sig in doc.Signatures)
{
    Console.WriteLine($"Signature '{sig.FieldName}'");
    Console.WriteLine($"  Reason:    {sig.Reason}");
    Console.WriteLine($"  ByteRange: {string.Join(",", sig.ByteRange)}");
    Console.WriteLine($"  Certs:     {sig.CertificateChain.Count}");
}

if (doc.SecurityStore is { } dss)
    Console.WriteLine($"DSS: {dss.Certificates.Count} certs, " +
                      $"{dss.Crls.Count} CRLs, {dss.Ocsps.Count} OCSPs");
```

Layers, viewports & measures

Optional content groups, viewport rectangles and measurement dictionaries all round-trip into reader DTOs. Use them to render a layer toggle UI, look up a viewport's coordinate system, or convert pixel distances back to real units.

Type	Description
`doc.OptionalContentGroups`	All OCGs declared by the input.
`JetsonPDF.Reading.ReadOptionalContentGroup`	OCG DTO: Name, VisibleByDefault, Intent.
`page.Viewports`	ReadViewport entries from the page /VP array.
`JetsonPDF.Reading.ReadViewport`	BBox + Name + optional ReadMeasure.
`JetsonPDF.Reading.ReadMeasure`	Abstract base; subclasses for RL and GEO.
`JetsonPDF.Reading.ReadRectilinearMeasure`	ScaleRatio + X/Y/Distance/Area number-format chains.
`JetsonPDF.Reading.ReadGeospatialMeasure`	Geographic coordinate system + projection.
`JetsonPDF.Reading.ReadNumberFormat`	Unit, ConversionFactor, Precision (chain element).
`JetsonPDF.Reading.ReadCoordinateSystem`	EPSG number or WKT definition.

Example

```
foreach (var ocg in doc.OptionalContentGroups)
    Console.WriteLine($"Layer: {ocg.Name}  visible={ocg.VisibleByDefault}");

foreach (var vp in doc.Pages[0].Viewports)
{
    Console.WriteLine($"Viewport '{vp.Name}': {vp.BBox}");
    if (vp.Measure is ReadRectilinearMeasure rl)
        Console.WriteLine($"  scale = {rl.ScaleRatio}");
}
```

Conformance, metadata & associated files

Conformance and language come from XMP plus catalog hints; output intents from /OutputIntents. Associated files are returned at every level - catalog, page, annotation - so a Factur-X consumer can find the XML payload regardless of where the producer stashed it.

Type	Description
`doc.Conformance`	PdfConformance flags detected from XMP.
`doc.Language`	Catalog /Lang (BCP 47).
`doc.OutputIntents`	IList<ReadOutputIntent> from /OutputIntents.
`JetsonPDF.Reading.ReadOutputIntent`	Subtype, ICC profile bytes, condition / registry.
`doc.AssociatedFiles`	Catalog-level /AF entries.
`page.AssociatedFiles`	Page-level /AF entries.
`JetsonPDF.Reading.ReadAssociatedFile`	Filename + MIME + relationship + bytes.
`doc.PieceInfo`	Editor-private metadata round-trip (/PieceInfo).

Example

```
Console.WriteLine($"Conformance: {doc.Conformance}");
Console.WriteLine($"Language:    {doc.Language ?? "(none)"}");

foreach (var oi in doc.OutputIntents)
    Console.WriteLine($"OutputIntent: {oi.OutputConditionIdentifier}");

foreach (var af in doc.AssociatedFiles)
    Console.WriteLine($"AF: {af.FileName} ({af.MimeType}, {af.Relationship})");
```

Worked example: a dump pipeline

Putting it all together: a single function that loads any PDF, walks every page and every annotation, and prints a one-line summary. Drop into a console project as your first reader exercise.

Type	Description
`PdfReader.Load(...)`	Entry point - file, stream or bytes.
`doc.Pages`	Iterate.
`page.Items`	Pattern-match on PageItem subtypes.
`page.Annotations`	Pattern-match on PageAnnotation subtypes.
`page.Widgets`	AcroForm widgets present on this page.
`doc.Outlines`	Top-level outline tree for navigation.
`doc.SecurityStore`	Optional DSS for LTV trust.

Example

```
static void Dump(string path)
{
    var doc = PdfReader.Load(path);
    Console.WriteLine($"== {path} ==");
    Console.WriteLine($"Title:  {doc.Title}");
    Console.WriteLine($"Pages:  {doc.Pages.Count}");

    for (int i = 0; i < doc.Pages.Count; i++)
    {
        var page = doc.Pages[i];
        int textCount = page.Items.OfType<PageTextItem>().Count();
        int imgCount  = page.Items.OfType<PageImageItem>().Count();
        Console.WriteLine($"  page {i + 1}: " +
            $"{textCount} text, {imgCount} images, " +
            $"{page.Annotations.Count} annotations, " +
            $"{page.Widgets.Count} widgets");
    }
}
```