

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 2026-06-13 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. Reader entry point 3
- 2. Pages & content items 4
- 3. Annotations 5
- 4. Widgets & form actions 6
- 5. Document navigation 7
- 6. Tagged PDF (structure tree) 8
- 7. Security & signatures 9
- 8. Layers, viewports & measures 10
- 9. Conformance, metadata & associated files 11
- 10. Worked example: a dump pipeline 12

Reader entry point

`Reader.Load` returns an immutable `ReadDocument` 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
<code>JetsonPDF.Reading.Reader.Load(path)</code>	Open a PDF from disk; throws on parse error.
<code>Reader.Load(stream byte[])</code>	In-memory variants - no filesystem access required.
<code>Reader.Load(path, password)</code>	Decrypts using the standard security handler.
<code>JetsonPDF.Reading.ReadDocument</code>	Catalog DTO: Pages, Outlines, Conformance, Language, ...
<code>doc.Pages</code>	Read-only <code>IList<ReadPage></code> in catalog order.
<code>doc.NamedDestinations</code>	<code>Map<string, Destination></code> for cross-page navigation.
<code>doc.Conformance</code>	Conformance flags as advertised by the input file.

Example

```
using var fs = File.OpenRead("invoice.pdf");
ReadDocument doc = Reader.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:
ReadDocument secured = Reader.Load("secret.pdf", "user-pwd");
```

Pages & content items

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

Example

```
ReadDocument doc = Reader.Load("sample.pdf");

foreach (ReadPage 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 ReadPage.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

```
ReadPage page = Reader.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. WidgetAction 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.WidgetAction	Abstract action base parsed from /A.
JetsonPDF.UriAction	External URL on activation.
JetsonPDF.NamedAction	Standard viewer command (Print, NextPage, ...).
JetsonPDF.ResetFormAction	Reset listed fields (or every field if empty).
JetsonPDF.SubmitFormAction	POST form data to a URL.
JetsonPDF.Reading.ReadAdditionalActions	/AA dictionary: Keystroke, Format, Validate, Calculate, mouse triggers.
JetsonPDF.Reading.ReadBarcodeMetadata	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 UriAction uri:    Console.WriteLine($"  -> {uri.Uri}"); break;
        case NamedAction nm:   Console.WriteLine($"  named: {nm.Name}"); break;
        case ResetFormAction:  Console.WriteLine("  resets form"); break;
        case SubmitFormAction submit: Console.WriteLine($"  submit -> {submit.Uri}"); 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
<code>doc.Outlines</code>	<code>IReadOnlyList<ReadOutlineItem></code> rooted at the catalog /Outlines.
<code>JetsonPDF.Reading.ReadOutlineItem</code>	Bookmark node: Title, Destination, Children, styling flags.
<code>JetsonPDF.Destination</code>	Page + zoom hint (<code>FitEntire</code> , <code>XYZ</code> , <code>FitH</code> , <code>FitR</code> , ...).
<code>doc.NamedDestinations</code>	<code>Map<string, Destination></code> from catalog /Names tree.
<code>page.PageLabel</code>	Label for this page (roman numerals, A-prefix, ...).
<code>JetsonPDF.PageLabelRange</code>	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
<code>doc.StructureTree</code>	Root <code>ReadStructureTree</code> (null for untagged PDFs).
<code>JetsonPDF.Reading.ReadStructureTree</code>	Document role/class maps + <code>Children</code> .
<code>JetsonPDF.Reading.ReadStructureElement</code>	Tree node: <code>Type</code> , <code>Lang</code> , <code>Title</code> , <code>Alt</code> , <code>ActualText</code> , <code>ID</code> , <code>Attributes</code> , <code>Children</code> .
<code>JetsonPDF.Reading.ReadStructureAttributes</code>	One attribute bundle: <code>Owner</code> + named values.
<code>element.AnnotationRefs</code>	/OBJR pointers - resolved if <code>ObjrResolver</code> 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

ReadDocument 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
<code>doc.IsEncrypted</code>	True when the input was encrypted (decrypted in memory).
<code>doc.EncryptionAlgorithm</code>	EncryptionAlgorithm in effect on the input.
<code>doc.Signatures</code>	All signature widgets parsed from the form.
<code>JetsonPDF.Reading.ReadSignature</code>	ByteRange + cert chain + reason + DocMDP/FieldMDP.
<code>JetsonPDF.Reading.ReadDocumentSecurityStore</code>	Top-level DSS: Certs, Crls, Ocsps + Vri map.
<code>JetsonPDF.Reading.ReadVri</code>	Per-signature validation-related info entry.
<code>JetsonPDF.Reading.ReadSignatureSeedValue</code>	/SV constraints attached to a signature widget.
<code>JetsonPDF.Reading.ReadFieldLock</code>	/Lock action: All / Include / Exclude.
<code>JetsonPDF.Reading.ReadFieldMdpRestriction</code>	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
<code>doc.OptionalContentGroups</code>	All OCGs declared by the input.
<code>JetsonPDF.Reading.ReadOptionalContentGroup</code>	OCG DTO: Name, VisibleByDefault, Intent.
<code>page.Viewports</code>	ReadViewport entries from the page /VP array.
<code>JetsonPDF.Reading.ReadViewport</code>	BBox + Name + optional ReadMeasure.
<code>JetsonPDF.Reading.ReadMeasure</code>	Abstract base; subclasses for RL and GEO.
<code>JetsonPDF.Reading.ReadRectilinearMeasure</code>	ScaleRatio + X/Y/Distance/Area number-format chains.
<code>JetsonPDF.Reading.ReadGeospatialMeasure</code>	Geographic coordinate system + projection.
<code>JetsonPDF.Reading.ReadNumberFormat</code>	Unit, ConversionFactor, Precision (chain element).
<code>JetsonPDF.Reading.ReadCoordinateSystem</code>	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 Factor-X consumer can find the XML payload regardless of where the producer stashed it.

Type	Description
<code>doc.Conformance</code>	Conformance flags detected from XMP.
<code>doc.Language</code>	Catalog /Lang (BCP 47).
<code>doc.OutputIntents</code>	<code>IList<ReadOutputIntent></code> from /OutputIntents.
<code>JetsonPDF.Reading.ReadOutputIntent</code>	Subtype, ICC profile bytes, condition / registry.
<code>doc.AssociatedFiles</code>	Catalog-level /AF entries.
<code>page.AssociatedFiles</code>	Page-level /AF entries.
<code>JetsonPDF.Reading.ReadAssociatedFile</code>	Filename + MIME + relationship + bytes.
<code>doc.PieceInfo</code>	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
<code>Reader.Load(...)</code>	Entry point - file, stream or bytes.
<code>doc.Pages</code>	Iterate.
<code>page.Items</code>	Pattern-match on <code>PageItem</code> subtypes.
<code>page.Annotations</code>	Pattern-match on <code>PageAnnotation</code> subtypes.
<code>page.Widgets</code>	AcroForm widgets present on this page.
<code>doc.Outlines</code>	Top-level outline tree for navigation.
<code>doc.SecurityStore</code>	Optional DSS for LTV trust.

Example

```
static void Dump(string path)
{
    var doc = Reader.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");
    }
}
```