# JetsonPDF.Fluent

## API Reference

*Built from scratch against ISO 32000-2 (PDF 2.0)*

This document was authored by JetsonPDF.Fluent itself. The cover band, page header, repeating chapter tables and per-page footer are produced by the same fluent surface the chapters describe. Use the Bookmarks pane to jump between sections.

```
Generated  2026-06-13 00:10 UTC

Producer   JetsonPDF.Fluent

Pipeline   FluentDocument.Create -> GeneratePdf
```

# Contents

One-line entries per feature area. Click an entry in the Bookmarks pane for direct navigation.

# Document & pages

FluentDocument.Create takes a build callback that adds one or more Page blocks. Each page has five layered slots - Background, Header, Content, Footer, Foreground - drawn in that order. Content auto-paginates on overflow.

| Type | Description |
|---|---|
| `JetsonPDF.Fluent.FluentDocument` | Fluent entry point; .Create + .GeneratePdf() / GeneratePdf(path\|stream) (no-arg returns byte[]). |
| `JetsonPDF.Fluent.IDocumentContainer` | Top-level builder; one .Page() call per page block. |
| `JetsonPDF.Fluent.IPageDescriptor` | Page builder: Size, Margin, DefaultTextStyle, slots. |
| `p.Size(PageSize) / Size(w,h)` | Set page dimensions (defaults to Letter). |
| `p.Margin(all) / Margin(h,v)` | Inset around the inner page. |
| `p.Header() / Footer() / Content()` | The three primary slots. |
| `p.Background() / Foreground()` | Behind / in front of the content for every paginated page. |
| `doc.WithMetadata(title, author)` | Set Title/Author on the resulting Document. |

## Example

```
var doc = FluentDocument.Create(d =>
{
    d.Page(p =>
    {
        p.Size(PageSize.Letter);
        p.Margin(50);
        p.DefaultTextStyle(s => s.FontSize = 11);
        p.Header().Text("Quarterly Report",
            s => s.FontSize = 18);
        p.Content().Text("Body content goes here.");
        p.Footer().AlignCenter().Text("Confidential");
    });
});
doc.WithMetadata("Q1 Report", "Finance team")
   .GeneratePdf("report.pdf");
// …or serialize in-memory: byte[] bytes = doc.GeneratePdf();
```

# Layout containers

Containers compose vertically (Column), horizontally (Row), in z-order (Stack), in fixed-column flow (Grid) or in line-wrapping flow (Inlined). Row distinguishes ConstantItem (fixed pt) from RelativeItem (proportional).

| Type | Description |
|---|---|
| `c.Column(col => ...)` | Vertical flow with Spacing + Item(). |
| `c.Row(row => ...)` | Horizontal flow with ConstantItem / RelativeItem. |
| `c.Stack(stack => ...)` | Z-order overlay - layers draw on top of one another. |
| `c.Grid(grid => ...)` | Fixed-column grid (rows wrap when full). |
| `c.Inlined(inl => ...)` | Horizontal line-wrapping flow (mixed-width items). |
| `col.Item() / stack.Item() / grid.Item()` | Open a slot for one child. |
| `row.ConstantItem(width)` | Fixed-width slot in points. |
| `row.RelativeItem(weight=1)` | Proportional slot of remaining width. |
| `col.Spacing(gap) / row.Spacing(gap)` | Gap between siblings (also on Inlined/Grid). |

## Example

```
p.Content().Column(col =>
{
    col.Spacing(8);
    col.Item().Row(row =>
    {
        row.RelativeItem(2).Text("INVOICE",
            s => s.FontSize = 24);
        row.ConstantItem(120).AlignRight()
           .Text("#" + invoiceNumber);
    });
    col.Item().LineHorizontal(1, Colors.Grey300);
    col.Item().Text("Body...");
});
```

# Text & rich text

Three .Text overloads: a bare string, a string + style configurer, and a TextDescriptor builder for inline runs and embedded page numbers. Spans inherit from the descriptor's base TextStyle but can override Bold/Italic/FontSize.

| Type | Description |
|---|---|
| `c.Text(string)` | Quick text with the default style. |
| `c.Text(string, configure)` | Text with a one-off style override. |
| `c.Text(td => td.Span(...).Bold())` | Rich text with mixed inline runs. |
| `JetsonPDF.Fluent.TextStyle` | FontFamily / FontStyle / FontSize / FontColor / LineHeight. |
| `JetsonPDF.Fluent.TextDescriptor` | Builder for rich text; Span / CurrentPageNumber / TotalPages. |
| `JetsonPDF.Fluent.TextSpanDescriptor` | Per-run style: .Bold / .Italic / .FontSize / .FontColor. |
| `c.PageNumber() / c.TotalPages()` | Stand-alone page-number leaves (no surrounding text). |

## Example

```
p.Footer().AlignCenter().Text(t =>
{
    t.Span("Page ").FontColor(Colors.Grey700);
    t.CurrentPageNumber().Bold();
    t.Span(" of ").FontColor(Colors.Grey700);
    t.TotalPages().Bold();
});
p.Content().Text("Bold heading", s =>
{
    s.FontSize  = 18;
    s.FontStyle = FontStyle.Bold;
    s.FontColor = Colors.IndigoDark;
});
```

# Tables — headers, bodies & footers

Tables are built from a ColumnsDefinition plus optional Header/Footer plus body Cells. Header repeats on every page; Footer renders only on the final page; the body auto-paginates at row boundaries. columnSpan and rowSpan are first-class.

| Type | Description |
|---|---|
| `c.Table(t => ...)` | Build a table from columns + cells. |
| `t.ColumnsDefinition(cd => ...)` | Declare relative or constant columns. |
| `cd.RelativeColumn(weight=1)` | Proportional column. |
| `cd.ConstantColumn(width)` | Fixed-width column in points. |
| `t.Header(h => h.Cell()...)` | Repeating header (drawn on every page). |
| `t.Footer(f => f.Cell()...)` | Footer (drawn on the final page only). |
| `t.Cell(columnSpan, rowSpan)` | Body cell; spans flow auto-occupancy. |

## Example

```
p.Content().Table(t =>
{
    t.ColumnsDefinition(cd =>
    {
        cd.RelativeColumn(3);
        cd.RelativeColumn(1);
        cd.RelativeColumn(1);
    });
    t.Header(h =>
    {
        h.Cell().Text("Item");
        h.Cell().AlignRight().Text("Qty");
        h.Cell().AlignRight().Text("Price");
    });
    foreach (var (item, qty, price) in lineItems)
    {
        t.Cell().Text(item);
        t.Cell().AlignRight().Text(qty.ToString());
        t.Cell().AlignRight().Text(price.ToString("C"));
    }
    t.Footer(f =>
    {
        f.Cell(columnSpan: 2).AlignRight().Text("Total");
        f.Cell().AlignRight().Text(total.ToString("C"));
    });
});
```

# Spacing & sizing

Padding, Width, Height and the Min/Max bounds are decorators - they wrap a slot and reshape it before the child measures itself. Width / Height pin a dimension exactly; Min/Max set bounds. PaddingHorizontal / PaddingVertical for symmetry.

| Type | Description |
|---|---|
| `c.Padding(all)` | Equal pad on all four sides. |
| `c.Padding(horizontal, vertical)` | Two-axis pad. |
| `c.PaddingLeft / Right / Top / Bottom` | One side at a time. |
| `c.PaddingHorizontal / Vertical` | Both sides on one axis. |
| `c.Width(pt) / c.Height(pt)` | Pin the dimension exactly. |
| `c.MinWidth / MaxWidth` | Bound horizontal size. |
| `c.MinHeight / MaxHeight` | Bound vertical size. |

## Example

```
p.Content().Column(col =>
{
    col.Spacing(12);
    col.Item().Padding(20).Background(Colors.Grey50)
              .Text("Padded callout block.");
    col.Item().Height(80).Border(1, Colors.Grey400)
              .AlignCenter().AlignMiddle()
              .Text("Fixed-height box");
    col.Item().PaddingHorizontal(40)
              .Text("Long-form paragraph with " +
                    "horizontal breathing room only...");
});
```

# Alignment & positioning

Align decorators claim the available slot and align the child within it. Translate shifts a slot by an offset; Rotate spins it (CTM); AspectRatio constrains width/height to a ratio. Applied in chain order.

| Type | Description |
|---|---|
| `c.AlignLeft / AlignCenter / AlignRight` | Horizontal alignment within the slot. |
| `c.AlignTop / AlignMiddle / AlignBottom` | Vertical alignment within the slot. |
| `c.Translate(dx, dy)` | Translate the child by an offset. |
| `c.Rotate(degrees)` | Rotate the child via CTM (about its origin). |
| `c.AspectRatio(ratio, mode)` | Constrain w/h to a ratio (Fit / Fill / Stretch). |

## Example

```
p.Foreground().AlignTop().AlignRight().Padding(10)
              .Text("DRAFT", s =>
              {
                  s.FontSize  = 32;
                  s.FontColor = Colors.Red;
                  s.FontStyle = FontStyle.Bold;
              });
p.Content().AlignCenter().Width(360).Rotate(-3)
           .Background(Colors.AmberDark).Padding(20)
           .Text("Stamped & spinning");
```

# Visual styling

Border, Background and Shadow are simple decorators on top of any slot. Pair with the Colors palette - a Material-flavoured set of named hues plus Colors.FromHex for arbitrary brand colours.

| Type | Description |
|---|---|
| `c.Border(width, color = null)` | Stroke a rectangle around the slot. |
| `c.Background(color)` | Solid fill behind the slot's child. |
| `c.Shadow(dx, dy, color)` | Drop-shadow rectangle behind the slot. |
| `JetsonPDF.Fluent.Colors` | Material palette (Red/Pink/.../Brown + Greys + Light/Dark). |
| `Colors.FromHex("#1A2B3C")` | Parse an arbitrary hex colour. |

## Example

```
p.Content().Background(Colors.IndigoLight)
           .Border(1, Colors.IndigoDark)
           .Shadow(2, 2, Colors.Grey400)
           .Padding(16)
           .Text("Branded callout", s =>
           {
               s.FontStyle = FontStyle.Bold;
               s.FontColor = Colors.Grey900;
           });
```

# Forms — AcroForm widgets

AsTextField / AsCheckBox / AsComboBox / AsListBox / AsPushButton attach an AcroForm widget to the slot's measured rectangle. The widget consumes the slot as a leaf; Acrobat draws its native chrome at view-time.

| Type | Description |
|---|---|
| `c.AsTextField(name, configure?)` | Single- or multi-line text field. |
| `c.AsCheckBox(name, configure?)` | Toggle widget (named On state). |
| `c.AsComboBox(name, opts, configure?)` | Drop-down (editable or fixed). |
| `c.AsListBox(name, opts, configure?)` | Multi-select list. |
| `c.AsPushButton(name, caption, configure?)` | Action button (configure to set Action). |

## Example

```
p.Content().Column(col =>
{
    col.Spacing(10);
    col.Item().Row(row =>
    {
        row.ConstantItem(120).AlignMiddle().Text("Name:");
        row.RelativeItem().Height(22)
           .AsTextField("name", t => t.MaxLength = 64);
    });
    col.Item().Row(row =>
    {
        row.ConstantItem(20).Height(20)
           .AsCheckBox("subscribe");
        row.ConstantItem(8);
        row.RelativeItem().AlignMiddle().Text("Subscribe");
    });
    col.Item().Width(120).Height(28)
       .AsPushButton("submit", "Submit");
});
```

# Navigation — links, sections & outlines

Link wraps a slot in an external URL hotspot; Section registers a named destination at the slot's top-left; SectionLink jumps to one. Pair with WithOutline to give the bookmarks pane a tree.

| Type | Description |
|---|---|
| `c.Link(uri)` | External-URL hotspot over the slot. |
| `c.Section(anchor)` | Register a named destination at this point. |
| `c.SectionLink(anchor)` | Internal jump to a named destination. |
| `doc.WithOutline(b => b.Item(...))` | Build the bookmarks tree against named destinations. |
| `JetsonPDF.Fluent.OutlineBuilder` | Tree builder; Item returns an OutlineEntryBuilder for nesting. |
| `doc.WithPageLabels(...)` | Roman numerals / letter prefixes for the page-number bar. |

## Example

```
FluentDocument.Create(d =>
{
    d.Page(p =>
    {
        p.Content().Column(col =>
        {
            col.Item().Section("intro").Text("Introduction",
                s => s.FontSize = 22);
            col.Item().Text("Body...");
            col.Item().Section("toc").Text("Table of Contents",
                s => s.FontSize = 22);
            col.Item().SectionLink("intro")
                      .Text("Jump to Introduction",
                            s => s.FontColor = Colors.BlueDark);
            col.Item().Link("https://www.iso.org")
                      .Text("ISO website");
        });
    });
})
.WithOutline(o =>
{
    o.Item("Introduction", "intro");
    o.Item("Table of Contents", "toc");
});
```

# Pagination, layers & advanced

Pagination decorators control how slots react to page breaks. Layer assigns the child to an optional-content group. Component yields an IDynamicComponent that composes per page (knows the current page number). Canvas is the escape hatch.

| Type | Description |
|---|---|
| `c.ShowOnce()` | Render once - skip on subsequent pages of the same block. |
| `c.SkipOnce()` | Skip on the first page; render thereafter. |
| `c.ShowEntire()` | Treat as atomic - never split across pages. |
| `c.EnsureSpace(minHeight)` | Page-break before if less than minHeight remains. |
| `c.Layer(handle)` | Put content in an optional-content group (layer). |
| `c.Component(IDynamicComponent)` | Per-page composer (sees CurrentPageNumber/TotalPages). |
| `c.Canvas((surface,w,h) => ...)` | Drop to a low-level IDrawingSurface for custom painting. |
| `doc.WithLayer(name, visible?, intent?)` | Register an OCG and get a LayerHandle. |

## Example

```
var watermark = doc.WithLayer("Watermark", visibleByDefault: true);
FluentDocument.Create(d =>
{
    d.Page(p =>
    {
        p.Foreground().Layer(watermark).AlignCenter().AlignMiddle()
                      .Text("CONFIDENTIAL", s =>
                      {
                          s.FontSize  = 60;
                          s.FontColor = Colors.RedLight;
                      });
        p.Content().Column(col =>
        {
            col.Item().ShowOnce().Text("First-page intro");
            col.Item().ShowEntire().Padding(8)
                      .Background(Colors.Grey100)
                      .Text("Don't split me across pages.");
            col.Item().Canvas((surface, w, h) =>
            {
                surface.DrawRectangle(0, 0, w, h,
                    fill: Color.Rgb(0.95, 0.95, 1.0));
            });
        });
    });
});
```