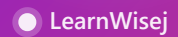


FlowLayoutPanel, TableLayoutPanel, FlexLayoutPanel, and Extended Layout Properties

Lesson Guide · Lab / Exam Guide



Lesson Guide

Choosing flow, table or flex containers by content behaviour, the extended child properties FillWeight, FlowBreak, AlignX, AlignY, Row, Column, spans and styles, when layout engines honour margins, and fluent markup extensions for programmatic layouts. What this means for a Wisej.NET developer — and how to read this module.



Production focus

Choosing flow, table or flex containers by content behaviour, the extended child properties FillWeight, FlowBreak, AlignX, AlignY, Row, Column, spans and styles, when layout engines honour margins, and fluent markup extensions for programmatic layouts.

1. The production problem

Module 4 gave the Adaptive Operations Console a shell of docked regions, but docking only decides where a region sits, not how the content inside it behaves when the browser is 1400 pixels wide on Monday and 700 on a tablet on Tuesday. A row of metric cards should wrap, a ticket editor should keep its labels and editors aligned however long the German translation is, and the list and details regions should share the workspace in a proportion you decide. Each is a different content behaviour, and Wisej.NET ships a container for each: `FlowLayoutPanel`, `TableLayoutPanel` and `FlexLayoutPanel`. Picking the wrong one is how teams end up writing resize code again.

2. Key concepts

Concept	Why it matters
FlowLayoutPanel	Arranges children in automatic rows or columns along <code>FlowDirection</code> , wraps when <code>WrapContents</code> is true and ignores each child's Location, Dock and Anchor, so chips and cards reflow on their own.
TableLayoutPanel	Places children in cells addressed by Row and Column, with spans for cells that stretch, so labels and editors stay aligned while each child can still dock or anchor inside its cell.
FlexLayoutPanel	Lays children out in one row or one column, always fills its client area and shares the remaining space by weight, which suits major regions that split the workspace proportionally.
FillWeight	An extended child property read by the flow and flex engines: a weight above zero stretches the child into the remaining space in proportion to the other weighted children; zero keeps its own size.
FlowBreak	An extended FlowLayoutPanel child property that forces the next child onto a new row or column, so a filter bar ends a line deliberately instead of relying on the width.
Row and column styles	<code>RowStyles</code> and <code>ColumnStyles</code> declare whether a row or column auto-sizes, takes a fixed pixel size or shares the rest by percentage; this is where a form's proportions live.
Margin and Padding	Padding is the container's inner space around all children; Margin is the child's distance to its neighbours, and only engines that honour margins will use it.

3. Lesson narrative

Choose the container by content behaviour

The decision rule is short enough to remember and strict enough to review against. Use `FlowLayoutPanel` for content that can wrap: filter chips, metric cards, toolbar overflow areas, responsive command sets. Use `TableLayoutPanel` for structured forms: field labels, aligned editors, summary rows and any grid of controls that must survive localisation. Use `FlexLayoutPanel` for major regions that must fill the available space proportionally, such as a navigation rail beside a content region and a side details panel, or stacked mobile sections that share the height by weight. Docking and anchoring remain right for stable shell regions and simple forms; the three containers are for the content inside those regions.

Look at the behaviour, not the visual. The console's metric-card area looks like a grid, but its behaviour is "as many cards per row as fit, then wrap", which is flow. The ticket editor looks like a list of fields, but its behaviour is "every editor's left edge on the same line", which is table. The workspace looks like two panels, but its behaviour is "the list takes two thirds, the details one third, and neither collapses", which is flex. The lab builds all three so you can resize the browser and watch which container survives which scenario; that written comparison is the deliverable that matters most.

The extended child properties

The containers work through extended properties: values that belong to the child but are stored and read by the parent, and that appear in the child's property grid only while it sits inside that container.

`FlowLayoutPanel` reads `Margin`, `FillWeight`, `MinimumSize`, `MaximumSize` and `FlowBreak`, set in code through the container: `SetFillWeight(search, 1)` lets the search box use whatever width the row has left, and `SetFlowBreak(apply, true)` ends the row so the cards below start on a fresh line.

`FlowDirection` chooses left-to-right, right-to-left, top-down or bottom-up; with `WrapContents` false the children are clipped or scrolled instead of wrapping, which is what a single-line toolbar with `AutoScroll` wants.

`TableLayoutPanel` reads `Row`, `Column`, `RowSpan` and `ColumnSpan` from each child and sizes the grid from `RowStyles` and `ColumnStyles`: auto-size, an absolute pixel value or a percentage. The ticket editor uses an absolute 140-pixel label column and a 100-percent editor column, so the editors grow with the panel while the labels stay put, and the Notes editor spans both columns in a row that takes the remaining height. `GrowStyle` decides what happens when a control is added after the cells are full: add rows, add columns or refuse to grow. Because a child can still dock inside its cell, an editor docked Fill in a percent column is the cleanest aligned form Wisej.NET produces.

`FlexLayoutPanel` reads `Margin`, `MinimumSize`, `MaximumSize`, `FillWeight`, `AlignX` and `AlignY`. Its `LayoutStyle` is horizontal (one row) or vertical (one column); the container always fills its client area and distributes the remaining space among weighted children. When a child does not fill a dimension, `AlignX` and `AlignY` override the parent's alignment for that child alone, so a details region with `AlignY` top hugs the top edge of a tall workspace instead of centring. The rule the lab checks: every fill-weighted child gets a `MinimumSize`, otherwise a narrow browser shrinks a region to zero and the user sees an empty strip where the details editor was. `MaximumSize` caps the same region so a 4K monitor does not stretch a form into whitespace.

```
// DashboardWorkspace : FlexLayoutPanel – the list/details split
this.Dock = DockStyle.Fill;
this.LayoutStyle = FlexLayoutStyle.Horizontal;
this.Padding = new Padding(8);

var list = new Panel { MinimumSize = new Size(300, 200) };
var details = new Panel { MinimumSize = new Size(280, 200),
                          MaximumSize = new Size(520, 0) };
this.Controls.Add(list);
this.Controls.Add(details);

this.SetFillWeight(list, 2); // two thirds of the spare width
this.SetFillWeight(details, 1); // one third, never below 280 px
this.SetAlignY(details, VerticalAlignment.Top);
```

When layout engines honour margins

`Margin` is the property most often set and least often respected, because whether it does anything depends on the engine laying the child out. The flow and flex engines enforce margins: every child of a `FlowLayoutPanel` or `FlexLayoutPanel` keeps the distance its `Margin` declares. The default layout engine does not use margins for docking; a Panel docked Left with a `Margin` of 8 sits flush against the edge

anyway, which is why Module 4 used Padding on the docked regions for gaps. `TableLayoutPanel` accounts for margins inside the cell, so a child's Margin becomes its distance to the cell border while Padding on the table adds the outer frame.

Treat the two properties as belonging to different owners: Padding is the container's decision about the inner space around all of its children, Margin the child's decision about its distance to adjacent controls. A consistent layout system for the console defines standard values once, for example dense, normal and spacious modes of 4, 8 and 16 pixels, and applies them through those two properties everywhere. When a gap does not appear, ask which engine owns the child before changing any number.

Fluent markup extensions

The `Wisej.Web.Markup` namespace adds fluent extension methods that read like a layout description. `FlowLayoutPanelExtensions` can set FillWeight and FlowBreak, `FlexLayoutPanelExtensions` can set FillWeight, AlignX and AlignY, and `TableLayoutPanelExtensions` can set column counts, grow style and layout settings. Each method returns the container, so a programmatic shell becomes one chain: `panel.FillWeight(nav, 0).FillWeight(content, 1).AlignY(nav, VerticalAlignment.Top)`. The lab asks you to build one region this way; the natural candidate is a region you already create in code, such as a dashboard whose cards depend on the signed-in user's role.

Use the fluent style where it makes a programmatic layout readable and leave designer-built screens to the designer. Mixing designer serialisation, imperative `SetFillWeight` calls and fluent chains in one class forces the next developer to read three notations for one panel, so keep one notation per container. Either way the checklist applies: containers instead of resize handlers, docking order checked, Padding on docked regions, flow for wrapping, table for forms, flex for proportional regions, and a MinimumSize on every fill-weighted control.



Common pitfalls

Setting Location, Dock or Anchor on a FlowLayoutPanel child and wondering why it is ignored. Fill-weighted regions without a MinimumSize that collapse on a phone. Expecting Margin to create gaps between docked Panels under the default layout engine. Reaching for FlexLayoutPanel for a form with aligned labels, or for TableLayoutPanel for a row of cards that should wrap.



Visual-system habit

Use the Wisej.NET mechanism that owns the problem: the theme for the identity of standard controls, AppearanceKey for semantic variants, CssClass for scoped app-specific CSS, CssStyle only for one-off dynamic values, layout containers instead of resize code, responsive properties for designer-set profile values, and ResponsiveProfileChanged for behaviour that is not simply visual.



Learning objectives

By the end of this module you'll be able to:

- ✓ Choose FlowLayoutPanel, TableLayoutPanel, or FlexLayoutPanel based on content behavior and screen purpose.

- ✓ Use FillWeight, FlowBreak, AlignX, AlignY, Row, Column, RowSpan, ColumnSpan, RowStyles, ColumnStyles, GrowStyle, MinimumSize, and MaximumSize.
- ✓ Understand when layout engines use margins and when they do not.
- ✓ Use markup extension methods for fluent layout definitions where appropriate.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

Build three versions of the Adaptive Operations Console content area: a `FlowLayoutPanel` filter and metric-card area that wraps and uses `FillWeight` and `FlowBreak`, a `TableLayoutPanel` ticket editor with aligned labels and editors driven by `RowStyles` and `ColumnStyles`, and a `FlexLayoutPanel` dashboard shell where the list and details regions share space by `FillWeight` with `AlignX`, `AlignY`, `MinimumSize` and `MaximumSize` set. Build one of the three in code with the `Wisej.Web.Markup` fluent extensions and compare how each survives a browser resize.

Starting point

Open the `AdaptiveOperationsConsole` solution as you left it after Module 4: the main Page is a nested-container shell with a toolbar, a navigation rail, a fill workspace, a details panel and a status region, the navigation and details regions are `UserControls`, and the workspace is still a plain `Panel` whose metric cards, ticket editor and list are positioned by hand. Keep the three new regions in a `Layout` folder so the capstone can pick the winner of each comparison.

Suggested approach

- ✓ Create a `FilterBar` that derives from `FlowLayoutPanel` with `FlowDirection` left-to-right and `WrapContents` on; add the search box, status combo and Apply button, give the search box `FillWeight` 1 and a `MinimumSize`, set `FlowBreak` on Apply, then drop the metric cards below it and watch them wrap.
- ✓ Build a `TicketEditor` `UserControl` around a two-column `TableLayoutPanel`: an absolute 140-pixel label column, a 100-percent editor column, one auto-size row per field, a Notes row that spans both columns with the remaining height, and every editor docked Fill in its cell.
- ✓ Replace the workspace `Panel` with a `DashboardWorkspace` that derives from `FlexLayoutPanel` in horizontal layout; add the list and details regions with `FillWeight` 2 and 1, a `MinimumSize` on both, a `MaximumSize` on details and `AlignY` top on details.
- ✓ Build one of the three regions entirely in code with `using Wisej.Web.Markup;` and the fluent `FillWeight`, `FlowBreak`, `AlignX` and `AlignY` extension methods chained on the container, and keep that class free of designer serialisation.
- ✓ Run the console, resize the browser from 1400 to 1024 to 700 pixels for each region, take a screenshot at each width and write down, per region, which container held up and which extended property did the work.

Required deliverables

- ✓ `FlowLayoutPanel` filter and metric-card area using `WrapContents`, `FillWeight` and `FlowBreak`
- ✓ `TableLayoutPanel` ticket editor with `RowStyles`, `ColumnStyles` and spans for aligned fields
- ✓ `FlexLayoutPanel` dashboard shell with `FillWeight`, `AlignX`, `AlignY`, `MinimumSize` and `MaximumSize`

- ✓ One region built in code with the Wisej.Web.Markup fluent extensions
- ✓ Resize comparison screenshots and a note on which container fits each region

Student review questions

- ✓ You gave the details region FillWeight 1 but no MinimumSize, and at 700 pixels it vanished. Explain what the flex engine did with the space and why the fix is a size constraint rather than a different weight.
- ✓ The Apply button wraps to the second row at 1024 pixels even though there is room for it. Which extended property did that, and how would the FilterBar behave differently if you removed it and made WrapContents false?
- ✓ A Margin of 8 on the cards in the FlowLayoutPanel produced gaps, but the same Margin on the docked navigation rail did nothing. Which engine honours margins in each case, and which property creates the gap for the rail?



Visual-system habit

Use the Wisej.NET mechanism that owns the problem: the theme for the identity of standard controls, AppearanceKey for semantic variants, CssClass for scoped app-specific CSS, CssStyle only for one-off dynamic values, layout containers instead of resize code, responsive properties for designer-set profile values, and ResponsiveProfileChanged for behaviour that is not simply visual.