Introduction

Nova.Avalonia.UI is a control library built for Avalonia. It focuses on controls that are themeable, accessible, and ready to drop into desktop, web, and mobile experiences.

Available controls

- **Avatar** and **AvatarGroup**: Identity visuals with initials, images, status badges, and grouping support.
- **Badge**: Notifications, status indicators, and counters with customizable placement and overflow handling.
- **BarcodeGenerator**: Generate QR codes, 1D barcodes, and 2D matrix codes with customizable styling.
- **RatingControl**: Interactive star ratings with customizable shapes and precision levels.
- **Shimmer**: Skeleton loading effect for async data scenarios.

How to use these docs

- Start with Getting Started to install the package and register the styles.
- Browse the individual control pages under **Controls** for API details and usage patterns.
- Refer to the API reference for full class members when you need to extend or customize behaviors.

Getting Started

Follow these steps to install Nova.Avalonia.UI, register its styles, and place your first control in a view.

Prerequisites

- Avalonia 11 or later
- .NET 9 (the library currently targets `net9.0`)

Install the NuGet package

From your application project, install the library:

```
dotnet add package Nova.Avalonia.UI
```

Register the control styles

Add the Nova styles to your `Application.Styles` so the controls pick up their templates. Keep your base theme (for example, `FluentTheme`) before the style include.

```
<Application xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="MyApp.App">
    <Application.Styles>
        <FluentTheme />
        <StyleInclude Source="avares://Nova.Avalonia.UI/Themes/Controls.axaml" />
    </Application.Styles>
</Application>
```

Use the controls in XAML

Declare the namespace for the controls and drop them into your layout. This example shows a shimmer placeholder wrapping content and a simple avatar.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="clr-
namespace:Nova.Avalonia.UI.Controls;assembly=Nova.Avalonia.UI">

    <StackPanel Spacing="16">
        <nova:Shimmer IsLoading="True" LoadingText="Loading profile">
            <StackPanel Spacing="8">
                <TextBlock FontSize="18" Text="Profile" />
```

```
                <Border Height="120" CornerRadius="12" Background="#1F1F1F" />
            </StackPanel>
        </nova:Shimmer>

        <nova:Avatar DisplayName="Avery Patel" Status="Online" />
    </StackPanel>
</UserControl>
```

Next, explore the individual control pages to see customization options and platform-specific notes.

ArcPanel

The `ArcPanel` arranges child elements along an arc, which is a portion of a circle. This is useful for creating semi-circular menus, dial interfaces, or decorative layouts.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:ArcPanel Radius="100" StartAngle="-90" SweepAngle="180">
        <Border Width="40" Height="40" Background="Red" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="Green" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="Blue" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="Orange" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="Purple" CornerRadius="20"/>
    </nova:ArcPanel>
</UserControl>
```

Positioning

- **StartAngle**: The angle (in degrees) where the arc begins. 0° is at the 3 o'clock position, and angles increase clockwise.
- **SweepAngle**: The total arc length in degrees. Use 180 for a semi-circle, 90 for a quarter-circle.

Items are distributed evenly along the arc based on the number of children.

Properties

Property	Type	Default	Description
`Radius`	`double`	`100`	The radius of the arc from center to item centers.
`StartAngle`	`double`	`0`	The starting angle in degrees (0° = 3 o'clock).
`SweepAngle`	`double`	`180`	The arc span in degrees.
`RotateItems`	`bool`	`false`	Whether items rotate to follow the arc tangent.

Examples

Semi-Circle Menu (Top)

```
<nova:ArcPanel Radius="120" StartAngle="-90" SweepAngle="180"
               HorizontalAlignment="Center" VerticalAlignment="Center">
    <Button Content="1"/>
```

```
    <Button Content="2"/>
    <Button Content="3"/>
    <Button Content="4"/>
    <Button Content="5"/>
</nova:ArcPanel>
```

Quarter Arc

```
<nova:ArcPanel Radius="80" StartAngle="0" SweepAngle="90">
    <Ellipse Width="30" Height="30" Fill="Coral"/>
    <Ellipse Width="30" Height="30" Fill="Teal"/>
    <Ellipse Width="30" Height="30" Fill="Gold"/>
</nova:ArcPanel>
```

AutoLayout

`AutoLayout` is a specialized panel that replicates the behavior of Figma's Auto Layout feature. It provides an intuitive way to stack items, control spacing, and manage alignment without the complexity of a Grid.

Basic Usage

The API is designed to be familiar to designers and developers used to Figma.

```
<nova:AutoLayout Orientation="Horizontal" Spacing="10" Padding="20">
    <Button Content="Item 1"/>
    <Button Content="Item 2"/>
</nova:AutoLayout>
```

Properties

Property	Type	Default	Description
`Orientation`	`Orientation`	`Vertical`	Direction to stack items (`Horizontal` or `Vertical`).
`Spacing`	`double`	`0`	Uniform distance between items.
`Padding`	`Thickness`	`0`	Padding around the content.
`Justification`	`AutoLayoutJustify`	`Packed`	Controls distribution (`Packed` vs `SpaceBetween`).
`HorizontalContentAlignment`	`HorizontalAlignment`	`Left`	Aligns the content horizontally within the panel.
`VerticalContentAlignment`	`VerticalAlignment`	`Top`	Aligns the content vertically within the panel.
`IsReverseZIndex`	`bool`	`false`	If true, reverses the visual stacking order (last item on bottom).

Alignment & Distribution

Alignment Matrix

Use `HorizontalContentAlignment` and `VerticalContentAlignment` to align the entire cluster of items within the panel. This eliminates the need to set alignment on individual children (unless you want to override it).

```
<!-- Centers items in the middle of the panel -->
<nova:AutoLayout HorizontalContentAlignment="Center"
VerticalContentAlignment="Center">
   ...
</nova:AutoLayout>
```

Justification

- **Packed (Default)**: Items are bundled together separated by `Spacing`. They adhere to the ContentAlignment properties.
- **SpaceBetween**: Items are distributed evenly to fill the available space. The `Spacing` property is ignored (spacing is calculated dynamically).

```
<nova:AutoLayout Justification="SpaceBetween">
   <TextBlock Text="Left"/>
   <TextBlock Text="Right"/>
</nova:AutoLayout>
```

Absolute Positioning

You can exclude items from the auto-layout flow (e.g., for notification badges or overlays) using the attached property `AutoLayout.IsAbsolute="True"`.

```
<nova:AutoLayout>
    <Button Content="Message"/>

    <!-- Floats on top at top-right corner -->
    <Border nova:AutoLayout.IsAbsolute="True"
            HorizontalAlignment="Right" VerticalAlignment="Top">
        <TextBlock Text="1"/>
    </Border>
</nova:AutoLayout>
```

Figma Mapping

Figma Concept	**AutoLayout Property**
Direction (Arrow)	`Orientation`

Figma Concept	AutoLayout Property
Resizing: Hug Contents	`HorizontalAlignment="Left/Center/Right"` (on Container)
Resizing: Fill Container	`HorizontalAlignment="Stretch"` (on Container)
Spacing Mode: Packed	`Justification="Packed"`
Spacing Mode: Auto	`Justification="SpaceBetween"`
Absolute Position	`AutoLayout.IsAbsolute="True"`

Avatar

The `Avatar` control presents a person's identity using an image, initials, icon, or custom content. It includes automatic background generation, size presets, and optional presence status indicators.

Create an avatar

Declare an `Avatar` and set `DisplayName`. With the default `DisplayMode` of `Auto`, the control will render initials when no image or icon is provided.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="clr-namespace:Nova.Avalonia.UI.Controls;assembly=Nova.Avalonia.

    <nova:Avatar DisplayName="Alex Martin" />
</UserControl>
```

Use images, icons, or custom content

Choose a display mode explicitly when you need to control the visual output:

- `DisplayMode="Image"` uses the `ImageSource` bitmap.
- `DisplayMode="Icon"` shows the provided `Icon` content.
- `DisplayMode="Content"` renders any custom `Content`.

```
<StackPanel Spacing="12">
    <nova:Avatar DisplayName="Jamie Fox" ImageSource="avares://Assets/jamie.png" Display
    <nova:Avatar DisplayName="Operations" DisplayMode="Icon">
        <nova:Avatar.Icon>
            <PathIcon Data="M18,13 L6,13 6,11 18,11z" />
        </nova:Avatar.Icon>
    </nova:Avatar>
    <nova:Avatar DisplayName="Admin" DisplayMode="Content">
        <nova:Avatar.Content>
            <Ellipse Fill="#F59E0B" Width="18" Height="18" />
        </nova:Avatar.Content>
    </nova:Avatar>
</StackPanel>
```

> **NOTE**
>
> If `DisplayMode` is left as `Auto`, the control picks an image when available, otherwise initials, then icon, then content.

Size, shape, and color

`Avatar` supports preset sizes via `Size` (`ExtraSmall` through `ExtraLarge`) and a `Custom` option controlled by `CustomSize`. Use `Shape` to switch between `Circle`, `Square`, and `Rectangle` corners.

The control can auto-generate a background color from the display name when `AutoGenerateBackground` is `True`, or you can set `BackgroundColor` and `ForegroundColor` directly.

```
<UniformGrid Columns="3" Rows="1" Margin="0,12,0,0">
    <nova:Avatar DisplayName="Kim Lee" Size="Small" />
    <nova:Avatar DisplayName="Drew Parker" Size="Large" Shape="Square" BackgroundColor="
    <nova:Avatar DisplayName="Avery Patel" Size="Custom" CustomSize="80" Shape="Rectangl
</UniformGrid>
```

Show presence status

Attach a status indicator with the `Status` property. You can override the default color per status with `StatusColor`.

```
<StackPanel Orientation="Horizontal" Spacing="10">
    <nova:Avatar DisplayName="Taylor Reed" Status="Online" />
    <nova:Avatar DisplayName="Morgan" Status="Away" />
    <nova:Avatar DisplayName="Jordan" Status="Busy" StatusColor="#C026D3" />
</StackPanel>
```

Tooltips automatically display the `DisplayName` when `ShowTooltip` is `True`, which helps identify users when only initials or icons are visible.

Arrange multiple avatars with AvatarGroup

Use `AvatarGroup` to stack or wrap multiple `Avatar` controls with configurable overlap and overflow handling. Combine `Spacing` and `MaxVisibleAvatars` to control layout, and place any remaining avatars in an overflow badge.

```
<StackPanel Spacing="12">
    <nova:AvatarGroup MaxVisibleAvatars="3" Spacing="-8">
        <nova:Avatar DisplayName="Taylor Reed" Status="Online" />
        <nova:Avatar DisplayName="Morgan Lee" Status="Away" />
        <nova:Avatar DisplayName="Jamie Fox" Status="Busy" />
        <nova:Avatar DisplayName="Avery Patel" />
    </nova:AvatarGroup>

    <nova:AvatarGroup Orientation="Vertical" Spacing="4">
        <nova:Avatar DisplayName="Ops" DisplayMode="Icon">
            <nova:Avatar.Icon>
                <PathIcon Data="M18,13 L6,13 6,11 18,11z" />
```

```
            </nova:Avatar.Icon>
        </nova:Avatar>
        <nova:Avatar DisplayName="Engineering" Status="Online" />
    </nova:AvatarGroup>
</StackPanel>
```

`AvatarGroup` also exposes `BorderBrush` and `BorderThickness` to add a ring around the stack when you need a stronger visual boundary against busy backgrounds.

Badge

The `Badge` control displays notifications, status indicators, or short information (like counts) attached to another element or as a standalone indicator. It supports different placements, themes (colors), shapes, and overflow handling for large numbers.

Create a badge

Wrap any content (like a `Button` or `Icon`) with the `Badge` control. Set the `BadgeContent` to define what is displayed inside the badge.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:controls="using:Nova.Avalonia.UI.Controls">

    <controls:Badge BadgeContent="5">
        <Button Content="Notifications" />
    </controls:Badge>
</UserControl>
```

Placements

Control where the badge appears relative to its content using the `BadgePlacement` property. Supported values are:

- `TopLeft`, `Top`, `TopRight` (Default)
- `Left`, `Right`
- `BottomLeft`, `Bottom`, `BottomRight`

```
<StackPanel Spacing="20" Orientation="Horizontal">
    <!-- Default is TopRight -->
    <controls:Badge BadgeContent="1">
        <Border Width="40" Height="40" Background="LightGray" CornerRadius="4"/>
    </controls:Badge>

    <controls:Badge BadgeContent="2" BadgePlacement="BottomRight">
        <Border Width="40" Height="40" Background="LightGray" CornerRadius="4"/>
    </controls:Badge>

    <controls:Badge BadgeContent="3" BadgePlacement="TopLeft">
        <Border Width="40" Height="40" Background="LightGray" CornerRadius="4"/>
    </controls:Badge>
</StackPanel>
```

Themes and Colors

The control includes several built-in themes for common semantic colors. You can apply them using the `Theme` property with a `StaticResource`.

Solid Themes:

- `PrimaryBadge`, `SecondaryBadge`
- `SuccessBadge`, `WarningBadge`, `DangerBadge`, `InfoBadge`
- `LightBadge`, `DarkBadge`

Outline Themes:

- `PrimaryOutlineBadge`, `SuccessOutlineBadge`, etc.

```
<WrapPanel>
    <controls:Badge BadgeContent="Success" Theme="{StaticResource SuccessBadge}" Margin=
    <controls:Badge BadgeContent="Warning" Theme="{StaticResource WarningBadge}" Margin=
    <controls:Badge BadgeContent="Danger" Theme="{StaticResource DangerBadge}" Margin="5
    <controls:Badge BadgeContent="Outline" Theme="{StaticResource PrimaryOutlineBadge}"
</WrapPanel>
```

Shapes and Sizes

You can customize the shape and size using themes or properties.

- **Shapes**: `SquareBadge`, `PillBadge`
- **Sizes**: `SmallBadge`, `LargeBadge`

```
<StackPanel Spacing="10" Orientation="Horizontal">
    <controls:Badge BadgeContent="Pill" Theme="{StaticResource PillBadge}" />
    <controls:Badge BadgeContent="Square" Theme="{StaticResource SquareBadge}" />
    <controls:Badge BadgeContent="Small" Theme="{StaticResource SmallBadge}" />
</StackPanel>
```

Dot Badges

If you only need a simple indicator without text, use `Kind="Dot"`. You can combined this with size themes like `SmallDotBadge` or `LargeDotBadge`.

```
<controls:Badge Kind="Dot" Theme="{StaticResource SuccessBadge}">
    <Button Content="Status" />
</controls:Badge>
```

Overflow Handling (MaxCount)

When displaying numbers, you can limit the maximum value shown using `MaxCount`. If the `BadgeContent` exceeds this limit, it will display the limit followed by a `+` (e.g., "99+").

- The default `MaxCount` is 99. If the content is numeric and exceeds this value, it will be truncated with a `+` suffix logic.

```
<!-- Displays "99+" if content is > 99 -->
<controls:Badge BadgeContent="150" Theme="{StaticResource DangerBadge}">
    <Button Content="Inbox" />
</controls:Badge>

<!-- Custom limit Example (Conceptual) -->
<controls:Badge BadgeContent="10" MaxCount="9" Theme="{StaticResource WarningBadge}">
    <Button Content="Messages" />
</controls:Badge>
```

Standalone Usage

The `Badge` control can also be used without wrapping content, acting as a standalone tag or label.

```
<StackPanel Orientation="Horizontal" Spacing="5">
    <TextBlock Text="Status:" VerticalAlignment="Center"/>
    <controls:Badge BadgeContent="New" Theme="{StaticResource InfoBadge}" />
</StackPanel>
```

Properties reference

Property	Type	Description
`BadgeContent`	`object`	The content to display inside the badge (text, number, etc.).
`BadgePlacement`	`BadgePlacement`	The position of the badge relative to the content. Default is `TopRight`.
`Kind`	`BadgeKind`	The visual style of the badge content. Values: `Content` (default), `Dot`.
`MaxCount`	`int`	The maximum numeric value to display before showing a `+` suffix. Default is 99.
`IsBadgeVisible`	`bool`	Controls the visibility of the badge itself. Default is `True`.
`BadgeOffset`	`double`	Additional X/Y offset to fine-tune the badge position.

BarcodeGenerator

The `BarcodeGenerator` control generates and renders various barcode symbologies including QR codes, Data Matrix, Code 128, and more. It uses the ZXing library for encoding and supports customizable colors, error correction, and logo overlays.

Create a barcode

Declare a `BarcodeGenerator` and set the `Value` and `Symbology` properties.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:bc="clr-namespace:Nova.Avalonia.UI.BarcodeGenerator;assembly=Nova.Ava

    <bc:BarcodeGenerator Value="https://avaloniaui.net"
                         Symbology="QRCode"
                         Width="200" Height="200" />
</UserControl>
```

Supported symbologies

The control supports multiple barcode formats via the `Symbology` property:

Symbology	Type	Use Case
`QRCode`	2D	URLs, contact info, general data
`DataMatrix`	2D	Small items, electronics
`Aztec`	2D	Transport tickets, boarding passes
`PDF417`	2D	ID cards, shipping labels
`Code128`	1D	Shipping, logistics
`Code39`	1D	Automotive, defense
`Code93`	1D	Postal services
`EAN13`	1D	Retail products (Europe)
`EAN8`	1D	Small retail products
`UPCA`	1D	Retail products (North America)
`UPCE`	1D	Small packages

Symbology	Type	Use Case
`Codabar`	1D	Libraries, blood banks
`ITF`	1D	Logistics, shipping

```
<StackPanel Spacing="12">
    <bc:BarcodeGenerator Value="PRODUCT123" Symbology="Code128" Width="250" Height="80"
    <bc:BarcodeGenerator Value="5901234123457" Symbology="EAN13" Width="200" Height="70"
    <bc:BarcodeGenerator Value="DATA-MATRIX" Symbology="DataMatrix" Width="150" Height="
</StackPanel>
```

Custom colors

Customize the barcode appearance using `BarBrush` and `BackgroundBrush`.

```
<bc:BarcodeGenerator Value="Styled QR"
                     Symbology="QRCode"
                     BarBrush="#1565C0"
                     BackgroundBrush="#E3F2FD"
                     Width="200" Height="200" />
```

> **TIP**
>
> Use `DynamicResource` bindings for theme-aware colors that adapt to light and dark modes.

Display text

Show the encoded value below the barcode with `ShowText`. Customize the text appearance with related properties.

```
<bc:BarcodeGenerator Value="AVALONIA2025"
                     Symbology="Code128"
                     ShowText="True"
                     TextFontSize="14"
                     TextAlignment="Center"
                     TextMargin="0,8,0,0"
                     Width="300" Height="100" />
```

Error correction (QR and Aztec)

For QR codes and Aztec codes, set the `ErrorCorrectionLevel` to control damage recovery:

Level	Recovery	Description
`L`	~7%	Low - smallest code size
`M`	~15%	Medium (default)
`Q`	~25%	Quartile
`H`	~30%	High - best for logos

```
<bc:BarcodeGenerator Value="https://avaloniaui.net"
                     Symbology="QRCode"
                     ErrorCorrectionLevel="H"
                     Width="200" Height="200" />
```

QR code with logo

Overlay a logo in the center of QR or Aztec codes. Use `ErrorCorrectionLevel="H"` for best results.

```
<bc:BarcodeGenerator Value="https://avaloniaui.net"
                     Symbology="QRCode"
                     ErrorCorrectionLevel="H"
                     LogoSizePercent="0.25"
                     Width="200" Height="200">
    <bc:BarcodeGenerator.Logo>
        <Bitmap>avares://MyApp/Assets/logo.png</Bitmap>
    </bc:BarcodeGenerator.Logo>
</bc:BarcodeGenerator>
```

The `LogoSizePercent` property controls the logo size relative to the barcode (0.1 to 0.4).

Quiet zone

The `QuietZone` property sets the margin around the barcode in modules. The default is 2.

```
<bc:BarcodeGenerator Value="QR" Symbology="QRCode" QuietZone="4" />
```

Handle events

Subscribe to `BarcodeGenerated` and `BarcodeError` events for generation feedback.

```
barcode.BarcodeGenerated += (s, e) =>
{
    Console.WriteLine($"Generated {e.Matrix.Width}x{e.Matrix.Height} matrix");
};
```

```
barcode.BarcodeError += (s, e) =>
{
    Console.WriteLine($"Error: {e.Exception.Message}");
};
```

Properties

Property	Type	Default	Description
`Value`	`string`	`""`	Data to encode
`Symbology`	`BarcodeSymbology`	`QRCode`	Barcode format
`BarBrush`	`IBrush`	`Black`	Fill for barcode bars
`BackgroundBrush`	`IBrush`	`White`	Background fill
`QuietZone`	`int`	`2`	Margin in modules
`ShowText`	`bool`	`false`	Show encoded value as text
`TextFontSize`	`double`	`14`	Text font size
`TextForeground`	`IBrush`	`Black`	Text color
`TextMargin`	`Thickness`	`0,8,0,0`	Text margin
`TextAlignment`	`TextAlignment`	`Center`	Text alignment
`ErrorCorrectionLevel`	`QRErrorCorrectionLevel`	`M`	Error recovery level
`Logo`	`IImage`	`null`	Center logo image
`LogoSizePercent`	`double`	`0.25`	Logo size (0.1-0.4)
`ErrorMessage`	`string`	`null`	Last error message

Events

Event	Description
`BarcodeGenerated`	Raised when barcode is successfully generated
`BarcodeError`	Raised when generation fails

BubblePanel

The `BubblePanel` arranges child elements using a circle packing algorithm, creating a compact bubble-like arrangement. It's ideal for tag clouds, data visualization, or decorative layouts where items should appear densely packed.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:BubblePanel>
        <Ellipse Width="60" Height="60" Fill="#FF6B6B"/>
        <Ellipse Width="40" Height="40" Fill="#4ECDC4"/>
        <Ellipse Width="80" Height="80" Fill="#45B7D1"/>
        <Ellipse Width="50" Height="50" Fill="#FFA07A"/>
        <Ellipse Width="70" Height="70" Fill="#98D8C8"/>
    </nova:BubblePanel>
</UserControl>
```

How It Works

The panel uses a force-directed circle packing algorithm that:

1. Places items starting from the center
2. Pushes items outward to avoid overlap
3. Packs larger items first for optimal density
4. Respects each item's `DesiredSize` as the bubble diameter

Properties

Property	Type	Default	Description
`Padding`	`Thickness`	`0`	Space around the packed bubbles.
`Spacing`	`double`	`4`	Minimum spacing between bubbles.

Tips

> **TIP**
>
> For best results, use circular items (equal width/height) like `Ellipse` or `Border` with `CornerRadius` equal to half the size.

Example with Data Binding

```
<ItemsControl ItemsSource="{Binding Tags}">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <nova:BubblePanel/>
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>
    <ItemsControl.ItemTemplate>
        <DataTemplate>
            <Border Width="{Binding Size}" Height="{Binding Size}"
                    Background="{Binding Color}"
                    CornerRadius="{Binding Size, Converter={StaticResource HalfConverter
                <TextBlock Text="{Binding Name}"
                           HorizontalAlignment="Center"
                           VerticalAlignment="Center"/>
            </Border>
        </DataTemplate>
    </ItemsControl.ItemTemplate>
</ItemsControl>
```

CircularPanel

The `CircularPanel` arranges child elements evenly around a circle. This is perfect for radial menus, clock faces, or any circular arrangement.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:CircularPanel Radius="100">
        <Border Width="40" Height="40" Background="#FF6B6B" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="#4ECDC4" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="#45B7D1" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="#FFA07A" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="#98D8C8" CornerRadius="20"/>
        <Border Width="40" Height="40" Background="#F7DC6F" CornerRadius="20"/>
    </nova:CircularPanel>
</UserControl>
```

Positioning

Items are automatically distributed evenly around the circle. The `StartAngle` property controls where the first item is placed.

- **0°**: 3 o'clock position (right)
- **-90°**: 12 o'clock position (top)
- **90°**: 6 o'clock position (bottom)
- **180°**: 9 o'clock position (left)

Properties

Property	Type	Default	Description
`Radius`	`double`	`100`	The radius of the circle.
`StartAngle`	`double`	`0`	The angle where the first item is placed (degrees).
`AngleStep`	`double`	`0`	Override the angle between items. 0 = auto-calculate.
`SweepDirection`	`SweepDirection`	`Clockwise`	Direction of item placement (`Clockwise` or `CounterClockwise`).

Examples

Clock Face

```
<nova:CircularPanel Radius="100" StartAngle="-90">
    <TextBlock Text="12" FontWeight="Bold"/>
    <TextBlock Text="1"/>
    <TextBlock Text="2"/>
    <TextBlock Text="3" FontWeight="Bold"/>
    <TextBlock Text="4"/>
    <TextBlock Text="5"/>
    <TextBlock Text="6" FontWeight="Bold"/>
    <TextBlock Text="7"/>
    <TextBlock Text="8"/>
    <TextBlock Text="9" FontWeight="Bold"/>
    <TextBlock Text="10"/>
    <TextBlock Text="11"/>
</nova:CircularPanel>
```

Radial Menu

```
<nova:CircularPanel Radius="80" StartAngle="-90">
    <Button Content="🏠" ToolTip.Tip="Home"/>
    <Button Content="⚙️" ToolTip.Tip="Settings"/>
    <Button Content="📁" ToolTip.Tip="Files"/>
    <Button Content="❤️" ToolTip.Tip="Favorites"/>
</nova:CircularPanel>
```

HexPanel

The `HexPanel` arranges items in a hexagonal grid pattern (honeycomb). It supports both "Flat Top" and "Pointy Top" hex orientations and uses row/column coordinates for item placement.

Basic Usage

To position items, wrap them in the `HexPanel` and use the `HexPanel.Row` and `HexPanel.Column` attached properties on the item container.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:HexPanel ColumnCount="3" RowCount="3" Orientation="Horizontal">
        <!-- Item at Row 0, Column 0 -->
        <Ellipse Width="80" Height="80" Fill="Red"
                 nova:HexPanel.Row="0" nova:HexPanel.Column="0" />

        <!-- Item at Row 1, Column 1 (Center) -->
        <Ellipse Width="80" Height="80" Fill="Blue"
                 nova:HexPanel.Row="1" nova:HexPanel.Column="1" />
    </nova:HexPanel>
</UserControl>
```

> **IMPORTANT**
>
> When using `HexPanel` as the `ItemsPanel` of an `ItemsControl`, you must apply the attached properties to the item container (e.g., `ContentPresenter`) using a Style, as the `DataTemplate` content is wrapped inside the container.

```
<ItemsControl ItemsSource="{Binding Items}">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <nova:HexPanel ColumnCount="3" RowCount="3" />
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>
    <ItemsControl.Styles>
        <Style Selector="ContentPresenter" x:DataType="local:HexItem">
            <Setter Property="nova:HexPanel.Row" Value="{Binding Row}" />
            <Setter Property="nova:HexPanel.Column" Value="{Binding Column}" />
        </Style>
    </ItemsControl.Styles>
```

```
    <!-- ... ItemTemplate ... -->
</ItemsControl>
```

Orientation

The `Orientation` property controls the shape and stacking direction of the hexagons.

- **Horizontal**: Creates a grid of "Pointy Top" hexagons.
- **Vertical**: Creates a grid of "Flat Top" hexagons.

Properties

Property	Type	Default	Description
`Orientation`	`Orientation`	`Vertical`	The orientation of the hex grid (`Horizontal` or `Vertical`).
`RowCount`	`int`	`0`	The number of rows in the grid (affects size calculation).
`ColumnCount`	`int`	`0`	The number of columns in the grid.

Attached Properties

Property	Type	Description
`Row`	`int`	The zero-based row index for the item.
`Column`	`int`	The zero-based column index for the item.

LoopPanel

The `LoopPanel` creates an infinite/looping scrolling experience where children wrap seamlessly. It supports drag gestures, mouse wheel scrolling, momentum (inertia), and snap-to-item behavior.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:LoopPanel Height="120" Spacing="10">
        <Border Width="100" Height="100" Background="Red" CornerRadius="8"/>
        <Border Width="100" Height="100" Background="Green" CornerRadius="8"/>
        <Border Width="100" Height="100" Background="Blue" CornerRadius="8"/>
        <Border Width="100" Height="100" Background="Orange" CornerRadius="8"/>
    </nova:LoopPanel>
</UserControl>
```

Properties

Property	Type	Default	Description
`Orientation`	`Orientation`	`Horizontal`	Layout direction (Horizontal or Vertical).
`Offset`	`double`	`0.0`	Current scroll position. 1.0 = one full item.
`AnchorPosition`	`double`	`0.5`	Viewport anchor point (0.0=start, 0.5=center, 1.0=end).
`Spacing`	`double`	`0.0`	Gap between items in pixels.
`IsInertiaEnabled`	`bool`	`True`	Enable momentum scrolling after drag release.
`SnapToItems`	`bool`	`True`	Snap to nearest item when scrolling ends.
`ScrollFactor`	`double`	`1.0`	Multiplier for drag/wheel sensitivity.

Methods

Method	Description
`ScrollBy(double units)`	Scroll by specified pixel units.
`ScrollToIndex(int index, bool animate)`	Scroll to bring a specific child to the anchor point.

Events

Event	Description
`CurrentIndexChanged`	Raised when the pivotal (anchored) item index changes.

Vertical Orientation

```
<nova:LoopPanel Orientation="Vertical" Height="300" Width="150" Spacing="10">
    <Border Height="80" Background="Red"/>
    <Border Height="80" Background="Green"/>
    <Border Height="80" Background="Blue"/>
</nova:LoopPanel>
```

Anchor Positioning

Control where items align in the viewport:

```
<!-- Items start at left edge -->
<nova:LoopPanel AnchorPosition="0.0" />

<!-- Items center in viewport (default) -->
<nova:LoopPanel AnchorPosition="0.5" />

<!-- Items align to right edge -->
<nova:LoopPanel AnchorPosition="1.0" />
```

Programmatic Scrolling

```
// Scroll by 100 pixels
myLoopPanel.ScrollBy(100);

// Jump to item at index 3
myLoopPanel.ScrollToIndex(3, animate: false);

// Animate to item at index 5
myLoopPanel.ScrollToIndex(5, animate: true);
```

Handling Index Changes

```
myLoopPanel.CurrentIndexChanged += (sender, index) =>
{
    Debug.WriteLine($"Now showing item {index}");
};
```

Disabling Inertia

For precise control without momentum:

```
<nova:LoopPanel IsInertiaEnabled="False" SnapToItems="True" />
```

Dynamic Items

Items can be added or removed at runtime:

```
myLoopPanel.Children.Add(new Border { Width = 100, Height = 100, Background =
Brushes.Purple });
myLoopPanel.Children.RemoveAt(myLoopPanel.Children.Count - 1);
```

Notes

- Each child control can only appear once in the viewport at any time.
- For very wide viewports, ensure you have enough items to fill the space.
- `ClipToBounds` is enabled by default to prevent overflow.

OrbitPanel

The `OrbitPanel` places items in concentric rings (orbits) around a central point. You assign items to specific orbits using the `OrbitPanel.Orbit` attached property.

Basic Usage

Items with `Orbit="0"` are placed in the center. Items with `Orbit="1"` go to the first ring, and so on.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:OrbitPanel InnerRadius="50" OrbitSpacing="60" StartAngle="0">

        <!-- Center Item -->
        <Ellipse Width="50" Height="50" Fill="Red" nova:OrbitPanel.Orbit="0" />

        <!-- Orbit 1 Items -->
        <Ellipse Width="30" Height="30" Fill="Blue" nova:OrbitPanel.Orbit="1" />
        <Ellipse Width="30" Height="30" Fill="Blue" nova:OrbitPanel.Orbit="1" />

        <!-- Orbit 2 Items -->
        <Ellipse Width="40" Height="40" Fill="Green" nova:OrbitPanel.Orbit="2" />
        <Ellipse Width="40" Height="40" Fill="Green" nova:OrbitPanel.Orbit="2" />
        <Ellipse Width="40" Height="40" Fill="Green" nova:OrbitPanel.Orbit="2" />

    </nova:OrbitPanel>
</UserControl>
```

Layout Logic

- **Orbit 0**: Items are stacked at the center coordinates (useful if you only have one center item, or want them to overlap at the center).
- **Orbit 1+**: Items are distributed evenly along the circumference of the ring.

Properties

Property	Type	Default	Description
`InnerRadius`	`double`	`50`	The radius of the first orbit ring (Orbit 1).
`OrbitSpacing`	`double`	`60`	The distance between consecutive rings.
`StartAngle`	`double`	`0`	The starting angle for item distribution in degrees.

Attached Properties

Property	Type	Description
`Orbit`	`int`	The zero-based index of the orbit ring to place the item in. 0 is center.

OverlapPanel

The `OverlapPanel` stacks child elements with configurable horizontal and vertical offsets, creating an overlapping card or pile effect. It's ideal for card stacks, notification badges, or layered UI elements.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:OverlapPanel OffsetX="20" OffsetY="15">
        <Border Width="100" Height="70" Background="#E74C3C" CornerRadius="8"/>
        <Border Width="100" Height="70" Background="#3498DB" CornerRadius="8"/>
        <Border Width="100" Height="70" Background="#2ECC71" CornerRadius="8"/>
        <Border Width="100" Height="70" Background="#F39C12" CornerRadius="8"/>
    </nova:OverlapPanel>
</UserControl>
```

Z-Order

By default, later items appear on top of earlier items. Use `ReverseZIndex` to flip this order.

```
<!-- First item on top -->
<nova:OverlapPanel OffsetX="20" OffsetY="10" ReverseZIndex="True">
    <Border Background="Red" Width="80" Height="60"/>
    <Border Background="Green" Width="80" Height="60"/>
    <Border Background="Blue" Width="80" Height="60"/>
</nova:OverlapPanel>
```

Negative Offsets

Use negative offsets to stack items in the opposite direction:

```
<!-- Stack from bottom-right to top-left -->
<nova:OverlapPanel OffsetX="-15" OffsetY="-10">
    <Border Background="Purple" Width="80" Height="60"/>
    <Border Background="Teal" Width="80" Height="60"/>
    <Border Background="Orange" Width="80" Height="60"/>
</nova:OverlapPanel>
```

Properties

Property	Type	Default	Description
`OffsetX`	`double`	`10`	Horizontal offset between items (can be negative).

Property	Type	Default	Description
`OffsetY`	`double`	`10`	Vertical offset between items (can be negative).
`ReverseZIndex`	`bool`	`false`	When true, earlier items appear on top.

Use Cases

- **Card stacks**: Playing cards, photo albums
- **Notification stacks**: Multiple alerts piled together
- **Breadcrumb trails**: Overlapping path indicators
- **Avatar groups**: Use with circular items for grouped avatars

RadialPanel

The `RadialPanel` arranges its children in a circular fan or ring pattern. It is useful for creating menus, dials, or decorative layouts where items surround a central point.

Basic Usage

The `RadialPanel` calculates positions based on a central point and a specified radius.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:RadialPanel Radius="150" StartAngle="0" SweepAngle="360" RotateItems="True">
        <Button Content="Item 1" />
        <Button Content="Item 2" />
        <Button Content="Item 3" />
        <Button Content="Item 4" />
        <Button Content="Item 5" />
    </nova:RadialPanel>
</UserControl>
```

Layout Configuration

You can control the arc of the layout using `StartAngle` and `SweepAngle`.

- **Radius**: The distance from the center of the panel to the center of the items.
- **StartAngle**: The angle (in degrees) where the first item is placed. 0 is exactly to the right (3 o'clock).
- **SweepAngle**: The total angle covered by the layout. 360 creates a full circle; 180 creates a semi-circle.
- **RotateItems**: If true, rotates the items so their top points outward from the center.

```
<nova:RadialPanel Radius="100"
                  StartAngle="-90"
                  SweepAngle="180"
                  RotateItems="False">
    <!-- Items arranged in a semi-circle starting from the top -->
</nova:RadialPanel>
```

Properties

Property	Type	Default	Description
`Radius`	`double`	`100`	The radius of the circle layout.

Property	Type	Default	Description
`StartAngle`	`double`	`0`	The starting angle in degrees.
`SweepAngle`	`double`	`360`	The total angle range in degrees.
`ItemAngle`	`double`	`0`	An additional rotation angle applied to each item.
`RotateItems`	`bool`	`True`	Whether to rotate items to face outward from the center.

RatingControl

The `RatingControl` allows users to view and set ratings using interactive items such as stars, hearts, or custom shapes. It supports multiple precision levels, customizable appearance, and full keyboard and pointer interaction.

Create a rating control

Declare a `RatingControl` and set the `Value` property. By default, the control displays 5 star-shaped items.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="clr-namespace:Nova.Avalonia.UI.Controls;assembly=Nova.Avalonia.

    <nova:RatingControl Value="3" />
</UserControl>
```

Precision modes

Control how values are selected using the `Precision` property:

- `Full` - Only whole numbers (1, 2, 3, etc.)
- `Half` - Half-step increments (1.5, 2.5, etc.)
- `Exact` - Continuous values based on pointer position

```
<StackPanel Spacing="12">
    <nova:RatingControl Value="3" Precision="Full" />
    <nova:RatingControl Value="3.5" Precision="Half" />
    <nova:RatingControl Value="4.2" Precision="Exact" />
</StackPanel>
```

> **NOTE**
>
> Keyboard navigation respects precision: arrow keys increment by 1.0 for `Full`, 0.5 for `Half`, and 0.1 for `Exact`.

Shapes

Choose from built-in shapes or provide a custom geometry:

- `Star` (default)
- `Heart`
- `Circle`

- `Diamond`
- `Custom` - Uses the `CustomGeometry` property

```
<StackPanel Spacing="12">
    <nova:RatingControl Shape="Star" Value="3" />
    <nova:RatingControl Shape="Heart" Value="4" RatedFill="#E91E63" />
    <nova:RatingControl Shape="Circle" Value="2" />
    <nova:RatingControl Shape="Diamond" Value="5" />
    <nova:RatingControl Shape="Custom"
                        CustomGeometry="M 12,2 L 2,22 L 22,22 Z"
                        Value="3" />
</StackPanel>
```

Colors and styling

Customize the appearance using fill and stroke properties for both rated and unrated states. A separate set of preview colors can highlight items during hover.

```
<nova:RatingControl Value="4"
                    ItemSize="40"
                    ItemSpacing="10"
                    RatedFill="#00BCD4"
                    UnratedFill="#B2EBF2"
                    PreviewFill="#4DD0E1"
                    StrokeThickness="1"
                    RatedStroke="#0097A7" />
```

Property	Description
`RatedFill`	Fill brush for rated (active) items
`UnratedFill`	Fill brush for unrated (inactive) items
`RatedStroke`	Stroke brush for rated items
`UnratedStroke`	Stroke brush for unrated items
`PreviewFill`	Fill brush shown when hovering
`PreviewStroke`	Stroke brush shown when hovering
`StrokeThickness`	Thickness of the item stroke

Size and layout

Control the size and spacing of rating items, and choose between horizontal or vertical orientation.

```
<StackPanel Spacing="12">
    <nova:RatingControl Value="3" ItemSize="24" ItemSpacing="4" />
    <nova:RatingControl Value="3" ItemSize="48" ItemSpacing="12" />
    <nova:RatingControl Value="3" Orientation="Vertical" />
</StackPanel>
```

Item count

Set the number of rating items with `ItemCount`. The default is 5, but any positive number is supported.

```
<StackPanel Spacing="12">
    <nova:RatingControl Value="2" ItemCount="3" />
    <nova:RatingControl Value="7" ItemCount="10" ItemSize="20" />
</StackPanel>
```

Read-only mode

Use `IsReadOnly` to display a rating without allowing user interaction. This is useful for showing existing ratings.

```
<nova:RatingControl Value="3.7" IsReadOnly="True" Precision="Exact" />
```

Handle value changes

Subscribe to the `ValueChanged` event to respond when the user changes the rating.

```
<nova:RatingControl x:Name="MyRating"
                    Value="0"
                    ValueChanged="OnRatingValueChanged" />
```

```
 private void OnRatingValueChanged(object? sender, RoutedEventArgs e)
 {
     if (sender is RatingControl rating)
     {
         Debug.WriteLine($"New rating: {rating.Value}");
     }
 }
```

Keyboard support

The control is fully accessible via keyboard:

Key	Action
`Right` / `Up`	Increase value by step

Key	Action
`Left` / `Down`	Decrease value by step
`Home`	Set value to 0
`End`	Set value to maximum

The step size depends on the `Precision` setting (1.0, 0.5, or 0.1).

Properties

Property	Type	Default	Description
`Value`	`double`	`0`	Current rating value
`ItemCount`	`int`	`5`	Number of rating items
`Precision`	`RatingPrecision`	`Full`	Selection precision (Full, Half, Exact)
`IsReadOnly`	`bool`	`false`	Whether the control is read-only
`Shape`	`RatingShape`	`Star`	Shape of rating items
`CustomGeometry`	`Geometry`	`null`	Custom geometry when Shape is Custom
`ItemSize`	`double`	`32`	Size of each rating item
`ItemSpacing`	`double`	`6`	Spacing between items
`Orientation`	`Orientation`	`Horizontal`	Layout orientation
`RatedFill`	`IBrush`	`Gold`	Fill for rated items
`UnratedFill`	`IBrush`	`LightGray`	Fill for unrated items
`RatedStroke`	`IBrush`	`null`	Stroke for rated items
`UnratedStroke`	`IBrush`	`null`	Stroke for unrated items
`PreviewFill`	`IBrush`	`Orange`	Fill during hover preview
`PreviewStroke`	`IBrush`	`null`	Stroke during hover preview
`StrokeThickness`	`double`	`0`	Stroke thickness

Events

Event	Description
`ValueChanged`	Raised when the `Value` property changes

ResponsivePanel

`ResponsivePanel` is an adaptive layout panel that selectively displays its children based on the available width and specified breakpoints. It enables simplified "Mobile vs Desktop" layout switching directly in XAML.

Basic Usage

The control works by attaching a `Condition` to its children. Only children matching the current size class are visible; others are hidden (collapsed) and do not participate in layout.

```
<nova:ResponsivePanel>

    <!-- Visible on Mobile (< 600px) -->
    <StackPanel nova:ResponsivePanel.Condition="Narrow">
        <TextBlock Text="Mobile View"/>
    </StackPanel>

    <!-- Visible on Tablet/Desktop (>= 600px) -->
    <Grid nova:ResponsivePanel.Condition="Normal | Wide">
        <TextBlock Text="Desktop View"/>
    </Grid>

</nova:ResponsivePanel>
```

Breakpoints

You can customize the breakpoints using the `NarrowBreakpoint` and `WideBreakpoint` properties on the panel.

Property	Default	Description
`NarrowBreakpoint`	`600`	Width below this is considered `Narrow`.
`WideBreakpoint`	`900`	Width above this is considered `Wide`. Width between Narrow and Wide is `Normal`.

Conditions

The `ResponsivePanel.Condition` attached property accepts a flag enum `ResponsiveBreakpoint`:

- `Narrow`
- `Normal`

- `Wide`
- `All` (Default)

You can combine them using utility syntax if supported, or by standard piping in code-behind. In XAML, the parser usually supports comma-separated values for flags in some frameworks, but here you typically specify one. If you need multiple, you can use:

```
<!-- Example if flag parsing is supported by XAML compiler for this enum -->
<Border nova:ResponsivePanel.Condition="Narrow, Normal" ... />
```

Lazy Layout & Performance

`ResponsivePanel` uses a **Lazy Layout** strategy:

- Hidden views have `IsVisible` set to `false`.
- They cost **zero** layout / render time.
- They **retain state** (e.g., text in a TextBox is preserved when switching views).
- They are **eagerly loaded** (created in memory), offering a simpler syntax than DataTemplates.

Shimmer

The `Shimmer` control shows a lightweight skeleton while your content is loading. It inspects the visual tree beneath it to draw shapes that match text, images, and buttons, then animates a gradient sweep over the placeholders.

Add a Shimmer placeholder

Wrap the content that loads asynchronously in a `Shimmer`. Toggle `IsLoading` to switch between the placeholder and the real content.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="clr-namespace:Nova.Avalonia.UI.Controls;assembly=Nova.Avalonia.

    <nova:Shimmer IsLoading="True" LoadingText="Loading profile">
        <StackPanel Spacing="8">
            <TextBlock FontSize="20" Text="Profile" />
            <Border Height="160" CornerRadius="12" Background="#202020" />
            <Button Content="Refresh" Width="120" />
        </StackPanel>
    </nova:Shimmer>
</UserControl>
```

When `IsLoading` is `True`, Shimmer disables hit testing on the child content and announces the loading state to screen readers.

Customize the effect

Use the following properties to align the effect with your theme:

- `HighlightBrush` sets the moving gradient. Bind it to a `DynamicResource` for theme switching.
- `ShimmerOpacity` adjusts the overlay opacity. The default is `0.5`.
- `ShimmerAngle` sets the gradient angle in degrees.
- `LoadingText` defines the automation name announced while loading.

```
<nova:Shimmer IsLoading="True"
              HighlightBrush="{DynamicResource AccentGradient}"
              ShimmerOpacity="0.35"
              ShimmerAngle="12"
              LoadingText="Loading dashboard cards" />
```

Show loaded content

Set `IsLoading` to `False` when your data is ready. The child content becomes visible and interactive, and the automation name is cleared.

```
  // ViewModel
  public bool IsBusy { get; set; }

<nova:Shimmer IsLoading="{Binding IsBusy}">
    <ItemsControl Items="{Binding Orders}" />
</nova:Shimmer>
```

StaggeredPanel

The `StaggeredPanel` arranges items into columns based on their height, filling the shortest column first. This creates a staggered "masonry" layout that is efficient for displaying items with varying heights, such as cards or images.

Basic Usage

The `StaggeredPanel` is typically used as the `ItemsPanel` of an `ItemsControl` or `ListBox`. Set the `DesiredColumnWidth` to control how many columns are created based on the available space.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <ScrollViewer>
        <ItemsControl ItemsSource="{Binding Items}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <nova:StaggeredPanel DesiredColumnWidth="200"
                                         ColumnSpacing="10"
                                         RowSpacing="10" />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>

            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Border Height="{Binding Height}" Background="{Binding Brush}" Corne
                        <TextBlock Text="{Binding Title}" Margin="10" />
                    </Border>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </ScrollViewer>
</UserControl>
```

Configure Columns

The control automatically determines the number of columns by dividing the available width by the `DesiredColumnWidth` (plus spacing). It ensures at least one column is always present.

- **DesiredColumnWidth**: The target width for each column. The actual width may be slightly larger to fill the available space.
- **ColumnSpacing**: The horizontal space between columns.
- **RowSpacing**: The vertical space between items in the same column.

```
<nova:StaggeredPanel DesiredColumnWidth="150"
                     ColumnSpacing="20"
                     RowSpacing="20"
                     Padding="10">
    <!-- Items... -->
</nova:StaggeredPanel>
```

Layout Logic

Items are added sequentially to the column with the **minimum total height**. This prevents large gaps and maintains a balanced visual weight across the panel.

> **NOTE**
>
> If the available width is infinite (e.g., inside a horizontal `StackPanel` or `ScrollViewer`), the panel will default to using the `DesiredColumnWidth` for every item, potentially creating a single horizontal row or one column per item depending on exact constraints. It is best used within a container that provides a constrained width (like a vertical `ScrollViewer`).

Properties

Property	Type	Default	Description
`DesiredColumnWidth`	`double`	`250`	The desired width for each column.
`ColumnSpacing`	`double`	`0`	The distance between columns.
`RowSpacing`	`double`	`0`	The distance between items in the same column.
`Padding`	`Thickness`	`0`	The padding inside the panel border.

Examples

Dynamic Resizing

The `StaggeredPanel` responds to window resizing by recalculating the number of columns.

```
<nova:StaggeredPanel DesiredColumnWidth="300" HorizontalAlignment="Stretch">
    <!-- As the window widens, more columns will appear. -->
</nova:StaggeredPanel>
```

Direct Children

You can also use `StaggeredPanel` directly with children defined in XAML:

```
<nova:StaggeredPanel DesiredColumnWidth="120" ColumnSpacing="5" RowSpacing="5">
    <Border Height="50" Background="Red" />
    <Border Height="100" Background="Green" />
    <Border Height="75" Background="Blue" />
    <Border Height="120" Background="Orange" />
</nova:StaggeredPanel>
```

Virtualized Version

For large datasets (100+ items), use `VirtualizedStaggeredLayout` with `ItemsRepeater` for efficient scrolling:

```
<ScrollViewer>
    <ItemsControl ItemsSource="{Binding LargeCollection}">
        <ItemsControl.ItemsPanel>
            <ItemsPanelTemplate>
                <nova:VirtualizingStaggeredPanel
                    DesiredColumnWidth="200"
                    ColumnSpacing="10"
                    RowSpacing="10" />
            </ItemsPanelTemplate>
        </ItemsControl.ItemsPanel>
        <ItemsControl.ItemTemplate>
            <DataTemplate>
                <Border Height="{Binding Height}" Background="{Binding Brush}" CornerRad
                    <TextBlock Text="{Binding Title}" Margin="10" />
                </Border>
            </DataTemplate>
        </ItemsControl.ItemTemplate>
    </ItemsControl>
</ScrollViewer>
```

VirtualizingStaggeredPanel Properties

Property	Type	Default	Description
`DesiredColumnWidth`	`double`	`250`	The desired width for each column.
`ColumnSpacing`	`double`	`0`	The distance between columns.
`RowSpacing`	`double`	`0`	The distance between items in the same column.

Performance Features

The `VirtualizingStaggeredPanel` is optimized for large datasets:

- **Container Recycling**: Off-screen items are hidden and reused, not destroyed
- **Minimal Allocations**: Reusable collections avoid per-frame allocations
- **2x Viewport Buffer**: Items above and below the viewport are pre-realized for smooth scrolling
- **Pool Size Limit**: Maximum 20 recycled containers per type to prevent memory bloat

> **TIP**
>
> For best performance with 500+ items, ensure your `ItemTemplate` is lightweight and avoid expensive bindings or effects on each item.

TimelinePanel

The `TimelinePanel` arranges child elements in a timeline or step-by-step flow, with connecting lines between items. It's perfect for wizards, process flows, order tracking, or historical timelines.

Basic Usage

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:TimelinePanel Orientation="Vertical" Spacing="20">
        <Border Background="#3498DB" CornerRadius="20" Width="40" Height="40">
            <TextBlock Text="1" Foreground="White" HorizontalAlignment="Center" Vertical
        </Border>
        <Border Background="#2ECC71" CornerRadius="20" Width="40" Height="40">
            <TextBlock Text="2" Foreground="White" HorizontalAlignment="Center" Vertical
        </Border>
        <Border Background="#E74C3C" CornerRadius="20" Width="40" Height="40">
            <TextBlock Text="3" Foreground="White" HorizontalAlignment="Center" Vertical
        </Border>
    </nova:TimelinePanel>
</UserControl>
```

Orientation

The panel supports both vertical and horizontal layouts:

- **Vertical**: Steps flow from top to bottom (default)
- **Horizontal**: Steps flow from left to right

```
<nova:TimelinePanel Orientation="Horizontal" Spacing="30">
    <!-- Steps arranged horizontally -->
</nova:TimelinePanel>
```

Connector Lines

The panel automatically draws connector lines between items. Customize their appearance with:

Property	Type	Default	Description
`ConnectorBrush`	`IBrush`	`Gray`	Color of the connecting lines.
`ConnectorThickness`	`double`	`2`	Thickness of the connecting lines.

```
<nova:TimelinePanel ConnectorBrush="#3498DB" ConnectorThickness="3">
    <!-- Steps with blue connectors -->
```

```
</nova:TimelinePanel>
```

Properties

Property	Type	Default	Description
`Orientation`	`Orientation`	`Vertical`	Layout direction (Vertical or Horizontal).
`Spacing`	`double`	`10`	Space between timeline items.
`ConnectorBrush`	`IBrush`	`Gray`	Brush for connector lines.
`ConnectorThickness`	`double`	`2`	Thickness of connector lines.

Example: Order Tracking

```
<nova:TimelinePanel Orientation="Vertical" Spacing="15" ConnectorBrush="#27AE60">
    <StackPanel Orientation="Horizontal" Spacing="10">
        <Ellipse Width="20" Height="20" Fill="#27AE60"/>
        <TextBlock Text="Order Placed" VerticalAlignment="Center"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Spacing="10">
        <Ellipse Width="20" Height="20" Fill="#27AE60"/>
        <TextBlock Text="Processing" VerticalAlignment="Center"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Spacing="10">
        <Ellipse Width="20" Height="20" Fill="#BDC3C7"/>
        <TextBlock Text="Shipped" VerticalAlignment="Center" Opacity="0.5"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal" Spacing="10">
        <Ellipse Width="20" Height="20" Fill="#BDC3C7"/>
        <TextBlock Text="Delivered" VerticalAlignment="Center" Opacity="0.5"/>
    </StackPanel>
</nova:TimelinePanel>
```

VariableSizeWrapPanel

The `VariableSizeWrapPanel` (formerly MetroPanel) lays out items based on a uniform unit size, but allows individual items to span multiple rows and columns. This layout style is famously used in the Windows 8/10 Start Screen "Metro" design.

Basic Usage

Define the basic `ItemHeight` and `ItemWidth`. Items default to a 1x1 span of this unit size. Use `VariableSizeWrapPanel.RowSpan` and `VariableSizeWrapPanel.ColumnSpan` to make items larger.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:VariableSizeWrapPanel ItemHeight="100" ItemWidth="100" Orientation="Horizontal

        <!-- Standard 1x1 Item (100x100) -->
        <Border Background="Red" />

        <!-- Large 2x2 Item (200x200) -->
        <Border Background="Blue"
                nova:VariableSizeWrapPanel.RowSpan="2"
                nova:VariableSizeWrapPanel.ColumnSpan="2" />

        <!-- Wide 2x1 Item (200x100) -->
        <Border Background="Green"
                nova:VariableSizeWrapPanel.ColumnSpan="2" />

    </nova:VariableSizeWrapPanel>
</UserControl>
```

Important Considerations

- The panel fills available space in the `Orientation` direction until `MaximumRowsOrColumns` is reached (or space runs out), then wraps to the next line/column.
- **Item Order**: The panel tries to fit items in the "holes" left by larger items if the `Orientation` allows, but generally places items sequentially. Layout gaps may occur if items dimensions don't sum up perfectly to the line capacity.

Properties

Property	Type	Default	Description
`ItemHeight`	`double`	`NaN`	The base height of a 1-unit block.

Property	Type	Default	Description
`ItemWidth`	`double`	`NaN`	The base width of a 1-unit block.
`Orientation`	`Orientation`	`Vertical`	The direction in which items are arranged before wrapping.
`MaximumRowsOrColumns`	`int`	`-1`	The maximum number of units in the non-wrapping dimension before forcing a wrap.

Attached Properties

Property	Type	Description
`RowSpan`	`int`	How many vertical units the item spans. Default is 1.
`ColumnSpan`	`int`	How many horizontal units the item spans. Default is 1.

Virtualized Version

For large datasets (100+ items), use `VirtualizedVariableSizeWrapLayout` with `ItemsRepeater` for efficient scrolling:

```
<ScrollViewer>
    <ItemsControl ItemsSource="{Binding LargeTileCollection}">
        <ItemsControl.ItemsPanel>
            <ItemsPanelTemplate>
                <nova:VirtualizingVariableSizeWrapPanel
                    TileSize="100"
                    Spacing="8"
                    Columns="4" />
            </ItemsPanelTemplate>
        </ItemsControl.ItemsPanel>
        <ItemsControl.ItemContainerTheme>
            <ControlTheme TargetType="ContentPresenter">
                <Setter Property="nova:VirtualizingVariableSizeWrapPanel.ColumnSpan" Val
                <Setter Property="nova:VirtualizingVariableSizeWrapPanel.RowSpan" Value=
            </ControlTheme>
        </ItemsControl.ItemContainerTheme>
        <ItemsControl.ItemTemplate>
            <DataTemplate>
                <Border Background="{Binding Brush}" CornerRadius="8">
                    <TextBlock Text="{Binding Title}"
                               HorizontalAlignment="Center" VerticalAlignment="Center"/>
                </Border>
```

```
            </DataTemplate>
        </ItemsControl.ItemTemplate>
    </ItemsControl>
</ScrollViewer>
```

VirtualizedVariableSizeWrapLayout Properties

Property	Type	Default	Description
`TileSize`	`double`	`100`	The base size of a single tile unit.
`Spacing`	`double`	`8`	The spacing between tiles.
`Columns`	`int`	`4`	The number of columns in the grid.

Attached Properties

Property	Type	Description
`RowSpan`	`int`	How many vertical units the item spans. Default is 1.
`ColumnSpan`	`int`	How many horizontal units the item spans. Default is 1.

Performance Features

The `VirtualizingVariableSizeWrapPanel` is optimized for large datasets:

- **Container Recycling**: Off-screen items are hidden and reused, not destroyed
- **Minimal Allocations**: Reusable collections avoid per-frame allocations
- **2x Viewport Buffer**: Items above and below the viewport are pre-realized for smooth scrolling
- **Pool Size Limit**: Maximum 20 recycled containers per type to prevent memory bloat

> **TIP**
>
> For best performance with 500+ items, ensure your `ItemTemplate` is lightweight and avoid expensive bindings or effects on each item.

StaggeredPanel

The `StaggeredPanel` arranges items into columns based on their height, filling the shortest column first. This creates a staggered "masonry" layout that is efficient for displaying items with varying heights, such as cards or images.

Basic Usage

The `StaggeredPanel` is typically used as the `ItemsPanel` of an `ItemsControl` or `ListBox`. Set the `DesiredColumnWidth` to control how many columns are created based on the available space.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <ScrollViewer>
        <ItemsControl ItemsSource="{Binding Items}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <nova:StaggeredPanel DesiredColumnWidth="200"
                                         ColumnSpacing="10"
                                         RowSpacing="10" />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>

            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Border Height="{Binding Height}" Background="{Binding Brush}" Corne
                        <TextBlock Text="{Binding Title}" Margin="10" />
                    </Border>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </ScrollViewer>
</UserControl>
```

Configure Columns

The control automatically determines the number of columns by dividing the available width by the `DesiredColumnWidth` (plus spacing). It ensures at least one column is always present.

- **DesiredColumnWidth**: The target width for each column. The actual width may be slightly larger to fill the available space.
- **ColumnSpacing**: The horizontal space between columns.
- **RowSpacing**: The vertical space between items in the same column.

```
<nova:StaggeredPanel DesiredColumnWidth="150"
                     ColumnSpacing="20"
                     RowSpacing="20"
                     Padding="10">
    <!-- Items... -->
</nova:StaggeredPanel>
```

Layout Logic

Items are added sequentially to the column with the **minimum total height**. This prevents large gaps and maintains a balanced visual weight across the panel.

> **NOTE**
>
> If the available width is infinite (e.g., inside a horizontal `StackPanel` or `ScrollViewer`), the panel will default to using the `DesiredColumnWidth` for every item, potentially creating a single horizontal row or one column per item depending on exact constraints. It is best used within a container that provides a constrained width (like a vertical `ScrollViewer`).

Properties

Property	Type	Default	Description
`DesiredColumnWidth`	`double`	`250`	The desired width for each column.
`ColumnSpacing`	`double`	`0`	The distance between columns.
`RowSpacing`	`double`	`0`	The distance between items in the same column.
`Padding`	`Thickness`	`0`	The padding inside the panel border.

Examples

Dynamic Resizing

The `StaggeredPanel` responds to window resizing by recalculating the number of columns.

```
<nova:StaggeredPanel DesiredColumnWidth="300" HorizontalAlignment="Stretch">
    <!-- As the window widens, more columns will appear. -->
</nova:StaggeredPanel>
```

Direct Children

You can also use `StaggeredPanel` directly with children defined in XAML:

```
<nova:StaggeredPanel DesiredColumnWidth="120" ColumnSpacing="5" RowSpacing="5">
    <Border Height="50" Background="Red" />
    <Border Height="100" Background="Green" />
    <Border Height="75" Background="Blue" />
    <Border Height="120" Background="Orange" />
</nova:StaggeredPanel>
```

Virtualized Version

For large datasets (100+ items), use `VirtualizedStaggeredLayout` with `ItemsRepeater` for efficient scrolling:

```
<ScrollViewer>
    <ItemsControl ItemsSource="{Binding LargeCollection}">
        <ItemsControl.ItemsPanel>
            <ItemsPanelTemplate>
                <nova:VirtualizingStaggeredPanel
                    DesiredColumnWidth="200"
                    ColumnSpacing="10"
                    RowSpacing="10" />
            </ItemsPanelTemplate>
        </ItemsControl.ItemsPanel>
        <ItemsControl.ItemTemplate>
            <DataTemplate>
                <Border Height="{Binding Height}" Background="{Binding Brush}" CornerRad
                    <TextBlock Text="{Binding Title}" Margin="10" />
                </Border>
            </DataTemplate>
        </ItemsControl.ItemTemplate>
    </ItemsControl>
</ScrollViewer>
```

VirtualizingStaggeredPanel Properties

Property	Type	Default	Description
`DesiredColumnWidth`	`double`	`250`	The desired width for each column.
`ColumnSpacing`	`double`	`0`	The distance between columns.
`RowSpacing`	`double`	`0`	The distance between items in the same column.

Performance Features

The `VirtualizingStaggeredPanel` is optimized for large datasets:

- **Container Recycling**: Off-screen items are hidden and reused, not destroyed
- **Minimal Allocations**: Reusable collections avoid per-frame allocations
- **2x Viewport Buffer**: Items above and below the viewport are pre-realized for smooth scrolling
- **Pool Size Limit**: Maximum 20 recycled containers per type to prevent memory bloat

> **TIP**
>
> For best performance with 500+ items, ensure your `ItemTemplate` is lightweight and avoid expensive bindings or effects on each item.

VariableSizeWrapPanel

The `VariableSizeWrapPanel` (formerly MetroPanel) lays out items based on a uniform unit size, but allows individual items to span multiple rows and columns. This layout style is famously used in the Windows 8/10 Start Screen "Metro" design.

Basic Usage

Define the basic `ItemHeight` and `ItemWidth`. Items default to a 1x1 span of this unit size. Use `VariableSizeWrapPanel.RowSpan` and `VariableSizeWrapPanel.ColumnSpan` to make items larger.

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:nova="using:Nova.Avalonia.UI.Controls">

    <nova:VariableSizeWrapPanel ItemHeight="100" ItemWidth="100" Orientation="Horizontal

        <!-- Standard 1x1 Item (100x100) -->
        <Border Background="Red" />

        <!-- Large 2x2 Item (200x200) -->
        <Border Background="Blue"
                nova:VariableSizeWrapPanel.RowSpan="2"
                nova:VariableSizeWrapPanel.ColumnSpan="2" />

        <!-- Wide 2x1 Item (200x100) -->
        <Border Background="Green"
                nova:VariableSizeWrapPanel.ColumnSpan="2" />

    </nova:VariableSizeWrapPanel>
</UserControl>
```

Important Considerations

- The panel fills available space in the `Orientation` direction until `MaximumRowsOrColumns` is reached (or space runs out), then wraps to the next line/column.
- **Item Order**: The panel tries to fit items in the "holes" left by larger items if the `Orientation` allows, but generally places items sequentially. Layout gaps may occur if items dimensions don't sum up perfectly to the line capacity.

Properties

Property	Type	Default	Description
`ItemHeight`	`double`	`NaN`	The base height of a 1-unit block.

Property	Type	Default	Description
`ItemWidth`	`double`	`NaN`	The base width of a 1-unit block.
`Orientation`	`Orientation`	`Vertical`	The direction in which items are arranged before wrapping.
`MaximumRowsOrColumns`	`int`	`-1`	The maximum number of units in the non-wrapping dimension before forcing a wrap.

Attached Properties

Property	Type	Description
`RowSpan`	`int`	How many vertical units the item spans. Default is 1.
`ColumnSpan`	`int`	How many horizontal units the item spans. Default is 1.

Virtualized Version

For large datasets (100+ items), use `VirtualizedVariableSizeWrapLayout` with `ItemsRepeater` for efficient scrolling:

```
<ScrollViewer>
    <ItemsControl ItemsSource="{Binding LargeTileCollection}">
        <ItemsControl.ItemsPanel>
            <ItemsPanelTemplate>
                <nova:VirtualizingVariableSizeWrapPanel
                    TileSize="100"
                    Spacing="8"
                    Columns="4" />
            </ItemsPanelTemplate>
        </ItemsControl.ItemsPanel>
        <ItemsControl.ItemContainerTheme>
            <ControlTheme TargetType="ContentPresenter">
                <Setter Property="nova:VirtualizingVariableSizeWrapPanel.ColumnSpan" Val
                <Setter Property="nova:VirtualizingVariableSizeWrapPanel.RowSpan" Value=
            </ControlTheme>
        </ItemsControl.ItemContainerTheme>
        <ItemsControl.ItemTemplate>
            <DataTemplate>
                <Border Background="{Binding Brush}" CornerRadius="8">
                    <TextBlock Text="{Binding Title}"
                               HorizontalAlignment="Center" VerticalAlignment="Center"/>
                </Border>
```

```
            </DataTemplate>
        </ItemsControl.ItemTemplate>
    </ItemsControl>
</ScrollViewer>
```

VirtualizedVariableSizeWrapLayout Properties

Property	Type	Default	Description
`TileSize`	`double`	`100`	The base size of a single tile unit.
`Spacing`	`double`	`8`	The spacing between tiles.
`Columns`	`int`	`4`	The number of columns in the grid.

Attached Properties

Property	Type	Description
`RowSpan`	`int`	How many vertical units the item spans. Default is 1.
`ColumnSpan`	`int`	How many horizontal units the item spans. Default is 1.

Performance Features

The `VirtualizingVariableSizeWrapPanel` is optimized for large datasets:

- **Container Recycling**: Off-screen items are hidden and reused, not destroyed
- **Minimal Allocations**: Reusable collections avoid per-frame allocations
- **2x Viewport Buffer**: Items above and below the viewport are pre-realized for smooth scrolling
- **Pool Size Limit**: Maximum 20 recycled containers per type to prevent memory bloat

> **TIP**
>
> For best performance with 500+ items, ensure your `ItemTemplate` is lightweight and avoid expensive bindings or effects on each item.