

Namespace RTB.Blazor

Enums

[Position](#)

Represents the position of a UI element relative to a reference element or container.

Enum Position

Namespace: [RTB.Blazor](#)

Assembly: RTB.Blazor.dll

Represents the position of a UI element relative to a reference element or container.

```
public enum Position
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Bottom = 2

Positioned below the reference element.

Left = 3

Positioned to the left of the reference element.

Right = 1

Positioned to the right of the reference element.

Top = 0

Positioned above the reference element.

Remarks

Commonly used by RTB.Blazor components to control alignment and placement such as tooltips, popovers, context menus, and flyouts.

Namespace RTB.Blazor.Components

Classes

[RTBComponent](#)

Base component for RTB Blazor components.

[TabItem](#)

A component representing a single tab item within a tab group.

Class RTBComponent

Namespace: [RTB.Blazor.Components](#)

Assembly: RTB.Blazor.dll

Base component for RTB Blazor components.

```
public abstract class RTBComponent : ComponentBase, IComponent,
IHandleEvent, IHandleAfterRender
```

Inheritance

[object](#) ← [ComponentBase](#) ← RTBComponent

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#)

Derived

[ColumnBase<TRow>](#), [StackBase](#), [TabItem](#), [BusyIndicator](#), [DialogBase](#)

Inherited Members

[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitialized\(\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSet\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Example: efficiently update a bound value and re-render only when it changes.

```
private string? _text;
[Parameter] public string? Text
{
    get => _text;
```

```
    set => SetProperty(ref _text, value);
}
```

Example: build a class attribute from optional parts.

```
var classes = CombineClass("rtb-btn", isPrimary ? "rtb-btn--primary" : null, Class);
```

Remarks

Provides common conveniences for derived components:

- Optional [Id](#) and [Class](#) parameters for markup.
- [CombineClass\(params string?\[\]\)](#) to safely compose CSS class strings.
- [SetProperty< TValue >\(ref TValue, TValue\)](#) to update a backing field and schedule a re-render only when the value changes.
- [StatefulAction\(Action\)](#) to run arbitrary logic and then schedule a re-render.

Re-rendering is scheduled via [InvokeAsync\(Func< Task >\)](#) to ensure it occurs on the correct synchronization context for Blazor.

Properties

Class

Optional CSS class(es) that derived components may append to their root element.

```
[Parameter]
public string? Class { get; set; }
```

Property Value

[string](#)

Id

Optional id attribute value that derived components may apply to their root element.

```
[Parameter]
```

```
public string? Id { get; set; }
```

Property Value

[string](#)

Methods

CombineClass(params string?[])

Combines CSS class name fragments into a single space-delimited string, ignoring null, empty, and whitespace-only entries.

```
protected static string CombineClass(params string?[] parts)
```

Parameters

parts [string](#)[]

The class name fragments to combine.

Returns

[string](#)

A space-delimited string containing only non-empty fragments.

SetProperty< TValue >(ref TValue, TValue)

Sets a backing field to a new value and, if the value changed, schedules a component re-render.

```
public void SetProperty< TValue >(ref TValue field, TValue value)
```

Parameters

field TValue

A reference to the backing field to update.

value TValue

The new value to assign to the field.

Type Parameters

TValue

The type of the backing field.

Remarks

If the new value equals the current one (by [Default](#)), no re-render is scheduled.

StatefulAction(Action)

Executes the provided action and then schedules a component re-render.

```
public void StatefulAction(Action action)
```

Parameters

action [Action](#)

The action to execute.

Remarks

Exceptions thrown by **action** will propagate to the caller.

Class TabItem

Namespace: [RTB.Blazor.Components](#)

Assembly: RTB.Blazor.dll

A component representing a single tab item within a tab group.

```
public class TabItem : RTBComponent, IComponent, IHandleEvent,  
IHandleAfterRender, IDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← TabItem

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IDisposable](#)

Inherited Members

[RTBComponent.Id](#), [RTBComponent.Class](#), [RTBComponent.CombineClass\(params string\[\]\)](#),
[RTBComponent SetProperty< TValue >\(ref TValue, TValue\)](#), [RTBComponent.StatefulAction\(Action\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitialized\(\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSetAsync\(\)](#),
[ComponentBase.StateHasChanged\(\)](#), [ComponentBase.ShouldRender\(\)](#),
[ComponentBase.OnAfterRender\(bool\)](#), [ComponentBase.OnAfterRenderAsync\(bool\)](#),
[ComponentBase.InvokeAsync\(Action\)](#), [ComponentBase.InvokeAsync\(Func< Task >\)](#),
[ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Fields

Guid

A unique identifier for the tab item.

```
public readonly Guid Guid
```

Field Value

[Guid ↗](#)

Properties

ChildContent

The child content of the tab item, typically used for defining the tab's label or header.

```
[Parameter]  
public RenderFragment ChildContent { get; set; }
```

Property Value

[RenderFragment ↗](#)

TabContent

The content to be displayed when the tab is active.

```
[Parameter]  
public RenderFragment? TabContent { get; set; }
```

Property Value

[RenderFragment ↗](#)

TabGroup

The tab group that this tab item belongs to.

```
[CascadingParameter]  
public IRegister<TabItem>? TabGroup { get; set; }
```

Property Value

Title

The title of the tab item.

```
[Parameter]  
[EditorRequired]  
public required string Title { get; set; }
```

Property Value

[string](#) ↗

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

OnParametersSet()

Method invoked when the component has received parameters from its parent in the render tree, and the incoming values have been assigned to properties.

```
protected override void OnParametersSet()
```

Namespace RTB.Blazor.Components.DataGrid

Classes

[ColumnBase<TRow>](#)

Base class for DataGrid columns.

[DataColumn<TRow, TValue>](#)

A column that displays a value from the row using a provided function.

[ViewColumn<TRow>](#)

A column that uses a RenderFragment to display custom content.

Interfaces

[IColumn<TRow>](#)

Column interface for DataGrid.

Class ColumnBase<TRow>

Namespace: [RTB.Blazor.Components.DataGrid](#)

Assembly: RTB.Blazor.dll

Base class for DataGrid columns.

```
public abstract class ColumnBase<TRow> : RTBComponent, IComponent, IHandleEvent,
IHandleAfterRender, IColumn<TRow>, IDisposable
```

Type Parameters

TRow

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← ColumnBase<TRow>

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IColumn](#)<TRow>, [IDisposable](#)

Derived

[DataColumn](#)<TRow, TValue>, [ViewColumn](#)<TRow>

Inherited Members

[RTBComponent.Id](#), [RTBComponent.Class](#), [RTBComponent.CombineClass\(params string\[\]\)](#),
[RTBComponent SetProperty](#)<TValue>(ref TValue, TValue), [RTBComponent.StatefulAction\(Action\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitialized\(\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSetAsync\(\)](#),
[ComponentBase.StateHasChanged\(\)](#), [ComponentBase.ShouldRender\(\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Fields

Guid

The unique identifier for the column.

```
public readonly Guid Guid
```

Field Value

[Guid](#) ↗

Properties

CanSort

Optional Sorting key function for the column.

```
public bool CanSort { get; }
```

Property Value

[bool](#) ↗

DefaultSortDescending

Indicates if the default sort order is descending.

[Parameter]

```
public bool DefaultSortDescending { get; set; }
```

Property Value

[bool](#) ↗

HeadContent

Custom content for the column header.

```
[Parameter]
public RenderFragment? HeadContent { get; set; }
```

Property Value

[RenderFragment](#)

MaxWidth

Optional maximum width for the column.

```
[Parameter]
public string? MaxWidth { get; set; }
```

Property Value

[string](#)

MinWidth

Optional minimum width for the column.

```
[Parameter]
public string? MinWidth { get; set; }
```

Property Value

[string](#)

Name

The unique identifier for the column.

```
[Parameter]
public string Name { get; set; }
```

Property Value

[string](#) ↗

ParentGrid

Reference to the parent grid for registration.

```
[CascadingParameter]  
public IRegister<ColumnBase<TRow>>? ParentGrid { get; set; }
```

Property Value

[IRegister<ColumnBase<TRow>>](#)

SortDescending

Indicates if the current sort order is descending.

```
public bool SortDescending { get; set; }
```

Property Value

[bool](#) ↗

SortKey

Optional Sorting key function for the column.

```
[Parameter]  
public Func<TRow, IComparable>? SortKey { get; set; }
```

Property Value

[Func](#) ↗ <TRow, [IComparable](#) ↗ >

Spacings

Optional spacings for the column header.

```
[Parameter]
public Spacing[] Spacings { get; set; }
```

Property Value

[Spacing\[\]](#)

Width

Optional content for the column header.

```
[Parameter]
public string? Width { get; set; }
```

Property Value

[string](#) ↗

Methods

Dispose()

Unregisters the column from the parent grid when disposed.

```
public void Dispose()
```

OnAfterRender(bool)

Sets the initial sort order after the first render.

```
protected override void OnAfterRender(bool firstRender)
```

Parameters

`firstRender` [bool](#)

OnParametersSet()

Registers the column with the parent grid when parameters are set.

```
protected override void OnParametersSet()
```

RenderCell(RenderTreeBuilder, TRow, int)

Renders a cell for the given row and column index.

```
public abstract void RenderCell(RenderTreeBuilder builder, TRow row, int col)
```

Parameters

`builder` [RenderTreeBuilder](#)

`row` TRow

`col` [int](#)

RenderHeader(RenderTreeBuilder, int)

Renders the header for the column.

```
public abstract void RenderHeader(RenderTreeBuilder builder, int col)
```

Parameters

`builder` [RenderTreeBuilder](#)

`col` [int](#)

Class DataColumn<TRow, TValue>

Namespace: [RTB.Blazor.Components.DataGrid](#)

Assembly: RTB.Blazor.dll

A column that displays a value from the row using a provided function.

```
public class DataColumn<TRow, TValue> : ColumnBase<TRow>, IComponent, IHandleEvent,
IHandleAfterRender, IColumn<TRow>, IDisposable
```

Type Parameters

TRow

TValue

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← [ColumnBase](#)<TRow> ← [DataColumn](#)<TRow, TValue>

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IColumn](#)<TRow>, [IDisposable](#)

Inherited Members

[ColumnBase](#)<TRow>.Guid, [ColumnBase](#)<TRow>.ParentGrid, [ColumnBase](#)<TRow>.Name,
[ColumnBase](#)<TRow>.HeadContent, [ColumnBase](#)<TRow>.Width, [ColumnBase](#)<TRow>.MinWidth,
[ColumnBase](#)<TRow>.MaxWidth, [ColumnBase](#)<TRow>.Spacings, [ColumnBase](#)<TRow>.SortKey,
[ColumnBase](#)<TRow>.DefaultSortDescending, [ColumnBase](#)<TRow>.CanSort,
[ColumnBase](#)<TRow>.SortDescending, [ColumnBase](#)<TRow>.OnParametersSet(),
[ColumnBase](#)<TRow>.OnAfterRender(bool), [ColumnBase](#)<TRow>.Dispose(), [RTBComponent](#).Id,
[RTBComponent](#).Class, [RTBComponent](#).CombineClass(params string[]),
[RTBComponent](#). SetProperty<TValue>(ref TValue, TValue), [RTBComponent](#).StatefulAction(Action),
[ComponentBase](#).BuildRenderTree(RenderTreeBuilder) , [ComponentBase](#).OnInitialized() ,
[ComponentBase](#).OnParametersSetAsync() , [ComponentBase](#).StateHasChanged() ,
[ComponentBase](#).ShouldRender() , [ComponentBase](#).OnAfterRenderAsync(bool) ,
[ComponentBase](#).InvokeAsync(Action) , [ComponentBase](#).InvokeAsync(Func<Task>) ,
[ComponentBase](#).DispatchExceptionAsync(Exception) ,
[ComponentBase](#).SetParametersAsync(ParameterView) , [ComponentBase](#).RendererInfo ,
[ComponentBase](#).Assets , [ComponentBase](#).AssignedRenderMode , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ValueFunc

Function to extract the value from the row to display in the cell.

```
[Parameter]  
[EditorRequired]  
public Func<TRow, TValue> ValueFunc { get; set; }
```

Property Value

[Func](#)<TRow, TValue>

Methods

OnInitializedAsync()

Initializes the component and sets up scoped styles.

```
protected override Task OnInitializedAsync()
```

Returns

[Task](#)

RenderCell(RenderTreeBuilder, TRow, int)

Renders a cell for the given row and column index.

```
public override void RenderCell(RenderTreeBuilder builder, TRow row, int col)
```

Parameters

builder [RenderTreeBuilder](#)

row TRow

col [int](#)

RenderHeader(RenderTreeBuilder, int)

Renders the header for the column. />

```
public override void RenderHeader(RenderTreeBuilder builder, int col)
```

Parameters

builder [RenderTreeBuilder](#)

col [int](#)

Interface IColumn<TRow>

Namespace: [RTB.Blazor.Components.DataGrid](#)

Assembly: RTB.Blazor.dll

Column interface for DataGrid.

```
public interface IColumn<TRow> : IDisposable
```

Type Parameters

TRow

Inherited Members

[IDisposable.Dispose\(\)](#) ↗

Properties

CanSort

Indicates if the column can be sorted.

```
bool CanSort { get; }
```

Property Value

[bool](#) ↗

DefaultSortDescending

Indicates if the default sort order is descending.

```
bool DefaultSortDescending { get; set; }
```

Property Value

[bool](#)

MaxWidth

Optional maximum width for the column.

```
string? MaxWidth { get; set; }
```

Property Value

[string](#)

MinWidth

Optional minimum width for the column.

```
string? MinWidth { get; set; }
```

Property Value

[string](#)

Name

The unique identifier for the column.

```
string Name { get; set; }
```

Property Value

[string](#)

ParentGrid

Reference to the parent grid for registration.

```
IRegister<ColumnBase<TRow>>? ParentGrid { get; set; }
```

Property Value

[IRegister<ColumnBase<TRow>>](#)

SortDescending

Indicates if the current sort order is descending.

```
bool SortDescending { get; set; }
```

Property Value

[bool](#)

SortKey

Optional Sorting key function for the column.

```
Func<TRow, IComparable>? SortKey { get; set; }
```

Property Value

[Func<TRow, IComparable>](#)

Spacings

Optional spacings for the column header.

```
Spacing[] Spacings { get; set; }
```

Property Value

[Spacing\[\]](#)

Width

Optional content for the column header.

```
string? Width { get; set; }
```

Property Value

[string](#)

Methods

RenderCell(RenderTreeBuilder, TRow, int)

Renders a cell for the given row and column index.

```
void RenderCell(RenderTreeBuilder builder, TRow row, int col)
```

Parameters

builder [RenderTreeBuilder](#)

row TRow

col [int](#)

RenderHeader(RenderTreeBuilder, int)

Renders the header for the column.

```
void RenderHeader(RenderTreeBuilder builder, int col)
```

Parameters

builder [RenderTreeBuilder](#)

col [int](#)

Class ViewColumn<TRow>

Namespace: [RTB.Blazor.Components.DataGrid](#)

Assembly: RTB.Blazor.dll

A column that uses a RenderFragment to display custom content.

```
public class ViewColumn<TRow> : ColumnBase<TRow>, IComponent, IHandleEvent,  
IHandleAfterRender, IColumn<TRow>, IDisposable
```

Type Parameters

TRow

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← [ColumnBase](#)<TRow> ← [ViewColumn](#)<TRow>

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IColumn](#)<TRow>, [IDisposable](#)

Inherited Members

[ColumnBase](#)<TRow>.Guid, [ColumnBase](#)<TRow>.ParentGrid, [ColumnBase](#)<TRow>.Name, [ColumnBase](#)<TRow>.HeadContent, [ColumnBase](#)<TRow>.Width, [ColumnBase](#)<TRow>.MinWidth, [ColumnBase](#)<TRow>.MaxWidth, [ColumnBase](#)<TRow>.Spacings, [ColumnBase](#)<TRow>.SortKey, [ColumnBase](#)<TRow>.DefaultSortDescending, [ColumnBase](#)<TRow>.CanSort, [ColumnBase](#)<TRow>.SortDescending, [ColumnBase](#)<TRow>.OnParametersSet(), [ColumnBase](#)<TRow>.OnAfterRender(bool), [ColumnBase](#)<TRow>.Dispose(), [RTBComponent](#).Id, [RTBComponent](#).Class, [RTBComponent](#).CombineClass(params string[]), [RTBComponent](#). SetProperty< TValue >(ref TValue, TValue), [RTBComponent](#).StatefulAction(Action), [ComponentBase](#).BuildRenderTree(RenderTreeBuilder), [ComponentBase](#).OnInitialized(), [ComponentBase](#).OnInitializedAsync(), [ComponentBase](#).OnParametersSetAsync(), [ComponentBase](#).StateHasChanged(), [ComponentBase](#).ShouldRender(), [ComponentBase](#).OnAfterRenderAsync(bool), [ComponentBase](#).InvokeAsync(Action), [ComponentBase](#).InvokeAsync(Func< Task >), [ComponentBase](#).DispatchExceptionAsync(Exception), [ComponentBase](#).SetParametersAsync(ParameterView), [ComponentBase](#).RendererInfo, [ComponentBase](#).Assets, [ComponentBase](#).AssignedRenderMode, [object](#).Equals([object](#)), [object](#).Equals([object](#), [object](#)), [object](#).GetHashCode(), [object](#).GetType(), [object](#).MemberwiseClone(), [object](#).ReferenceEquals([object](#), [object](#)), [object](#).ToString()

Properties

ChildContent

The content to render for each cell, provided with the current row.

```
[Parameter]  
[EditorRequired]  
public RenderFragment<TRow> ChildContent { get; set; }
```

Property Value

[RenderFragment](#)<TRow>

Methods

RenderCell(RenderTreeBuilder, TRow, int)

Renders a cell for the given row and column index.

```
public override void RenderCell(RenderTreeBuilder builder, TRow row, int col)
```

Parameters

builder [RenderTreeBuilder](#)

row TRow

col [int](#)

RenderHeader(RenderTreeBuilder, int)

Renders the header for the column.

```
public override void RenderHeader(RenderTreeBuilder builder, int col)
```

Parameters

builder [RenderTargetBuilder](#)

col [int](#)

Namespace RTB.Blazor.Components.Layout

Classes

[StackBase](#)

Base class for layout stack components. A stack arranges its child content along a main axis (row or column), optionally reversing order, applying gaps, wrapping, and alignment similar to CSS flexbox.

Class StackBase

Namespace: [RTB.Blazor.Components.Layout](#)

Assembly: RTB.Blazor.dll

Base class for layout stack components. A stack arranges its child content along a main axis (row or column), optionally reversing order, applying gaps, wrapping, and alignment similar to CSS flexbox.

```
public abstract class StackBase : RTBComponent, IComponent, IHandleEvent, IHandleAfterRender
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← StackBase

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#)

Inherited Members

[RTBComponent.Id](#), [RTBComponent.Class](#), [RTBComponent.CombineClass\(params string\[\]\)](#),
[RTBComponent SetProperty< TValue >\(ref TValue, TValue\)](#), [RTBComponent.StatefulAction\(Action\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitialized\(\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSet\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func< Task >\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

Concrete implementations should define the default direction by overriding [GetDirection\(\)](#). All parameters are optional. When null (or default false for booleans), the corresponding CSS is omitted, allowing styles to inherit or be controlled by parent components or CSS.

Properties

AlignItem

Alignment of items on the cross axis.

[Parameter]
`public Flex.Align? AlignItem { get; set; }`

Property Value

[Flex.Align?](#)

Remarks

Maps to CSS `align-items`. When null, no explicit alignment is emitted.

Background

Background color applied to the stack container.

[Parameter]
`public RTBColor? Background { get; set; }`

Property Value

[RTBColor?](#)

Remarks

Maps to CSS `background-color`. When null, no explicit background is emitted.

ChildContent

The content to be rendered inside the stack.

[Parameter]
`public RenderFragment ChildContent { get; set; }`

Property Value

[RenderFragment](#) ↗

Gap

The spacing (gap) between immediate children.

```
[Parameter]
public Spacing? Gap { get; set; }
```

Property Value

[Spacing?](#)

Remarks

Maps to CSS `gap`. When null, no explicit gap is emitted.

Grow

Grow factor when this stack acts as a flex item within a parent flex container.

```
[Parameter]
public int? Grow { get; set; }
```

Property Value

[int?](#)

Remarks

Maps to CSS `flex-grow`. When null, no explicit grow is emitted.

JustifyContent

Distribution of items along the main axis.

```
[Parameter]
public Flex.Justify? JustifyContent { get; set; }
```

Property Value

[FlexJustify?](#)

Remarks

Maps to CSS **justify-content**. When null, no explicit justification is emitted.

Margin

Outer spacing around the stack container.

[Parameter]

```
public Spacing[]? Margin { get; set; }
```

Property Value

[Spacing\[\]](#)

Remarks

Accepts 1 to 4 values using CSS margin shorthand semantics: 1 value: all sides; 2 values: vertical | horizontal; 3 values: top | horizontal | bottom; 4 values: top | right | bottom | left. When null or empty, no explicit margin is emitted.

Padding

Inner spacing of the stack container.

[Parameter]

```
public Spacing[]? Padding { get; set; }
```

Property Value

[Spacing\[\]](#)

Remarks

Accepts 1 to 4 values using CSS padding shorthand semantics: 1 value: all sides; 2 values: vertical | horizontal; 3 values: top | horizontal | bottom; 4 values: top | right | bottom | left. When null or empty, no explicit padding is emitted.

Reverse

When true, reverses the main axis order (e.g., row-reverse or column-reverse).

```
[Parameter]
public bool Reverse { get; set; }
```

Property Value

[bool](#)

Remarks

Combines with [GetDirection\(\)](#) to determine the effective axis.

Shrink

Shrink factor when this stack acts as a flex item within a parent flex container.

```
[Parameter]
public int? Shrink { get; set; }
```

Property Value

[int](#)?

Remarks

Maps to CSS `flex-shrink`. When null, no explicit shrink is emitted.

Wrap

Controls wrapping of child items along the main axis.

```
[Parameter]
public Flex.WrapMode? Wrap { get; set; }
```

Property Value

[Flex.WrapMode?](#)

Remarks

Maps to CSS `flex-wrap`: NoWrap, Wrap, or WrapReverse. When null, no explicit wrap rule is emitted.

Methods

GetDirection()

Determine the base axis direction of this stack.

```
protected abstract Flex.AxisDirection? GetDirection()
```

Returns

[Flex.AxisDirection?](#)

The intended axis direction ([Row](#) or [Column](#)), optionally including reverse variants. A null value allows defaults to be applied by derived components.

Remarks

Implementations may combine the returned direction with the [Reverse](#) flag to compute the effective main axis.

Namespace RTB.Blazor.Extensions

Classes

[ServiceCollectionExtension](#)

Provides extension methods for registering RTB Blazor services into the DI container.

[ServiceCollectionExtension.RTBConfig](#)

Configuration class for selecting theme and enabling optional RTB Blazor services.

Class ServiceCollectionExtension

Namespace: [RTB.Blazor.Extensions](#)

Assembly: RTB.Blazor.dll

Provides extension methods for registering RTB Blazor services into the DI container.

```
public static class ServiceCollectionExtension
```

Inheritance

[object](#) ← ServiceCollectionExtension

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

UseRTBBlazor(IServiceCollection, Action<RTBConfig>)

Registers RTB Blazor services and styling into the provided [IServiceCollection](#).

```
public static IServiceCollection UseRTBBlazor(this IServiceCollection services,  
Action<ServiceCollectionExtension.RTBConfig> config)
```

Parameters

services [IServiceCollection](#)

The service collection to register services into.

config [Action](#)<[ServiceCollectionExtension.RTBConfig](#)>

A configuration action used to specify which RTB services to include and which theme to use.

Returns

[IServiceCollection](#)

The same [IServiceCollection](#) for chaining.

Examples

```
builder.Services.UseRTBBlazor(cfg => cfg  
    .UseTheme<MyTheme>()  
    .UseDefaultServices());
```

Remarks

This method always registers the styled infrastructure via [UseRTBStyled\(\)](#). Optional services such as Busy Tracker, Dialog, Drag&Drop, Data Navigation, and Input are registered based on the flags set via [ServiceCollectionExtension.RTBConfig](#).

Class ServiceCollectionExtension.RTBConfig

Namespace: [RTB.Blazor.Extensions](#)

Assembly: RTB.Blazor.dll

Configuration class for selecting theme and enabling optional RTB Blazor services.

```
public class ServiceCollectionExtension.RTBConfig
```

Inheritance

[object](#) ← ServiceCollectionExtension.RTBConfig

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

UseBusyTracker()

Enables the Busy Tracker service.

```
public ServiceCollectionExtension.RTBConfig UseBusyTracker()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

UseDataNavigationService()

Enables the Data Navigation service.

```
public ServiceCollectionExtension.RTBConfig UseDataNavigationService()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

UseDefaultServices()

Enables all optional RTB services: Busy Tracker, Dialog, Drag&Drop, Data Navigation, and Input.

```
public ServiceCollectionExtension.RTBConfig UseDefaultServices()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

Examples

```
cfg.UseDefaultServices();
```

UseDialogService()

Enables the Dialog service.

```
public ServiceCollectionExtension.RTBConfig UseDialogService()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

UseDragDropService()

Enables the Drag & Drop service.

```
public ServiceCollectionExtension.RTBConfig UseDragDropService()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

UseInputService()

Enables the Input service.

```
public ServiceCollectionExtension.RTBConfig UseInputService()
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

UseTheme<T>()

Specifies the concrete theme type to be used for theming.

```
public ServiceCollectionExtension.RTBConfig UseTheme<T>() where T : ITheme
```

Returns

[ServiceCollectionExtension.RTBConfig](#)

The same [ServiceCollectionExtension.RTBConfig](#) instance for chaining.

Type Parameters

T

A type that implements [ITheme](#).

Examples

```
cfg.UseTheme<MyCustomTheme>();
```

Namespace RTB.Blazor.Interfaces

Interfaces

[IRegister<T>](#)

Defines a contract for components that can register and unregister items of type `T`.

Interface IRegister<T>

Namespace: [RTB.Blazor.Interfaces](#)

Assembly: RTB.Blazor.dll

Defines a contract for components that can register and unregister items of type **T**.

```
public interface IRegister<T>
```

Type Parameters

T

The type of item to register and unregister. Implementations may constrain or validate this type.

Remarks

Implementations should document thread-safety and whether operations are idempotent.

Methods

Register(T)

Registers the specified item with the implementing component.

```
void Register(T item)
```

Parameters

item **T**

The item to register. When **T** is a reference type, this parameter should not be null.

Unregister(T)

Unregisters the specified item from the implementing component.

```
void Unregister(T item)
```

Parameters

item T

The item to unregister. When T is a reference type, this parameter should not be null.

Namespace RTB.Blazor.Services.BusyIndicator

Classes

[BusyIndicator](#)

A Blazor component that conditionally renders a "busy" placeholder while work is in progress, as reported by an injected [IBusyTracker](#).

[BusyTracker](#)

Default implementation of [IBusyTracker](#) that tracks busy states by key using reference-counted scopes stored in concurrent dictionaries.

Structs

[BusyTracker.BusyToken](#)

Represents a tracked busy scope that automatically ends when disposed. Use with `using` to ensure the busy state is always released.

Interfaces

[IBusyTracker](#)

Abstraction for tracking and reacting to "busy" states during asynchronous or long-running operations. Useful for driving UI concerns such as spinners, disabling actions, or detecting background work.

Class BusyIndicator

Namespace: [RTB.Blazor.Services.BusyIndicator](#)

Assembly: RTB.Blazor.dll

A Blazor component that conditionally renders a "busy" placeholder while work is in progress, as reported by an injected [IBusyTracker](#).

```
public class BusyIndicator : RTBComponent, IComponent, IHandleEvent,  
IHandleAfterRender, IDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← BusyIndicator

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IDisposable](#)

Inherited Members

[RTBComponent.Id](#), [RTBComponent.Class](#), [RTBComponent.CombineClass\(params string\[\]\)](#),
[RTBComponent SetProperty< TValue >\(ref TValue, TValue\)](#), [RTBComponent.StatefulAction\(Action\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSet\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRenderAsync\(bool\)](#),
[ComponentBase.InvokeAsync\(Action\)](#), [ComponentBase.InvokeAsync\(Func< Task >\)](#),
[ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

Behavior:

- When [IsBusy\(string?\)](#) returns true for [TrackId](#), the component renders [BusyContent](#) if provided; otherwise, a minimal fallback "Busy.." message.
- When not busy, the component renders [ChildContent](#).
- If [TrackId](#) is null or empty, the component reflects the "any key" busy state.

Lifecycle:

- Subscribes to `IBusyTracker.OnBusyChanged` in `OnInitialized()` and requests a re-render when the relevant key changes.
- On first render, forces a state check to ensure the initial UI reflects the current busy state.

Typical usage:

```
// In a Razor component:
<BusyIndicator TrackId="LoadData">
    <BusyContent>
        <div class="spinner">Loading...</div>
    </BusyContent>
    <ChildContent>
        <!-- Normal UI goes here -->
    </ChildContent>
</BusyIndicator>

@code {
    [Inject] IBusyTracker Busy { get; set; } = default!;

    private async Task LoadData()
    {
        using var _ = Busy.Track(nameof(LoadData)); // Matches TrackId="LoadData"
        await Task.Delay(1000);
    }
}
```

Properties

BusyContent

Content to render when the component is busy for the specified `TrackId`. If not supplied, a minimal fallback is rendered.

```
[Parameter]
public RenderFragment? BusyContent { get; set; }
```

Property Value

[RenderFragment](#)

ChildContent

Content to render when the component is not busy for the specified [TrackId](#).

```
[Parameter]  
public RenderFragment? ChildContent { get; set; }
```

Property Value

[RenderFragment](#) ↗

TrackId

Optional key that scopes the busy state for this component instance. When null or empty, the component reflects whether any key is busy.

```
[Parameter]  
public string? TrackId { get; set; }
```

Property Value

[string](#) ↗

Tracker

The busy state tracker used to determine whether to display [BusyContent](#) or [ChildContent](#). Provided via dependency injection.

```
[Inject]  
public IBusyTracker Tracker { get; set; }
```

Property Value

[IBusyTracker](#)

Methods

BuildRenderTree(RenderTreeBuilder)

Builds the component's UI based on the current busy state for [TrackId](#). Renders [ChildContent](#) when not busy; otherwise renders [BusyContent](#) or a fallback.

```
protected override void BuildRenderTree(RenderTreeBuilder builder)
```

Parameters

builder [RenderTreeBuilder](#)

The render tree builder.

Dispose()

Unsubscribes from tracker events and suppresses finalization.

```
public void Dispose()
```

OnAfterRender(bool)

On first render, ensures the component reflects the current busy state immediately.

```
protected override void OnAfterRender(bool firstRender)
```

Parameters

firstRender [bool](#)

True when this is the first time the component has rendered.

OnInitialized()

Subscribes to the tracker's [OnBusyChanged](#) event to update the UI when busy state transitions occur.

```
protected override void OnInitialized()
```

Class BusyTracker

Namespace: [RTB.Blazor.Services.BusyIndicator](#)

Assembly: RTB.Blazor.dll

Default implementation of [IBusyTracker](#) that tracks busy states by key using reference-counted scopes stored in concurrent dictionaries.

```
public class BusyTracker : IBusyTracker
```

Inheritance

[object](#) ← BusyTracker

Implements

[IBusyTracker](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

Thread-safe. Designed for UI scenarios (e.g., Blazor) to reflect background work state. Keys are case-sensitive. Disposing a tracking token decrements the count and may complete any awaiting task.

Properties

CurrentState

Returns the current mapping of keys to their active scope counts.

```
public IReadOnlyDictionary<string, int> CurrentState { get; }
```

Property Value

[IReadOnlyDictionary](#)<[string](#), [int](#)>

Remarks

This is a live, thread-safe view backed by a [ConcurrentDictionary<TKey, TValue>](#). Contents can change while iterating; prefer enumerating promptly.

IsAnyBusy

Indicates whether any tracked key is currently busy.

```
public bool IsAnyBusy { get; }
```

Property Value

[bool](#)

Tracks

The set of keys that are currently busy (i.e., have a positive outstanding count).

```
public string[] Tracks { get; }
```

Property Value

[string](#)[]

Methods

Await(string)

Returns a task that completes when the specified key is no longer busy.

```
public Task Await(string key)
```

Parameters

key [string](#)

The key to await completion for.

Returns

[Task](#) ↗

A task that completes when the key's busy count reaches zero. If the key is not currently busy, the task is already completed.

Remarks

Only a single waiter is maintained per key; subsequent calls overwrite previous waiters. Continuations run asynchronously.

IsBusy(string?)

Determines whether the specified key is currently busy.

```
public bool IsBusy(string? key = "")
```

Parameters

[key](#) [string](#) ↗

The key to check. When null or empty, this reports whether any key is busy ([IsAnyBusy](#)).

Returns

[bool](#) ↗

True if the specified key has an outstanding busy count greater than zero; or, when key is null/empty, true if any key is busy.

Track(string, Action?)

Begins tracking a busy scope for the specified key and returns a token that ends the scope on dispose.

```
public IDisposable Track(string key = "", Action? onDispose = null)
```

Parameters

key [string](#)

onDispose [Action](#)

Optional callback invoked after the busy scope is ended and the internal count is decremented/cleared.

Returns

[IDisposable](#)

An [IDisposable](#) token. Dispose to end the busy scope. Intended to be used with a using statement.

Examples

```
using var _ = busyTracker.Track(); // key becomes the caller's method name
await DoWorkAsync(); // disposing '_' marks this scope as complete
```

Events

OnBusyChanged

Raised when any busy state changes (work begins or ends).

```
public event Action<string?>? OnBusyChanged
```

Event Type

[Action](#) <[string](#)>

Remarks

The provided string argument is the key whose state changed. In Blazor components, marshal UI updates via `InvokeAsync(StateHasChanged)`.

Struct BusyTracker.BusyToken

Namespace: [RTB.Blazor.Services.BusyIndicator](#)

Assembly: RTB.Blazor.dll

Represents a tracked busy scope that automatically ends when disposed. Use with `using` to ensure the busy state is always released.

```
public readonly struct BusyTracker.BusyToken : IDisposable
```

Implements

[IDisposable](#)

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Remarks

Instances are created by [Track\(string, Action?\)](#). The provided `onDispose` action (if any) is invoked after the busy state is released.

Constructors

BusyToken(string, Action?)

Represents a tracked busy scope that automatically ends when disposed. Use with `using` to ensure the busy state is always released.

```
public BusyToken(string key, Action? onDispose = null)
```

Parameters

`key` [string](#)

`onDispose` [Action](#)

Remarks

Instances are created by [Track\(string, Action?\)](#). The provided onDispose action (if any) is invoked after the busy state is released.

Properties

Key

The key associated with this busy scope.

```
public string Key { get; }
```

Property Value

[string](#) ↗

Methods

Dispose()

Ends the busy scope and invokes the registered dispose callback (if any).

```
public void Dispose()
```

Interface IBusyTracker

Namespace: [RTB.Blazor.Services.BusyIndicator](#)

Assembly: RTB.Blazor.dll

Abstraction for tracking and reacting to "busy" states during asynchronous or long-running operations. Useful for driving UI concerns such as spinners, disabling actions, or detecting background work.

```
public interface IBusyTracker
```

Remarks

Thread-safe. Multiple independent scopes can be tracked per key. Keys are case-sensitive. Passing null or empty when supported indicates "any key".

Properties

IsAnyBusy

Indicates whether any tracked key is currently busy.

```
bool IsAnyBusy { get; }
```

Property Value

[bool](#)

Tracks

The set of keys that are currently busy (i.e., have a positive outstanding count).

```
string[] Tracks { get; }
```

Property Value

[string](#)[]

Methods

Await(string)

Returns a task that completes when the specified key is no longer busy.

Task `Await(string key)`

Parameters

`key string`

The key to await completion for.

Returns

[Task](#)

A task that completes when the key's busy count reaches zero. If the key is not currently busy, the task is already completed.

Remarks

Only a single waiter is maintained per key; subsequent calls overwrite previous waiters. Continuations run asynchronously.

IsBusy(string?)

Determines whether the specified key is currently busy.

`bool IsBusy(string? key = null)`

Parameters

`key string`

The key to check. When null or empty, this reports whether any key is busy ([IsAnyBusy](#)).

Returns

bool

True if the specified key has an outstanding busy count greater than zero; or, when key is null/empty, true if any key is busy.

Track(string, Action?)

Begins tracking a busy scope for the specified key and returns a token that ends the scope on dispose.

```
IDisposable Track(string method = "", Action? onDispose = null)
```

Parameters

method string

The key to track. Defaults to the caller's member name via [CallerMemberNameAttribute](#).

onDispose Action

Optional callback invoked after the busy scope is ended and the internal count is decremented/cleared.

Returns

IDisposable

An [IDisposable](#) token. Dispose to end the busy scope. Intended to be used with a using statement.

Examples

```
using var _ = busyTracker.Track(); // key becomes the caller's method name await DoWorkAsync(); // disposing '_' marks this scope as complete
```

Events

OnBusyChanged

Raised whenever the busy state changes for any key (work started or finished). The string argument contains the affected key; it may be null when callers check "any" state.

```
event Action<string?>? OnBusyChanged
```

Event Type

[Action](#) <[string](#)>

Remarks

In Blazor components, prefer invoking UI updates via `InvokeAsync(StateHasChanged)` from this callback to avoid threading issues. This event may fire frequently; keep handlers lightweight.

Namespace RTB.Blazor.Services.DataNavigation

Classes

[DataNavigationService](#)

Default implementation of [IDataNavigationService](#) that stores parameters in-memory and delegates navigation to [NavigationManager](#).

Interfaces

[IDataNavigationService](#)

Provides a simple, in-memory handoff of data between navigations in Blazor and a thin wrapper around [NavigationManager](#) for page/component navigation.

Class DataNavigationService

Namespace: [RTB.Blazor.Services.DataNavigation](#)

Assembly: RTB.Blazor.dll

Default implementation of [IDataNavigationService](#) that stores parameters in-memory and delegates navigation to [NavigationManager](#).

```
public class DataNavigationService : IDataNavigationService
```

Inheritance

[object](#) ← DataNavigationService

Implements

[IDataNavigationService](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Examples

Inject and use in a component:

```
@inject IDataNavigationService Nav

// Source component
Nav.NavigateTo("/details", parameter: new Dictionary<string, object?>
{
    ["itemId"] = 42
});

// Destination component
if (Nav.TryGetData<int>("itemId", out var id))
{
    // use id
}
```

Remarks

- Not thread-safe; intended for typical Blazor UI usage.

- The dictionary is in-memory only and not persisted across app restarts or different circuits/sessions.

Constructors

DataNavigationService(NavigationManager)

Default implementation of [IDataNavigationService](#) that stores parameters in-memory and delegates navigation to [NavigationManager](#).

```
public DataNavigationService(NavigationManager navigationManager)
```

Parameters

`navigationManager` [NavigationManager](#)

Examples

Inject and use in a component:

```
@inject IDataNavigationService Nav

// Source component
Nav.NavigateTo("/details", parameter: new Dictionary<string, object?>
{
    ["itemId"] = 42
});

// Destination component
if (Nav.TryGetData<int>("itemId", out var id))
{
    // use id
}
```

Remarks

- Not thread-safe; intended for typical Blazor UI usage.
- The dictionary is in-memory only and not persisted across app restarts or different circuits/sessions.

Methods

Clear(string?)

Clears stored values.

```
public void Clear(string? prefix = null)
```

Parameters

prefix [string](#)

If null, clears all stored data. If specified, removes keys that start with the prefix using [StartsWith\(string\)](#).
↗ default semantics (culture-sensitive, case-sensitive).

HasData(string)

Determines whether a value exists for the given **key**.

```
public bool HasData(string key)
```

Parameters

key [string](#)

The key to check.

Returns

[bool](#)

True if the key exists; otherwise false.

NavigateTo(string, bool, bool, IDictionary<string, object?>?)

Navigates to the specified **uri** and optionally attaches a set of in-memory parameters that can be retrieved on the destination via [TryGetData<T>\(string, out T?, bool\)](#).

```
public void NavigateTo(string uri, bool forceLoad = false, bool replace = false,  
IDictionary<string, object?>? parameter = null)
```

Parameters

`uri` [string](#)

The target URI (relative or absolute) to navigate to.

`forceLoad` [bool](#)

If true, bypasses client-side routing and forces the browser to load the new page from the server.

`replace` [bool](#)

If true, replaces the current entry in the history stack (does not create a new history entry).

`parameter` [IDictionary](#)<[string](#), [object](#)>

Optional key-value pairs to store in-memory prior to navigation. Keys are case-sensitive and are not included in the URL. Values are available on the destination via [TryGetData<T>\(string, out T?, bool\)](#).

TryGetData<T>(string, out T?, bool)

Attempts to retrieve a previously stored value for `key` and cast it to `T`.

```
public bool TryGetData<T>(string key, out T? value, bool remove = true)
```

Parameters

`key` [string](#)

The key used when storing the value.

`value` [T](#)

When this method returns true, contains the value cast to `T`. Otherwise, set to default.

`remove` [bool](#)

If true (default), removes the value from the store upon successful retrieval. Set false to keep it for subsequent reads.

Returns

[bool](#)

True if a value exists for `key` and is of type `T`; otherwise false.

Type Parameters

`T`

Expected type of the stored value.

Interface IDataNavigationService

Namespace: [RTB.Blazor.Services.DataNavigation](#)

Assembly: RTB.Blazor.dll

Provides a simple, in-memory handoff of data between navigations in Blazor and a thin wrapper around [NavigationManager](#) for page/component navigation.

```
public interface IDataNavigationService
```

Remarks

- Keys and values supplied to [NavigateTo\(string, bool, bool, IDictionary<string, object?>?\)](#) are stored in-memory and are not encoded into the URL. Retrieve them on the destination via [TryGetData<T>\(string, out T?, bool\)](#).
- The service is not thread-safe; it relies on Blazor's typical single-threaded UI usage.
- Lifetime of the stored data matches the service's DI lifetime (commonly **Scoped** in Blazor).

Methods

Clear(string?)

Clears stored values.

```
void Clear(string? prefix = null)
```

Parameters

prefix [string](#)

If null, clears all stored data. If specified, removes keys that start with the prefix using [StartsWith\(string\)](#) default semantics (culture-sensitive, case-sensitive).

HasData(string)

Determines whether a value exists for the given **key**.

```
bool HasData(string key)
```

Parameters

key [string](#)

The key to check.

Returns

[bool](#)

True if the key exists; otherwise false.

NavigateTo(string, bool, bool, IDictionary<string, object?>?)

Navigates to the specified **uri** and optionally attaches a set of in-memory parameters that can be retrieved on the destination via [TryGetData<T>\(string, out T?, bool\)](#).

```
void NavigateTo(string uri, bool forceLoad = false, bool replace = false,  
IDictionary<string, object?>? parameter = null)
```

Parameters

uri [string](#)

The target URI (relative or absolute) to navigate to.

forceLoad [bool](#)

If true, bypasses client-side routing and forces the browser to load the new page from the server.

replace [bool](#)

If true, replaces the current entry in the history stack (does not create a new history entry).

parameter [IDictionary<string, object>](#)

Optional key-value pairs to store in-memory prior to navigation. Keys are case-sensitive and are not included in the URL. Values are available on the destination via [TryGetData<T>\(string, out T?, bool\)](#).

TryGetData<T>(string, out T?, bool)

Attempts to retrieve a previously stored value for **key** and cast it to **T**.

```
bool TryGetData<T>(string key, out T? value, bool remove = true)
```

Parameters

key [string](#)

The key used when storing the value.

value [T](#)

When this method returns true, contains the value cast to **T**. Otherwise, set to default.

remove [bool](#)

If true (default), removes the value from the store upon successful retrieval. Set false to keep it for subsequent reads.

Returns

[bool](#)

True if a value exists for **key** and is of type **T**; otherwise false.

Type Parameters

T

Expected type of the stored value.

Namespace RTB.Blazor.Services.Dialog

Classes

[DialogBase](#)

Base class for dialog host components.

[DialogResult](#)

Immutable result returned by a dialog when it closes.

[DialogService](#)

Default [IDialogService](#) implementation using [RenderFragment](#) composition.

Interfaces

[IDialogReference](#)

A reference to an active dialog instance. Allows awaiting completion and closing it programmatically.

[IDialogService](#)

Service surface for presenting dialogs.

Enums

[DialogResultKind](#)

Represents the outcome of a dialog interaction.

Class DialogBase

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

Base class for dialog host components.

```
public abstract class DialogBase : RTBComponent, IComponent, IHandleEvent,  
IHandleAfterRender, IDialogReference
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBComponent](#) ← DialogBase

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IDialogReference](#)

Inherited Members

[RTBComponent.Id](#), [RTBComponent.Class](#), [RTBComponent.CombineClass\(params string\[\]\)](#),
[RTBComponent SetProperty< TValue >\(ref TValue, TValue\)](#), [RTBComponent.StatefulAction\(Action\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitialized\(\)](#),
[ComponentBase.OnInitializedAsync\(\)](#), [ComponentBase.OnParametersSet\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func< Task >\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

```
// Showing a dialog var result = await Dialogs.ShowAsync<MyDialog>(new() { ["Title"] = "Edit item" }); if  
(result.Kind is DialogResultKind.Ok) { /* handle success */ }
```

```
// Inside dialog content component [CascadingParameter] public IDialogReference? Dialog { get; set; }
```

```
void Save(object model) => Dialog?.Close(DialogResult.Ok(model)); void Cancel() => Dialog?.Close(); //  
Equivalent to Cancel(null)
```

Remarks

Provides a common implementation of [IDialogReference](#) using an internal [TaskCompletionSource<TResult>](#) to signal completion when the dialog is closed.

Derived components can call one of the `Close` overloads to complete `Result`. Calls to `Close` are idempotent — only the first call wins; subsequent calls are ignored.

Typically used in conjunction with [IDialogService](#) which creates and cascades an [IDialogReference](#) to the dialog content.

Properties

Backdrop

Whether an overlay/backdrop is rendered behind the dialog.

```
[Parameter]
public bool Backdrop { get; set; }
```

Property Value

[bool](#)

ChildContent

Arbitrary content rendered inside the dialog host (typically the dialog body component).

```
[Parameter]
public RenderFragment? ChildContent { get; set; }
```

Property Value

[RenderFragment](#)

Dialog

The reference to the active dialog instance, cascaded to child content.

```
[CascadingParameter]  
public IDialogReference? Dialog { get; set; }
```

Property Value

[IDialogReference](#)

DialogService

The ambient dialog service, cascaded from the application root.

```
[CascadingParameter]  
public IDialogService? DialogService { get; set; }
```

Property Value

[IDialogService](#)

Open

Indicates whether the dialog is still open ([true](#)) or has completed ([false](#)).

```
public bool Open { get; }
```

Property Value

[bool](#) ↗

Result

A task that completes when the dialog is closed, yielding the [DialogResult](#).

```
public Task<DialogResult> Result { get; }
```

Property Value

Methods

Close()

Closes the dialog with [Cancel](#).

```
public void Close()
```

Remarks

This method is idempotent. Only the first call completes [Result](#).

Close(DialogResult)

Closes the dialog with an explicit [result](#).

```
public void Close(DialogResult result)
```

Parameters

[result](#) [DialogResult](#)

The result to return to the caller awaiting the dialog.

Remarks

This method is idempotent. Only the first call completes [Result](#).

Close(object?)

Closes the dialog with [Ok](#), optionally returning [data](#).

```
public void Close(object? data)
```

Parameters

data [object](#)

Optional payload to attach to the OK result.

Remarks

This method is idempotent. Only the first call completes [Result](#).

Class DialogResult

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

Immutable result returned by a dialog when it closes.

```
public sealed record DialogResult : IEquatable<DialogResult>
```

Inheritance

[object](#) ← DialogResult

Implements

[IEquatable](#)<[DialogResult](#)>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Examples

In a dialog component:

```
[CascadingParameter] public IDialogReference? Dialog { get; set; }
```

```
void Save() => Dialog?.Close(DialogResult.Ok(new { Name, Email }));
```

```
void Dismiss() => Dialog?.Close(); // Equivalent to Cancel()
```

Remarks

Use [Ok\(object?\)](#) or [Cancel\(object?\)](#) to create a result with a conventional outcome.

Constructors

DialogResult(DialogResultKind, object?)

Immutable result returned by a dialog when it closes.

```
public DialogResult(DialogResultKind Kind, object? Data = null)
```

Parameters

Kind [DialogResultKind](#)

The outcome of the dialog.

Data [object](#)

Optional payload returned by the dialog (e.g., form data).

Examples

In a dialog component:

```
[CascadingParameter] public IDialogReference? Dialog { get; set; }
```

```
void Save() => Dialog?.Close(DialogResult.Ok(new { Name, Email }));
```

```
void Dismiss() => Dialog?.Close(); // Equivalent to Cancel()
```

Remarks

Use [Ok\(object?\)](#) or [Cancel\(object?\)](#) to create a result with a conventional outcome.

Properties

Data

Optional payload returned by the dialog (e.g., form data).

```
public object? Data { get; init; }
```

Property Value

[object](#)

Kind

The outcome of the dialog.

```
public DialogResultKind Kind { get; init; }
```

Property Value

[DialogResultKind](#)

Methods

Cancel(object?)

Creates a canceled result with an optional payload (e.g., reason).

```
public static DialogResult Cancel(object? data = null)
```

Parameters

data [object](#)

Returns

[DialogResult](#)

Ok(object?)

Creates a successful result with an optional payload.

```
public static DialogResult Ok(object? data = null)
```

Parameters

data [object](#)

Returns

[DialogResult](#)

Enum DialogResultKind

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

Represents the outcome of a dialog interaction.

```
public enum DialogResultKind
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Cancel = 1

The dialog was canceled (e.g., secondary action, escape, close button).

None = 2

The dialog closed without an explicit result. Typically used as a sentinel.

Ok = 0

The dialog completed successfully (e.g., primary action).

Class DialogService

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

Default [IDialogService](#) implementation using [RenderFragment](#) composition.

```
public class DialogService : IDialogService
```

Inheritance

[object](#) ← DialogService

Implements

[IDialogService](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- Raises [OnShow](#) with a [RenderFragment](#) to render a dialog host ([DialogHost](#) or [AlertHost](#)).
- Completes the awaiting task when the host signals completion through [Result](#).
- Invokes [OnClose](#) when the dialog completes.

Ensure a provider component (e.g., [DialogProvider](#)) subscribes to [OnShow](#) and renders the supplied fragment.

Methods

Alert<TDIALOG>(Dictionary<string, object?>?)

Presents a transient alert using the specified [TDIALOG](#) content.

```
public void Alert<TDIALOG>(Dictionary<string, object?>? parameters = null) where TDIALOG : IComponent
```

Parameters

parameters [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the alert content component.

Type Parameters

TDIALOG

The alert content component type.

Remarks

- Renders an [AlertHost](#) and nests **TDIALOG** inside it.
- Does not return a task; suitable for ephemeral notifications.
- When the alert completes, [OnClose](#) is invoked.

ShowAsync(Type, Dictionary<string, object?>?, Dictionary<string, object?>?)

Shows a dialog rendering the specified component [dialogType](#).

```
public Task<DialogResult> ShowAsync(Type dialogType, Dictionary<string, object?>? parameters = null, Dictionary<string, object?>? dialogParameters = null)
```

Parameters

dialogType [Type](#)

The dialog content component type (must implement [IComponent](#)).

parameters [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the dialog content component.

dialogParameters [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the dialog host (e.g., size, modality).

Returns

[Task](#)<[DialogResult](#)>

A task that completes with the [DialogResult](#) when the dialog closes.

Exceptions

[ArgumentException](#)

Thrown when `dialogType` does not implement [IComponent](#).

ShowAsync<TDialog>(Dictionary<string, object?>?, Dictionary<string, object?>?)

Shows a dialog rendering the specified `TDialog` component.

```
public Task<DialogResult> ShowAsync<TDialog>(Dictionary<string, object?>? parameters = null, Dictionary<string, object?>? dialogParameters = null) where TDialog : IComponent
```

Parameters

`parameters` [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the dialog content component.

`dialogParameters` [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the dialog host (e.g., size, modality).

Returns

[Task](#)<[DialogResult](#)>

A task that completes with the [DialogResult](#) when the dialog closes.

Type Parameters

`TDialog`

The dialog content component type.

Events

OnClose

Raised when a dialog has closed (after the result is set).

```
public event Action? OnClose
```

Event Type

[Action](#)

OnShow

Raised when a dialog should be rendered. The provided fragment renders the host and content.

```
public event Action<RenderFragment>? OnShow
```

Event Type

[Action](#) <[RenderFragment](#)>

See Also

[IDialogService](#)

Interface IDialogReference

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

A reference to an active dialog instance. Allows awaiting completion and closing it programmatically.

```
public interface IDialogReference
```

Properties

Result

A task that completes when the dialog is closed, yielding the [DialogResult](#).

```
Task<DialogResult> Result { get; }
```

Property Value

[Task](#) <[DialogResult](#)>

Methods

Close()

Closes the dialog, returning [Cancel](#).

```
void Close()
```

Close(DialogResult)

Closes the dialog with an explicit [result](#).

```
void Close(DialogResult result)
```

Parameters

result [DialogResult](#)

The result to return to the caller awaiting the dialog.

Close([object?](#))

Closes the dialog, returning [Ok](#) and the provided **data**.

void [Close\(\[object?\]\(#\) **data**\)](#)

Parameters

data [object](#)

Optional payload to attach to an OK result.

Interface IDialogService

Namespace: [RTB.Blazor.Services.Dialog](#)

Assembly: RTB.Blazor.dll

Service surface for presenting dialogs.

```
public interface IDialogService
```

Remarks

Registration (Program.cs):

```
builder.Services.AddScoped<IDialogService, DialogService>();
```

App.razor:

```
@using RTB.Blazor.Services.Dialog
@inject IDialogService DialogService

<CascadingValue Value="DialogService">
    <Router AppAssembly="typeof(App).Assembly">
        <Found Context="routeData">
            <RouteView RouteData="routeData" DefaultLayout="typeof(MainLayout)" />
        </Found>
        <NotFound>...</NotFound>
    </Router>
    <DialogProvider />
</CascadingValue>
```

Usage in a component:

```
@inject IDialogService Dialogs

var result = await Dialogs.ShowAsync<MyDialog>(new()
{
    ["Title"] = "Edit item",
    ["Model"] = item
});
if (result.Kind is DialogResultKind.Ok) { /* handle success */ }
```

Methods

Alert<TDialog>(Dictionary<string, object?>?)

Presents a transient alert using the specified [TDialog](#) content.

```
void Alert<TDialog>(Dictionary<string, object?>? parameters = null) where TDialog : IComponent
```

Parameters

[parameters](#) [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the alert content component.

Type Parameters

[TDialog](#)

Remarks

This is fire-and-forget and does not return a [Task](#). Use for notifications/toasts.

ShowAsync(Type, Dictionary<string, object?>?, Dictionary<string, object?>?)

Shows a dialog rendering the specified component [dialogType](#).

```
Task<DialogResult> ShowAsync(Type dialogType, Dictionary<string, object?>? contentParameters = null, Dictionary<string, object?>? dialogParameters = null)
```

Parameters

[dialogType](#) [Type](#)

The dialog content component type (must implement [IComponent](#)).

[contentParameters](#) [Dictionary](#)<[string](#), [object](#)>

Attributes passed to the dialog content component.

dialogParameters [Dictionary<string, object>](#)

Attributes passed to the dialog host (e.g., size, modality).

Returns

[Task<DialogResult>](#)

A task that completes with the [DialogResult](#) when the dialog closes.

ShowAsync<TDialog>(Dictionary<string, object?>?, Dictionary<string, object?>?)

Shows a dialog rendering the specified [TDialog](#) component.

```
Task<DialogResult> ShowAsync<TDialog>(Dictionary<string, object?>? contentParameters = null, Dictionary<string, object?>? dialogParameters = null) where TDialog : IComponent
```

Parameters

contentParameters [Dictionary<string, object>](#)

Attributes passed to the dialog content component.

dialogParameters [Dictionary<string, object>](#)

Attributes passed to the dialog host (e.g., size, modality).

Returns

[Task<DialogResult>](#)

A task that completes with the [DialogResult](#) when the dialog closes.

Type Parameters

TDialog

The dialog content component type.

Namespace RTB.Blazor.Services.DragDrop

Classes

[DragDropService](#)

Default implementation of [IDragDropService](#) for Blazor components.

Interfaces

[IDragDropService](#)

Provides a lightweight, in-memory mechanism to pass a single drag payload between draggable and droppable components in a Blazor application.

Class DragDropService

Namespace: [RTB.Blazor.Services.DragDrop](#)

Assembly: RTB.Blazor.dll

Default implementation of [IDragDropService](#) for Blazor components.

```
public class DragDropService : IDragDropService
```

Inheritance

[object](#) ← DragDropService

Implements

[IDragDropService](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

Register as a Scoped service in DI for Blazor: services.AddScoped<IDragDropService, DragDropService>();

Properties

DraggedItemData

Gets the raw object currently stored as the drag payload.

```
public object? DraggedItemData { get; }
```

Property Value

[object](#)

Remarks

This value is set by [StartDrag<TObject>\(TObject?\)](#) and cleared by [GetDataOnDrop<TObject>\(\)](#). Prefer using the typed APIs instead of accessing this property directly.

Methods

GetDataOnDrop<TObject>()

Retrieves the drag payload as the requested type and clears the stored payload.

```
public TObject? GetDataOnDrop<TObject>()
```

Returns

TObject

The stored payload cast to **TObject**.

Type Parameters

TObject

The expected type for the payload.

Remarks

After calling this method, the internal payload is reset to its default (null).

Exceptions

[InvalidCastException](#)

Thrown when no payload is present or the stored payload cannot be cast to **TObject**.

StartDrag<TObject>(TObject?)

Starts a drag operation by storing the provided payload.

```
public void StartDrag<TObject>(TObject? itemData)
```

Parameters

itemData TObject

The payload to store. May be null to explicitly indicate no data.

Type Parameters

TObject

The compile-time type of the payload.

Remarks

This method overwrites any previously stored payload.

Interface IDragDropService

Namespace: [RTB.Blazor.Services.DragDrop](#)

Assembly: RTB.Blazor.dll

Provides a lightweight, in-memory mechanism to pass a single drag payload between draggable and droppable components in a Blazor application.

```
public interface IDragDropService
```

Remarks

Usage:

- Call [StartDrag< TObject >\(TObject? \)](#) when a drag operation begins.
- Call [GetDataOnDrop< TObject >\(\)](#) on the drop target to retrieve and clear the payload. Notes:
 - The stored payload is cleared on every successful call to [GetDataOnDrop< TObject >\(\)](#).
 - Intended DI lifetime is Scoped in Blazor (one instance per circuit/user session).
 - Not thread-safe. Access from the Blazor UI synchronization context.

Properties

DraggedItemData

Gets the raw object currently stored as the drag payload.

```
object? DraggedItemData { get; }
```

Property Value

[object](#) ↗

Remarks

This value is set by [StartDrag< TObject >\(TObject? \)](#) and cleared by [GetDataOnDrop< TObject >\(\)](#). Prefer using the typed APIs instead of accessing this property directly.

Methods

GetDataOnDrop<TObject>()

Retrieves the drag payload as the requested type and clears the stored payload.

```
TObject? GetDataOnDrop<TObject>()
```

Returns

TObject

The stored payload cast to **TObject**.

Type Parameters

TObject

The expected type for the payload.

Remarks

After calling this method, the internal payload is reset to its default (null).

Exceptions

[InvalidCastException](#)

Thrown when no payload is present or the stored payload cannot be cast to **TObject**.

StartDrag<TObject>(TObject?)

Starts a drag operation by storing the provided payload.

```
void StartDrag<TObject>(TObject? itemData)
```

Parameters

itemData TObject

The payload to store. May be null to explicitly indicate no data.

Type Parameters

TObject

The compile-time type of the payload.

Remarks

This method overwrites any previously stored payload.

Namespace RTB.Blazor.Services.Input

Classes

[InputService](#)

Default implementation of [IInputService](#) using [IJSRuntime](#) to receive keyboard events from the browser.

Interfaces

[IInputService](#)

Provides a Blazor-friendly keyboard input subscription service that bridges DOM keyboard events from JavaScript to .NET callbacks.

Interface IInputService

Namespace: [RTB.Blazor.Services.Input](#)

Assembly: RTB.Blazor.dll

Provides a Blazor-friendly keyboard input subscription service that bridges DOM keyboard events from JavaScript to .NET callbacks.

```
public interface IInputService : IDisposable
```

Inherited Members

[IAsyncDisposable.DisposeAsync\(\)](#)

Remarks

- Call [InitializeAsync\(\)](#) once to attach JS listeners (typically at app startup or the first time a page needing input renders).
- Use [RegisterKeyHandler\(string, Action<bool>\)](#) to subscribe to specific keys via `KeyboardEvent.key` values (e.g., "Shift", "Control", "a").
- Callbacks receive `true` on key down and `false` on key up.
- Use [UnregisterKeyHandler\(string, Action<bool>\)](#) to remove subscriptions.
- Dispose the service (or its scope) to detach JS listeners. Thread-safety: Handler lists are protected with locks on the per-key list; enumeration uses a snapshot to avoid concurrent modification.

Methods

InitializeAsync()

Initializes the JavaScript interop by attaching global keyboard listeners once.

```
Task InitializeAsync()
```

Returns

[Task](#)

A task that completes when the JS listeners are attached.

Remarks

This calls the JS function `inputService.register(dotNetObjRef)`. Ensure a corresponding JS implementation exists and is loaded. Safe to call multiple times; only the first call needs to wire up JS.

RegisterKeyHandler(string, Action<bool>)

Subscribe a callback to a specific key as reported by `KeyboardEvent.key`.

```
void RegisterKeyHandler(string key, Action<bool> callback)
```

Parameters

`key` [string](#)

The key identifier string (e.g., "Shift", "Control", "Alt", "ArrowUp", "a", "A"). Use the exact value produced by `KeyboardEvent.key`.

`callback` [Action](#)<[bool](#)>

The callback invoked with `true` on key down and `false` on key up.

Remarks

Multiple handlers can be registered for the same key. Handlers are invoked in registration order.

UnregisterKeyHandler(string, Action<bool>)

Unsubscribe a previously registered callback for a specific key.

```
void UnregisterKeyHandler(string key, Action<bool> callback)
```

Parameters

`key` [string](#)

The same key used during registration.

`callback` [Action](#)<[bool](#)>

The callback instance to remove.

Class InputService

Namespace: [RTB.Blazor.Services.Input](#)

Assembly: RTB.Blazor.dll

Default implementation of [IInputService](#) using [IJSRuntime](#) to receive keyboard events from the browser.

```
public class InputService : IInputService, IAsyncDisposable
```

Inheritance

[object](#) ← InputService

Implements

[IInputService](#), [IAsyncDisposable](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

Expects the following JS functions to exist:

- `inputService.register(dotNetObjRef)`: Attaches keydown/keyup listeners and forwards events via [OnKeyDown\(string\)](#) and [OnKeyUp\(string\)](#).
- `inputService.unregister()`: Detaches listeners.

Constructors

InputService(IJSRuntime)

Default implementation of [IInputService](#) using [IJSRuntime](#) to receive keyboard events from the browser.

```
public InputService(IJSRuntime jsRuntime)
```

Parameters

`jsRuntime` [IJSRuntime](#)

Remarks

Expects the following JS functions to exist:

- `inputService.register(dotNetObjRef)`: Attaches keydown/keyup listeners and forwards events via [OnKeyDown\(string\)](#) and [OnKeyUp\(string\)](#).
- `inputService.unregister()`: Detaches listeners.

Methods

DisposeAsync()

Detaches JavaScript listeners and releases resources.

```
public ValueTask DisposeAsync()
```

Returns

[ValueTask](#)

Remarks

Swallows errors if the JS runtime is already disposed (e.g., during app shutdown).

InitializeAsync()

Initializes the JavaScript interop by attaching global keyboard listeners once.

```
public Task InitializeAsync()
```

Returns

[Task](#)

A task that completes when the JS listeners are attached.

Remarks

This calls the JS function `inputService.register(dotNetObjRef)`. Ensure a corresponding JS implementation exists and is loaded. Safe to call multiple times; only the first call needs to wire up JS.

OnKeyDown(string)

Invoked from JavaScript on keydown with the `KeyboardEvent.key` value.

```
[JSInvokable]  
public void OnKeyDown(string key)
```

Parameters

`key` [string](#)

The key identifier (exact string from the browser event).

OnKeyUp(string)

Invoked from JavaScript on keyup with the `KeyboardEvent.key` value.

```
[JSInvokable]  
public void OnKeyUp(string key)
```

Parameters

`key` [string](#)

The key identifier (exact string from the browser event).

RegisterKeyHandler(string, Action<bool>)

Subscribe a callback to a specific key as reported by `KeyboardEvent.key`.

```
public void RegisterKeyHandler(string key, Action<bool> callback)
```

Parameters

key [string](#)

The key identifier string (e.g., "Shift", "Control", "Alt", "ArrowUp", "a", "A"). Use the exact value produced by `KeyboardEvent.key`.

callback [Action](#)<[bool](#)>

The callback invoked with `true` on key down and `false` on key up.

Remarks

Multiple handlers can be registered for the same key. Handlers are invoked in registration order.

UnregisterKeyHandler(string, Action<bool>)

Unsubscribe a previously registered callback for a specific key.

```
public void UnregisterKeyHandler(string key, Action<bool> callback)
```

Parameters

key [string](#)

The same key used during registration.

callback [Action](#)<[bool](#)>

The callback instance to remove.

Namespace RTB.Blazor.Services.Theme

Classes

[RTBThemeService<TThemeBase>](#)

Provides theme discovery, selection, and persistence for Blazor using JS interop.

[ThemeAttribute](#)

Attribute to mark a class as a theme.

Interfaces

[ITheme](#)

Defines a contract for a UI theme within the Blazor UI layer.

[IThemeServiceFactory](#)

Factory interface to access the current theme service instance without a generic type.

[IThemeService<TTheme>](#)

Theme provider interface.

Interface ITheme

Namespace: [RTB.Blazor.Services.Theme](#)

Assembly: RTB.Blazor.dll

Defines a contract for a UI theme within the Blazor UI layer.

```
public interface ITheme
```

Examples

Example: public sealed class DarkTheme : ITheme { public string Name => "Dark"; }

Remarks

Implementations encapsulate theme-specific metadata and resources (e.g., colors, typography, component styles).

Properties

Name

Gets the unique, human-readable theme name used for selection and display.

```
string Name { get; }
```

Property Value

[string](#)

A short identifier such as "Light" or "Dark".

Interface IThemeServiceFactory

Namespace: [RTB.Blazor.Services.Theme](#)

Assembly: RTB.Blazor.dll

Factory interface to access the current theme service instance without a generic type.

```
public interface IThemeServiceFactory
```

Remarks

Useful in scenarios where the concrete `TTheme` is not known at compile time, such as dynamic composition or cross-cutting infrastructure.

Methods

GetCurrent()

Gets the current theme service instance as an [`object`](#).

```
object GetCurrent()
```

Returns

[`object`](#)

The current theme service instance; callers may cast to the expected generic interface.

Interface IThemeService<TTheme>

Namespace: [RTB.Blazor.Services.Theme](#)

Assembly: RTB.Blazor.dll

Theme provider interface.

```
public interface IThemeService<TTheme> where TTheme : ITheme
```

Type Parameters

TTheme

Interface type that represents a Theme and implements [ITheme](#).

Examples

Example usage in a Blazor component:

```
@inject IThemeService<MyTheme> ThemeService

<select @onchange="OnChange">
    @foreach (var theme in ThemeService.Themes)
    {
        <option selected="@(theme == ThemeService.Current)" value="@theme.Name">@theme.Name</option>
    }
</select>

@code {
    protected override void OnInitialized()
    {
        ThemeService.OnThemeChanged += StateHasChanged;
    }

    private async Task OnChange(ChangeEventArgs e)
    {
        var next = ThemeService.Themes.First(t => t.Name == (string)e.Value!);
        await ThemeService.SetThemeAsync(next);
    }

    public void Dispose()
```

```
{  
    ThemeService.OnThemeChanged -= StateHasChanged;  
}  
}
```

Remarks

Typical implementations:

- Expose a list of available themes via [Themes](#).
- Track the currently selected theme via [Current](#).
- Optionally persist the selection (e.g., to local storage in Blazor, preferences in MAUI).
- Raise [OnThemeChanged](#) after the theme is changed.

Threading:

- [OnThemeChanged](#) may be raised on a non-UI thread; consumers should marshal to the UI thread if required.

Properties

Current

Gets the currently active theme.

```
TTheme Current { get; }
```

Property Value

TTheme

Default

Gets the default theme used when no persisted preference is available.

```
TTheme Default { get; }
```

Property Value

Themes

Gets the list of available themes.

```
IList<TTheme> Themes { get; }
```

Property Value

[IList](#)<TTheme>

Methods

SetThemeAsync(TTheme)

Sets the current theme.

```
ValueTask SetThemeAsync(TTheme theme)
```

Parameters

theme TTheme

The theme to apply.

Returns

[ValueTask](#)

A task-like value representing the asynchronous operation. Implementations may persist the selection and apply side effects (e.g., updating CSS variables) before completing.

Events

OnThemeChanged

Occurs after the current theme has changed.

event Action? OnThemeChanged

Event Type

[Action](#)

Remarks

Implementations should invoke this event after updating [Current](#). Subscribers should assume it may be raised on a non-UI thread.

Class RTBThemeService<TThemeBase>

Namespace: [RTB.Blazor.Services.Theme](#)

Assembly: RTB.Blazor.dll

Provides theme discovery, selection, and persistence for Blazor using JS interop.

```
public class RTBThemeService<TThemeBase> : IThemeService<TThemeBase> where TThemeBase : ITheme
```

Type Parameters

TThemeBase

Base theme interface that must implement [ITheme](#). All concrete types assignable to this type with a public parameterless constructor are discovered via reflection as available themes.

Inheritance

[object](#) ← RTBThemeService<TThemeBase>

Implements

[IThemeService](#)<TThemeBase>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

RTBThemeService(IJSRuntime)

Provides theme discovery, selection, and persistence for Blazor using JS interop.

```
public RTBThemeService(IJSRuntime jsRuntime)
```

Parameters

jsRuntime [IJSRuntime](#)

JS runtime used to persist the selected theme to browser localStorage (key: "rtbtheme").

Properties

Current

Gets the current theme. If not explicitly set, this returns [Default](#).

```
public TThemeBase Current { get; }
```

Property Value

TThemeBase

Default

Gets the default theme. Determined by the presence of [ThemeAttribute](#) with `IsDefault == true`, otherwise falls back to the first discovered theme.

```
public TThemeBase Default { get; }
```

Property Value

TThemeBase

Themes

Gets all available theme instances discovered via reflection. A theme is considered available if it:

- Is a non-abstract class
- Is assignable to [TThemeBase](#)
- Has a public parameterless constructor Each matching type is instantiated once when this property is evaluated.

```
public IList<TThemeBase> Themes { get; }
```

Property Value

[IList](#) <TThemeBase>

Methods

SetThemeAsync(TThemeBase)

Sets the current theme and persists the selection to browser localStorage using JS interop.

```
public ValueTask SetThemeAsync(TThemeBase theme)
```

Parameters

theme TThemeBase

The theme instance to set as current.

Returns

[ValueTask](#)

A task representing the asynchronous operation.

Events

OnThemeChanged

Raised after the current theme has changed via [SetThemeAsync\(TThemeBase\)](#).

```
public event Action? OnThemeChanged
```

Event Type

[Action](#)

Class ThemeAttribute

Namespace: [RTB.Blazor.Services.Theme](#)

Assembly: RTB.Blazor.dll

Attribute to mark a class as a theme.

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = true)]
public class ThemeAttribute : Attribute
```

Inheritance

[object](#) ← [Attribute](#) ← ThemeAttribute

Inherited Members

[Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

IsDefault

If true, this theme will be used as the default theme.

```
public bool IsDefault { get; set; }
```

Property Value

[bool](#)

Namespace RTB.Blazor.Styled.Components

Classes

[Animation](#)

Describes animation-related CSS longhands and establishes an @keyframes scope for a single animation.

[AnimationStyleExtensions](#)

Extension helpers that map [Animation](#) parameters to CSS `animation-*` longhand declarations.

[Background](#)

Contributes a CSS background-color declaration to the current style scope.

[BackgroundExtensions](#)

Extension helpers for adding background-related CSS to a [StyleBuilder](#).

[Border](#)

Contributes CSS `border` and `border-radius` declarations to the cascading [StyleBuilder](#).

[BorderStyleExtensions](#)

Extension methods on [StyleBuilder](#) for composing border and radius declarations.

[Color](#)

Contributes a CSS `color` declaration to the current [StyleBuilder](#) scope.

[ColorExtensions](#)

[StyleBuilder](#) extensions for the CSS `color` property.

[Flex](#)

Contributes CSS declarations for a flex container (`display: flex`) to the current [StyleBuilder](#) scope.

[FlexExtensions](#)

Extension helpers for composing flex container styles fluently.

[Grid](#)

Contributes CSS for a grid container to the current [StyleBuilder](#) scope. When placed inside a style scope, this component sets:

- `display: grid`
- `grid-template-columns`
- `grid-template-rows`
- `gap` (optional)
- `place-items` (optional)

[GridExtensions](#)

Extension methods for configuring grid-related CSS on a [StyleBuilder](#).

[GridPlacement](#)

Contributes CSS Grid placement declarations to a [StyleBuilder](#).

[GridPlacementExtensions](#)

Extension methods for composing CSS Grid placement with a [StyleBuilder](#).

[Keyframe](#)

Represents a single keyframe entry inside an ancestor animation's `@keyframes` block.

[Margin](#)

Contributes CSS margin declarations to the current [StyleBuilder](#) scope.

[MarginExtensions](#)

Fluent extensions for applying CSS margin declarations using [StyleBuilder](#).

[Media](#)

Contributes a `@media` group to the ambient [StyleBuilder](#) using a provided [BreakPoint](#).

[Opacity](#)

Contributes the CSS `opacity` declaration to the current [StyleBuilder](#).

[OpacityExtensions](#)

Extension helpers for configuring `opacity` on a [StyleBuilder](#).

[Other](#)

A generic style component that writes an arbitrary CSS declaration to the current style scope.

[OtherExtensions](#)

Extension helpers for emitting arbitrary CSS declarations using [StyleBuilder](#).

[Overflow](#)

A style contributor that sets CSS overflow properties. Behavior:

- When all parameters (`X`, `Y`, `Value`) are null, this emits 'overflow: auto'.
- Otherwise, it emits any provided axis-specific values (`overflow-x`, `overflow-y`) and/or the shorthand (`overflow`). Note: If both axis-specific values and the shorthand are supplied, the emitted order is: `overflow-x`, `overflow-y`, `overflow` (last). In CSS, the shorthand can reset axis-specific values. Prefer setting either axis-specific values or the shorthand to avoid unintended overrides.

[OverflowExtensions](#)

Extension methods on [StyleBuilder](#) for setting overflow-related CSS.

[Padding](#)

Contributes CSS padding declarations to a cascading [StyleBuilder](#). Supports:

- A single value for all sides ([All](#)).
- Axis shorthands ([Vertical](#) for top/bottom and [Horizontal](#) for left/right).
- Individual sides ([Top](#), [Right](#), [Bottom](#), [Left](#)).

Precedence (from lowest to highest): 1) [All](#) (padding) 2) Axis shorthand via [Vertical](#)/[Horizontal](#) (padding: vertical horizontal) 3) Individual sides (padding-top/right/bottom/left) Later declarations override earlier ones when present.

Notes: - When only one axis is provided, the missing axis defaults to 0 (CSS zero). - Two-value padding shorthand uses the CSS form "padding: <vertical> <horizontal>", i.e., "top/bottom left/right".

[PaddingExtensions](#)

Extension methods on [StyleBuilder](#) for emitting padding declarations.

[Positioned](#)

A Blazor style contributor that emits CSS positioning declarations.

[PositionedExtensions](#)

Extension methods for [StyleBuilder](#) related to CSS positioning.

[RTBStyleBase](#)

Base component for contributing styles to a [StyleBuilder](#) within a Blazor render tree.

[Selector](#)

Blazor component that scopes child style contributions under a specific CSS selector/query.

[Size](#)

Contributes CSS size-related declarations (width/height and their min/max variants) to the current [StyleBuilder](#).

[SizeExtensions](#)

Extension methods for [StyleBuilder](#) to compose size-related declarations.

[Styled](#)

A component that provides a scoped CSS class and style builder context to its children.

[Transform](#)

Modern transform component: compose transforms via Parts and set optional Origin.

[TransformExtensions](#)

Extensions for fluent StyleBuilder usage.

[Transition](#)

Shorthand-first transitions with typed times and multi-item support.

[TransitionExtensions](#)

Style builder extensions for transitions.

[TransitionItem](#)

Single transition item for use in [Items](#).

[Visibility](#)

Sets the visibility of an element.

[VisibilityExtensions](#)

Style builder extensions for visibility.

Enums

[Animation.AnimationDirection](#)

Maps to CSS `animation-direction` keywords.

[Animation.AnimationFillMode](#)

Maps to CSS `animation-fill-mode` keywords.

[Animation.AnimationPlayState](#)

Maps to CSS `animation-play-state` keywords.

[Border.BorderCorner](#)

Flags enum describing which corner(s) should receive a radius.

[Border.BorderSide](#)

Flags enum describing which side(s) of a box should receive border declarations.

[Border.BorderStyle](#)

CSS `border-style` values supported by this component.

[Flex.Align](#)

Alignment in the cross axis (or multi-line distribution for [AlignContent](#)).

[Flex.AxisDirection](#)

Flexbox main-axis direction. When omitted, the browser default is `row`.

[Flex.Justify](#)

Distribution along the main axis.

[Flex.WrapMode](#)

Flexbox wrapping behavior. When omitted, the browser default is `nowrap`.

[Grid.Placement](#)

Alignment values mapped to the CSS `place-items` property.

[Overflow.OverflowMode](#)

Represents allowed values for CSS overflow properties.

[Positioned.PositionMode](#)

The CSS positioning mode to emit.

[Visibility.Mode](#)

The visibility mode.

Class Animation

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Describes animation-related CSS longhands and establishes an @keyframes scope for a single animation.

```
public class Animation : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Animation

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

- Non-visual: contributes CSS via a cascading [StyleBuilder](#); emits no visible DOM.
- [Name](#) is used for both [animation-name](#) and as the identifier of the associated [@keyframes](#) block.
- [Name](#) is cascaded so nested keyframe components can attach frames to the correct [@keyframes](#) block.
- Only animation longhand properties are emitted; the [animation](#) shorthand is never used.

Usage:

```
<Animation Name="fade"  
Duration="@{TimeSpan.FromMilliseconds(250)}"
```

```
TimingFunction="ease-out"
KeyFrames="..."/>
```

Properties

Composition

Maps to [animation-composition](#) (CSS Animations Level 2).

```
[Parameter]
public string? Composition { get; set; }
```

Property Value

[string](#)?

Remarks

- Typical values include `replace` (default in most engines) or `add`.
- Browser support may be limited/experimental; value is emitted as-is without validation.
- Omitted when null or whitespace.

Delay

Maps to [animation-delay](#). Omitted when `null`.

```
[Parameter]
public TimeSpan? Delay { get; set; }
```

Property Value

[TimeSpan](#)?

Remarks

- Uses the same formatting rules as [Duration](#).
- Examples: 75ms → "75ms", 2s → "2s".
- When omitted, the UA default is `0s`.

Direction

Maps to `animation-direction`. Omitted when `null`.

[Parameter]

```
public Animation.AnimationDirection? Direction { get; set; }
```

Property Value

[Animation.AnimationDirection?](#)

Duration

Maps to `animation-duration`. Omitted when `null`.

[Parameter]

```
public TimeSpan? Duration { get; set; }
```

Property Value

[TimeSpan?](#)

Remarks

- Uses seconds (s) for values $\geq 1\text{s}$ (up to 3 decimals) and milliseconds (ms) for values $< 1\text{s}$ (rounded to nearest ms).
- Examples: 250ms → "250ms", 1.5s → "1.5s", 2s → "2s".
- When omitted, the UA default is `0s`, which makes the animation instantaneous.

FillMode

Maps to `animation-fill-mode`. Omitted when `null`.

[Parameter]

```
public Animation.AnimationFillMode? FillMode { get; set; }
```

Property Value

[Animation.AnimationFillMode?](#)

Infinite

If true, sets `animation-iteration-count: infinite` and ignores [IterationCount](#).

[Parameter]

```
public bool Infinite { get; set; }
```

Property Value

[bool](#) ↗

IterationCount

Maps to `animation-iteration-count` when [Infinite](#) is false.

[Parameter]

```
public int? IterationCount { get; set; }
```

Property Value

[int](#) ↗?

Remarks

- Ignored when [Infinite](#) is true (then `infinite` is emitted).
- Not validated; callers should avoid negative values per CSS spec.
- When both this and [Infinite](#) are omitted/false, the UA default is [1](#).

KeyFrames

Child content that defines frames for the `@keyframes` named by [Name](#).

[Parameter]

[EditorRequired]

```
public required RenderFragment KeyFrames { get; set; }
```

Property Value

[RenderFragment](#) ↗

Remarks

The [Name](#) is cascaded so child keyframe components attach frames to the correct block.

Name

The animation name (CSS identifier).

```
[Parameter]
[EditorRequired]
public required string Name { get; set; }
```

Property Value

[string](#) ↗

Remarks

- Used for [animation-name](#) and as the identifier of the [@keyframes](#) block.
- Must be a valid CSS identifier; no validation or escaping is performed.

PlayState

Maps to [animation-play-state](#). Omitted when [null](#).

```
[Parameter]
public Animation.AnimationPlayState? PlayState { get; set; }
```

Property Value

[Animation.AnimationPlayState?](#)

TimingFunction

Maps to `animation-timing-function` (e.g., `ease`, `linear`, `ease-in-out`, `cubic-bezier(...)`, `steps(...)`).

[Parameter]

```
public string? TimingFunction { get; set; }
```

Property Value

[string](#)

Remarks

Omitted when null or whitespace. No validation is performed. When omitted, the UA default is `ease`.

Methods

BuildRenderTree(RenderTreeBuilder)

Emits a cascading value containing [Name](#) so nested keyframe components can bind to the corresponding `@keyframes` block.

```
protected override void BuildRenderTree(RenderTreeBuilder builder)
```

Parameters

`builder` [RenderTreeBuilder](#)

The render tree builder.

Remarks

- Produces no visual markup; this component affects only style generation and keyframes scope.
- The cascading parameter name is `Keyframe.AnimationName` to match keyframe components' expectations.

BuildStyle(StyleBuilder)

Contributes animation-related CSS declarations to the current [StyleBuilder](#).

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The builder to receive declarations, selectors, groups, and keyframes.

Remarks

This method is called by the infrastructure when [Contribute\(StyleBuilder\)](#) is invoked by the owning [StyleBuilder](#) during composition, provided [Condition](#) is true. Typical usage:

- Use `builder.Set` to add base declarations.
- Use `builder.Selector` to add nested selector rules.
- Use `builder.Media`, `Supports`, or `Container` for group rules.
- Use `builder.Keyframes` to define animations.

Avoid long-running or stateful operations; this method may be invoked multiple times during recomposition.

Enum Animation.AnimationDirection

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Maps to CSS `animation-direction` keywords.

```
public enum Animation.AnimationDirection
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

`Alternate = 2`

CSS: `alternate`. Even cycles 0%→100%, odd cycles 100%→0%.

`AlternateReverse = 3`

CSS: `alternate-reverse`. Even cycles 100%→0%, odd cycles 0%→100%.

`Normal = 0`

CSS: `normal`. Plays from 0% to 100% on each cycle.

`Reverse = 1`

CSS: `reverse`. Plays from 100% to 0% on each cycle.

Remarks

Normal → `normal`, Reverse → `reverse`, Alternate → `alternate`, AlternateReverse → `alternate-reverse`.

When omitted, the UA default is `normal`.

Enum Animation.AnimationFillMode

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Maps to CSS `animation-fill-mode` keywords.

```
public enum Animation.AnimationFillMode
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Backwards = 2

CSS: `backwards`. Apply styles from the first keyframe during the delay.

Both = 3

CSS: `both`. Applies both `forwards` and `backwards` behaviors.

Forwards = 1

CSS: `forwards`. Retain styles from the last keyframe after completion.

None = 0

CSS: `none`. Do not retain styles before/after execution.

Remarks

None → `none`, Forwards → `forwards`, Backwards → `backwards`, Both → `both`.

When omitted, the UA default is `none`.

Enum Animation.AnimationPlayState

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Maps to CSS `animation-play-state` keywords.

```
public enum Animation.AnimationPlayState
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

`Paused = 1`

CSS: `paused`. The animation is paused.

`Running = 0`

CSS: `running`. The animation is playing.

Remarks

Running → `running`, Paused → `paused`.

When omitted, the UA default is `running`.

Class AnimationStyleExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension helpers that map [Animation](#) parameters to CSS `animation-*` longhand declarations.

```
public static class AnimationStyleExtensions
```

Inheritance

[object](#) ← AnimationStyleExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- These helpers set individual longhand properties; no shorthand `animation` is emitted.
- Methods omit properties when inputs are null/whitespace; no validation of CSS values is performed.
- Returned [StyleBuilder](#) enables fluent chaining.

Methods

AnimationDelay(StyleBuilder, TimeSpan?)

Sets `animation-delay` using s/ms formatting.

```
public static StyleBuilder AnimationDelay(this StyleBuilder b, TimeSpan? t)
```

Parameters

b [StyleBuilder](#)

t [TimeSpan](#)?

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Remarks

Uses the same formatting and omission rules as [AnimationDuration\(StyleBuilder, TimeSpan?\)](#).

AnimationDirection(StyleBuilder, AnimationDirection?)

Sets `animation-direction` when `dir` is not null; otherwise omitted.

```
public static StyleBuilder AnimationDirection(this StyleBuilder b,  
    Animation.AnimationDirection? dir)
```

Parameters

`b` [StyleBuilder](#)

`dir` [Animation.AnimationDirection?](#)

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Remarks

Maps enum values to CSS keywords: Normal → `normal`, Reverse → `reverse`, Alternate → `alternate`, AlternateReverse → `alternate-reverse`.

AnimationDuration(StyleBuilder, TimeSpan?)

Sets `animation-duration` using s/ms formatting.

```
public static StyleBuilder AnimationDuration(this StyleBuilder b, TimeSpan? t)
```

Parameters

`b` [StyleBuilder](#)

`t TimeSpan?`

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Remarks

- When `t` is null, an empty value is passed which is ignored by the underlying declaration set.
- Values < 1 second are rounded to the nearest millisecond (e.g., 249.6ms → "250ms").
- Values ≥ 1 second use seconds; integers are emitted without decimals (e.g., "2s"), fractional values use up to 3 decimals (e.g., "1.25s").

AnimationFillMode(StyleBuilder, AnimationFillMode?)

Sets `animation-fill-mode` when `fill` is not null; otherwise omitted.

```
public static StyleBuilder AnimationFillMode(this StyleBuilder b,  
    Animation.AnimationFillMode? fill)
```

Parameters

`b` [StyleBuilder](#)

`fill` [Animation.AnimationFillMode?](#)

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Remarks

Maps enum values to CSS keywords: None → `none`, Forwards → `forwards`, Backwards → `backwards`, Both → `both`.

AnimationIterationCount(StyleBuilder, int?, bool)

Sets `animation-iteration-count` to either a number or `infinite`.

```
public static StyleBuilder AnimationIterationCount(this StyleBuilder b, int? count, bool infinite = false)
```

Parameters

`b` [StyleBuilder](#)

The [StyleBuilder](#) being configured.

`count` [int](#)?

Number of iterations when `infinite` is false. Not validated; avoid negative values per CSS spec.

`infinite` [bool](#)

When true, sets `infinite` and ignores `count`.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

AnimationName(StyleBuilder, string?)

Sets `animation-name` when `name` is not null/whitespace; otherwise omitted.

```
public static StyleBuilder AnimationName(this StyleBuilder b, string? name)
```

Parameters

`b` [StyleBuilder](#)

`name` [string](#)

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

AnimationPlayState(StyleBuilder, AnimationPlayState?)

Sets `animation-play-state` when `st` is not null; otherwise omitted.

```
public static StyleBuilder AnimationPlayState(this StyleBuilder b,  
Animation.AnimationPlayState? st)
```

Parameters

`b` [StyleBuilder](#)

`st` [Animation.AnimationPlayState?](#)

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Remarks

Maps enum values to CSS keywords: Running → `running`, Paused → `paused`.

AnimationTiming(StyleBuilder, string?)

Sets `animation-timing-function` when `tf` is not null/whitespace; otherwise omitted.

```
public static StyleBuilder AnimationTiming(this StyleBuilder b, string? tf)
```

Parameters

`b` [StyleBuilder](#)

`tf` [string](#) ↗

Returns

StyleBuilder

The same [StyleBuilder](#) instance for chaining.

Class Background

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes a CSS background-color declaration to the current style scope.

```
public class Background : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Background

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

As a Blazor style contributor:

```
<Background Color="@RTBColor.FromCss("royalblue")" />
```

In a style build routine:

```
builder.Background(RTBColor.FromCss("#09f"));
```

Remarks

- This component participates in style composition via the cascading [StyleBuilder](#) from [RTBStyleBase](#).
- When [Condition](#) is true and [Color](#) is non-null, a "background-color" declaration is emitted.
- When [Color](#) is [null](#), no declaration is produced for background-color.

Properties

Color

The background color to apply. When [null](#), no "background-color" declaration is generated.

```
[Parameter]
public RTBColor? Color { get; set; }
```

Property Value

[RTBColor?](#)

Methods

BuildStyle(StyleBuilder)

Contributes the "background-color" declaration using the configured [Color](#).

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

[builder](#) [StyleBuilder](#)

The target [StyleBuilder](#) to receive the declaration.

Class BackgroundExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension helpers for adding background-related CSS to a [StyleBuilder](#).

```
public static class BackgroundExtensions
```

Inheritance

[object](#) ← BackgroundExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Background(StyleBuilder, RTBColor?)

Adds a "background-color" declaration to the builder if `color` is non-null.

```
public static StyleBuilder Background(this StyleBuilder builder, RTBColor? color)
```

Parameters

`builder` [StyleBuilder](#)

The style builder to mutate.

`color` [RTBColor?](#)

The color to set. When `null`, nothing is added.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for fluent chaining.

Examples

```
builder  
    .Background(RTBColor.FromCss("hsl(210 100% 40% / 50%"));
```

Class Border

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS `border` and `border-radius` declarations to the cascading [StyleBuilder](#).

```
public class Border : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Border

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Typical usage inside a style scope (component providing a cascading StyleBuilder):

```
<Border Side="Border.BorderSide.All"  
       Style="Border.BorderStyle.Solid"  
       Width="SizeUnit.Px(2)"  
       Color="RTBColors.RoyalBlue" />  
  
<Border Corner="Border.BorderCorner.Top"  
       Radius="SizeUnit.Rem(0.5)" />
```

Remarks

- Use [Side](#), [Style](#), [Width](#), and [Color](#) to render borders on one or more sides.
- Use [Corner](#) and [Radius](#) to control per-corner or all-corners radius.
- Defaults when unspecified:
 - When a border is requested ([Side](#) not null), [Width](#) defaults to [1px](#).
 - When a border is requested, [Color](#) defaults to black (#000).
 - When a radius is requested ([Corner](#) not null), [Radius](#) defaults to [1px](#).
- Special cases:
 - [None](#) results in `border: unset`.
 - [All](#) uses the shorthand `border: [width] [style] [color]`.

Properties

Color

The border color to use. When [Side](#) is provided and this is null, black is used.

[Parameter]

```
public RTBColor? Color { get; set; }
```

Property Value

[RTBColor?](#)

Corner

Which corner(s) should receive the border radius. Supports individual corners and grouped flags (e.g., [Top](#) or [All](#)).

[Parameter]

```
public Border.BorderCorner? Corner { get; set; }
```

Property Value

[Border.BorderCorner?](#)

Radius

The border radius to apply. When [Corner](#) is provided and this is null, a default of [1px](#) is used.

```
[Parameter]  
public SizeUnit? Radius { get; set; }
```

Property Value

[SizeUnit?](#)

Side

Which side(s) to apply a border to. When set, a border is emitted using [Width](#) (defaults to 1px when null), [Style](#), and [Color](#) (defaults to black when null).

```
[Parameter]  
public Border.BorderSide? Side { get; set; }
```

Property Value

[Border.BorderSide?](#)

Style

The CSS border style keyword to use (e.g., `solid`, `dashed`). Default is [Solid](#).

```
[Parameter]  
public Border.BorderStyle Style { get; set; }
```

Property Value

[Border.BorderStyle](#)

Width

The border width to use. When [Side](#) is provided and this is null, a default of [1px](#) is used.

```
[Parameter]
public SizeUnit? Width { get; set; }
```

Property Value

[SizeUnit?](#)

Methods

BuildStyle(StyleBuilder)

Contributes declarations to the builder:

- Emits `border-radius` declarations when [Corner](#) is not null (with [Radius](#) or default 1px).
- Emits `border` declarations when [Side](#) is not null (with [Width](#) default 1px and [Color](#) default black if omitted).

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The style builder to receive the declarations.

Enum Border.BorderCorner

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Flags enum describing which corner(s) should receive a radius.

```
[Flags]
public enum Border.BorderCorner
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

All = Top | Bottom

All four corners.

Bottom = BottomLeft | BottomRight

Both bottom corners.

BottomLeft = 4

Bottom-left corner.

BottomRight = 8

Bottom-right corner.

Left = TopRight | BottomLeft

Both left corners.

None = 0

No corners.

Right = TopRight | BottomRight

Both right corners.

Top = TopLeft | TopRight

Both top corners.

TopLeft = 1

Top-left corner.

TopRight = 2

Top-right corner.

Enum Border.BorderSide

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Flags enum describing which side(s) of a box should receive border declarations.

```
[Flags]  
public enum Border.BorderSide
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

All = Vertical | Horizontal

All four sides.

Bottom = 4

Bottom side.

Horizontal = Right | Left

Left and right sides.

Left = 8

Left side.

None = 0

No sides.

Right = 2

Right side.

Top = 1

Top side.

Vertical = Top | Bottom

Top and bottom sides.

Enum Border.BorderStyle

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

CSS `border-style` values supported by this component.

```
public enum Border.BorderStyle
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Dashed = 3

Dashed border.

Dotted = 2

Dotted border.

Double = 4

Double-line border.

Groove = 5

Groove border.

Inset = 7

Inset border.

None = 0

No border.

Outset = 8

Outset border.

Ridge = 6

Ridge border.

Solid = 1

Solid border.

Class BorderStyleExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods on [StyleBuilder](#) for composing border and radius declarations.

```
public static class BorderStyleExtensions
```

Inheritance

[object](#) ← BorderStyleExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Border(StyleBuilder, SizeUnit?, BorderStyle, RTBColor?, BorderSide?)

Emits border declarations for the specified side(s).

```
public static StyleBuilder Border(this StyleBuilder builder, SizeUnit? width,  
Border.BorderStyle style, RTBColor? color, Border.BorderSide? side)
```

Parameters

builder [StyleBuilder](#)

The target style builder.

width [SizeUnit?](#)

Width to use; when null, defaults to [1px](#).

style [Border.BorderStyle](#)

Border style keyword.

`color` [RTBColor](#)?

Color to use; when null, defaults to black.

`side` [Border.BorderSide](#)?

Side(s) to apply:

- [None](#): emits `border: unset`
- [All](#): emits shorthand `border`
- Mixed flags: emits per-side properties (`border-top`, `border-right`, etc.)

Returns

[StyleBuilder](#)

The builder for chaining.

BorderAll(StyleBuilder, SizeUnit?, RTBColor, BorderStyle)

Sets `border` shorthand for all sides: `[width] [style] [color]`.

```
public static StyleBuilder BorderAll(this StyleBuilder builder, SizeUnit? width, RTBColor color, Border.BorderStyle style)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`width` [SizeUnit](#)?

The border width (e.g., 1px).

`color` [RTBColor](#)

The border color.

`style` [Border.BorderStyle](#)

The border style keyword.

Returns

[StyleBuilder](#)

The builder for chaining.

BorderNone(StyleBuilder)

Unsets the border: `border: unset`.

```
public static StyleBuilder BorderNone(this StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

Returns

[StyleBuilder](#)

The builder for chaining.

BorderRadius(StyleBuilder, SizeUnit?, BorderCorner?)

Emits `border-radius` declarations for the specified corner(s).

```
public static StyleBuilder BorderRadius(this StyleBuilder builder, SizeUnit? radius,
Border.BorderCorner? corner)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`radius` [SizeUnit?](#)

Radius to use; when null, defaults to `1px` if any corner is specified.

`corner` [Border.BorderCorner?](#)

Corner(s) to apply:

- [None](#): emits `border-radius: unset`
- [All](#): emits uniform `border-radius`
- Mixed flags: emits the corresponding per-corner properties.

Returns

[StyleBuilder](#)

The builder for chaining.

`BorderRadiusAll(StyleBuilder, SizeUnit?)`

Sets a uniform `border-radius` value for all corners.

```
public static StyleBuilder BorderRadiusAll(this StyleBuilder builder, SizeUnit? radius)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`radius` [SizeUnit?](#)

The radius to apply. If null, nothing is emitted.

Returns

[StyleBuilder](#)

The builder for chaining.

`BorderRadiusNone(StyleBuilder)`

Unsets any corner radius: `border-radius: unset`.

```
public static StyleBuilder BorderRadiusNone(this StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The target style builder.

Returns

[StyleBuilder](#)

The builder for chaining.

Class Color

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes a CSS `color` declaration to the current [StyleBuilder](#) scope.

```
public class Color : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Color

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Remarks

- When [Condition](#) is true, this component registers itself with the cascading [StyleBuilder](#) and emits a `color` declaration during composition.
- When [Value](#) is `null`, no declaration is written (the property is omitted).
- The value is serialized using [RTBColor](#)'s formatting (e.g., hex RGBA).

Properties

Value

The color to apply to the CSS `color` property.

```
[Parameter]  
public RTBColor? Value { get; set; }
```

Property Value

[RTBColor?](#)

Remarks

- `null` omits the declaration.
- Use [Parse\(string\)](#) to create a value from a CSS string if needed.

Methods

BuildStyle(StyleBuilder)

Emits the CSS `color` declaration when [Value](#) is not `null`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The style builder receiving the declaration.

Remarks

See [BuildStyle\(StyleBuilder\)](#) for lifecycle details.

Class ColorExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

[StyleBuilder](#) extensions for the CSS `color` property.

```
public static class ColorExtensions
```

Inheritance

[object](#) ← ColorExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Color(StyleBuilder, RTBColor?)

Sets the CSS `color` property when `color` is not `null`.

```
public static StyleBuilder Color(this StyleBuilder builder, RTBColor? color)
```

Parameters

`builder` [StyleBuilder](#)

The target [StyleBuilder](#).

`color` [RTBColor?](#)

The color value to serialize and assign; `null` omits the declaration.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

Remarks

If the property is set multiple times, the last value wins (last-write-wins).

Class Flex

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS declarations for a flex container (display: flex) to the current [StyleBuilder](#) scope.

```
public class Flex : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Flex

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

```
// Equivalent via the extensions API:  
var css = StyleBuilder.Start  
    .Flex(direction: Flex.AxisDirection.Row,  
          wrap: Flex.WrapMode.NoWrap,  
          justifyContent: Flex.Justify.SpaceBetween,  
          alignItems: Flex.Align.Center,  
          gap: Spacing.Rem(1),  
          grow: 1)  
    .BuildScoped(".container");
```

Remarks

- This component writes only the properties you specify; when a parameter is null, the corresponding declaration is omitted and the browser default applies.
- Typical browser defaults (when not explicitly set):
flex-direction: row; flex-wrap: nowrap; justify-content: flex-start; align-items: stretch; align-content: normal;
flex-grow: 0; flex-shrink: 1.
- Place this component within a style scope that supplies the cascading [StyleBuilder](#).

Properties

AlignContent

Sets `align-content` (distribution of lines in a multi-line flex container). When null, browser default applies.

```
[Parameter]
public Flex.Align? AlignContent { get; set; }
```

Property Value

[Flex.Align?](#)

Remarks

Do not use [Baseline](#) with `align-content`; it is not a valid value for that property.

AlignItems

Sets `align-items` (alignment of items on the cross axis). When null, browser default applies.

```
[Parameter]
public Flex.Align? AlignItems { get; set; }
```

Property Value

[Flex.Align?](#)

Direction

Sets **flex-direction**. When null, no declaration is written (default: `row`).

[Parameter]

```
public Flex.AxisDirection? Direction { get; set; }
```

Property Value

[Flex.AxisDirection?](#)

Gap

Sets the **gap** between items (length or percentage, e.g., `1rem`, `8px`, `4%`).

[Parameter]

```
public Spacing? Gap { get; set; }
```

Property Value

[Spacing?](#)

Remarks

Note: The CSS keyword `auto` is not valid for **gap**. Avoid using [Auto](#) here.

Grow

Sets **flex-grow** (non-negative integer). When null, no declaration is written (default: `0`).

[Parameter]

```
public int? Grow { get; set; }
```

Property Value

[int?](#)

JustifyContent

Sets `justify-content` (distribution along the main axis). When null, browser default applies.

```
[Parameter]
public Flex.Justify? JustifyContent { get; set; }
```

Property Value

[Flex.Justify?](#)

Shrink

Sets `flex-shrink` (non-negative integer). When null, no declaration is written (default: `1`).

```
[Parameter]
public int? Shrink { get; set; }
```

Property Value

[int?](#)

Wrap

Sets `flex-wrap`. When null, no declaration is written (default: `nowrap`).

```
[Parameter]
public Flex.WrapMode? Wrap { get; set; }
```

Property Value

[Flex.WrapMode?](#)

Methods

BuildStyle(StyleBuilder)

Contributes the flex container declarations to the provided `builder`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The style builder receiving the flex container declarations.

Enum Flex.Align

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Alignment in the cross axis (or multi-line distribution for [AlignContent](#)).

```
public enum Flex.Align
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Baseline = 4

Maps to align-*: baseline;

Center = 2

Maps to align-*: center;

End = 1

Maps to align-*: flex-end;

Start = 0

Maps to align-*: flex-start;

Stretch = 3

Maps to align-*: stretch;

Remarks

Notes:

- **Baseline** is valid for `align-items/align-self`, but not for `align-content`. Avoid using [Baseline](#) with [AlignContent](#); many browsers will ignore it.

Enum Flex.AxisDirection

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Flexbox main-axis direction. When omitted, the browser default is `row`.

```
public enum Flex.AxisDirection
```

Extension Methods

[CssEnumExtensions.ToCss\(Enumerable\)](#)

Fields

`Column = 2`

Maps to `flex-direction: column;`

`ColumnReverse = 3`

Maps to `flex-direction: column-reverse;`

`Row = 0`

Maps to `flex-direction: row;`

`RowReverse = 1`

Maps to `flex-direction: row-reverse;`

Enum Flex.Justify

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Distribution along the main axis.

```
public enum Flex.Justify
```

Extension Methods

[CssEnumExtensions.ToCss\(Enumerable\)](#)

Fields

Center = 2

Maps to justify-content: center;

End = 1

Maps to justify-content: flex-end;

SpaceAround = 4

Maps to justify-content: space-around;

SpaceBetween = 3

Maps to justify-content: space-between;

SpaceEvenly = 5

Maps to justify-content: space-evenly;

Start = 0

Maps to justify-content: flex-start;

Enum Flex.WrapMode

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Flexbox wrapping behavior. When omitted, the browser default is `nowrap`.

```
public enum Flex.WrapMode
```

Extension Methods

[CssEnumExtensions.ToCss\(Enumerable\)](#)

Fields

`NoWrap` = 0

Maps to `flex-wrap: nowrap;`

`Wrap` = 1

Maps to `flex-wrap: wrap;`

`WrapReverse` = 2

Maps to `flex-wrap: wrap-reverse;`

Class FlexExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension helpers for composing flex container styles fluently.

```
public static class FlexExtensions
```

Inheritance

[object](#) ← FlexExtensions

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

Applies the same CSS as the [Flex\(StyleBuilder, AxisDirection?, WrapMode?, Justify?, Align?, Align?, Spacing?, int?, int?\)](#) component but directly on a [StyleBuilder](#). Only the provided arguments are emitted; omitted arguments leave browser defaults in effect.

Methods

Flex(StyleBuilder, AxisDirection?, WrapMode?, Justify?, Align?, Align?, Spacing?, int?, int?)

Adds flex container declarations to the builder.

```
public static StyleBuilder Flex(this StyleBuilder builder, Flex.AxisDirection? direction = null, Flex.WrapMode? wrap = null, Flex.Justify? justifyContent = null, Flex.Align? alignItems = null, Flex.Align? alignContent = null, Spacing? gap = null, int? shrink = null, int? grow = null)
```

Parameters

builder [StyleBuilder](#)

The [StyleBuilder](#) to mutate.

direction [Flex.AxisDirection?](#)

Maps to `flex-direction`. Null omits the declaration.

wrap [Flex.WrapMode?](#)

Maps to `flex-wrap`. Null omits the declaration.

justifyContent [Flex.Justify?](#)

Maps to `justify-content`. Null omits the declaration.

alignItems [Flex.Align?](#)

Maps to `align-items`. Null omits the declaration.

alignContent [Flex.Align?](#)

Maps to `align-content`. Null omits the declaration. Avoid [Baseline](#); it is not valid for this property.

gap [Spacing?](#)

Maps to `gap`. Use lengths or percentages (e.g., [Spacing.Px\(8\)](#), [Spacing.Rem\(1\)](#)). Do not use [Auto](#).

shrink [int?](#)

Maps to `flex-shrink` (non-negative integer). Null omits the declaration.

grow [int?](#)

Maps to `flex-grow` (non-negative integer). Null omits the declaration.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

Examples

```
var css = StyleBuilder.Start
    .Flex(direction: Flex.AxisDirection.Row, gap: Spacing.Em(0.5))
    .BuildScoped(".container");
```

Class Grid

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS for a grid container to the current [StyleBuilder](#) scope. When placed inside a style scope, this component sets:

- display: grid
- grid-template-columns
- grid-template-rows
- gap (optional)
- place-items (optional)

```
public class Grid : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Grid

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

Usage (as a Blazor style-contributor component):

```
<StyleScope>
  <Grid TemplateColumns="repeat(3, 1fr)"
        TemplateRows="auto 1fr"
        Gap="@Spacing.Rem(1)"
        ItemPlacement="Grid.Placement.Center" />
</StyleScope>
```

Usage (via [Grid\(StyleBuilder, string, string, Spacing?, Placement?\)](#) on a [StyleBuilder](#)):

```
var css = StyleBuilder.Start
    .Grid("repeat(12, 1fr)", "auto", Spacing.Rem(1), Grid.Placement.Start)
    .BuildScoped(".my-grid");
```

Properties

Gap

The CSS `gap` between grid rows and columns.

[Parameter]
public Spacing? Gap { get; set; }

Property Value

[Spacing?](#)

Remarks

When provided, emits `gap: {value}`. When `null`, no `gap` is set. Use [Spacing helpers](#) (e.g., [Rem\(double\)](#), [Px\(double\)](#)).

ItemPlacement

The CSS `place-items` value to align items inside the grid container.

[Parameter]
public Grid.Placement? ItemPlacement { get; set; }

Property Value

[Grid.Placement](#)?

Remarks

When provided, emits `place-items: {value}`. When `null`, no `place-items` is set. See [Grid.Placement](#) for available values and their CSS mapping.

TemplateColumns

The value for CSS `grid-template-columns`.

```
[Parameter]
public string TemplateColumns { get; set; }
```

Property Value

[string](#) ↗

Remarks

Accepts any valid CSS value (e.g., `1fr`, `repeat(3, 1fr)`, `200px auto 1fr`). Defaults to `1fr`.

TemplateRows

The value for CSS `grid-template-rows`.

```
[Parameter]
public string TemplateRows { get; set; }
```

Property Value

[string](#) ↗

Remarks

Accepts any valid CSS value (e.g., `auto`, `repeat(2, minmax(0, 1fr))`, `auto 1fr auto`). Defaults to `1fr`.

Methods

BuildStyle(StyleBuilder)

Contributes CSS declarations to the provided `builder`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The builder to receive declarations, selectors, groups, and keyframes.

Remarks

This method is called by the infrastructure when [Contribute\(StyleBuilder\)](#) is invoked by the owning [StyleBuilder](#) during composition, provided [Condition](#) is true. Typical usage:

- Use `builder.Set` to add base declarations.
- Use `builder.Selector` to add nested selector rules.
- Use `builder.Media`, `Supports`, or `Container` for group rules.
- Use `builder.Keyframes` to define animations.

Avoid long-running or stateful operations; this method may be invoked multiple times during recomposition.

Enum Grid.Placement

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Alignment values mapped to the CSS `place-items` property.

```
public enum Grid.Placement
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

`Center` = 3

Emits `center`.

`End` = 2

Emits `end`.

`FlexEnd` = 5

Emits `flex-end`.

`FlexStart` = 6

Emits `flex-start`.

`Normal` = 0

Emits `normal`.

`Start` = 1

Emits `start`.

`Stretch` = 4

Emits `stretch`.

Remarks

Mappings:

- Normal -> `normal`
- Start -> `start`
- End -> `end`
- Center -> `center`
- Stretch -> `stretch`
- FlexStart -> `flex-start`
- FlexEnd -> `flex-end`

Class GridExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods for configuring grid-related CSS on a [StyleBuilder](#).

```
public static class GridExtensions
```

Inheritance

[object](#) ← GridExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

Use when composing styles fluently without the component.

Methods

Grid(StyleBuilder, string, string, Spacing?, Placement?)

Configures the current [StyleBuilder](#) as a CSS Grid container.

```
public static StyleBuilder Grid(this StyleBuilder builder, string templateColumns = "1fr",  
    string templateRows = "1fr", Spacing? gap = null, Grid.Placement? itemPlacement = null)
```

Parameters

builder [StyleBuilder](#)

The style builder to mutate.

templateColumns [string](#)

CSS `grid-template-columns` value. Defaults to `1fr`.

templateRows [string](#)

CSS `grid-template-rows` value. Defaults to `1fr`.

gap [Spacing?](#)

Optional CSS `gap` value. When `null`, no gap is emitted.

itemPlacement [Grid.Placement?](#)

Optional CSS `place-items` value via [Grid.Placement](#).

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Examples

```
StyleBuilder.Start
    .Grid("repeat(4, 1fr)", "auto 1fr auto", Spacing.Rem(1), Grid.Placement.Center)
    .BuildScoped(".grid");
```

Class GridPlacement

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS Grid placement declarations to a [StyleBuilder](#).

```
public class GridPlacement : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← GridPlacement

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Usage in a style scope:

Produces:

```
grid-column: 2 / span 3;  
grid-row: 1 / span 2;
```

Remarks

- Emits `grid-column` and `grid-row` declarations using explicit start positions (when > 0) or `auto`, combined with their respective spans.
- This component renders no visual markup; it participates in style composition only.
- When [Column](#) or [Row](#) is less than or equal to 0, `auto` is used for the start.
- No validation is performed on spans; callers should pass values greater than or equal to 1.

Properties

ChildContent

Optional child content. Captured but not rendered by this component.

```
[Parameter]
public RenderFragment ChildContent { get; set; }
```

Property Value

[RenderFragment](#)

Column

The 1-based starting grid column track. When less than or equal to 0, the start resolves to `auto`.

```
[Parameter]
public int Column { get; set; }
```

Property Value

[int](#)

ColumnSpan

The number of columns to span. Should be greater than or equal to 1. Defaults to 1.

```
[Parameter]
public int ColumnSpan { get; set; }
```

Property Value

[int](#)

Row

The 1-based starting grid row track. When less than or equal to 0, the start resolves to `auto`.

```
[Parameter]
public int Row { get; set; }
```

Property Value

[int](#)

RowSpan

The number of rows to span. Should be greater than or equal to 1. Defaults to 1.

```
[Parameter]
public int RowSpan { get; set; }
```

Property Value

[int](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The style builder receiving the grid placement declarations.

Remarks

Writes:

- `grid-column: [Column|auto] / span [ColumnSpan]`
- `grid-row: [Row|auto] / span [RowSpan]`

Class GridPlacementExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods for composing CSS Grid placement with a [StyleBuilder](#).

```
public static class GridPlacementExtensions
```

Inheritance

[object](#) ← GridPlacementExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

GridPlacement(StyleBuilder, int, int, int, int)

Adds `grid-column` and `grid-row` declarations to the builder using the provided placement options.

```
public static StyleBuilder GridPlacement(this StyleBuilder builder, int column = 0, int  
columnSpan = 1, int row = 0, int rowSpan = 1)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`column` [int](#)

1-based starting column; when less than or equal to 0, uses `auto`.

`columnSpan` [int](#)

Number of columns to span; should be greater than or equal to 1.

`row` [int](#)

1-based starting row; when less than or equal to 0, uses `auto`.

`rowSpan` [int](#)

Number of rows to span; should be greater than or equal to 1.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for fluent chaining.

Examples

```
builder.GridPlacement(column: 2, columnSpan: 2, row: 1, rowspan: 3);  
// grid-column: 2 / span 2; grid-row: 1 / span 3;
```

Class Keyframe

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Represents a single keyframe entry inside an ancestor animation's `@keyframes` block.

```
public class Keyframe : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Keyframe

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Example (conceptual):

```
@* Contribute declarations for the 0% frame (e.g., opacity: 0.2; transform: scale(0.9);) *@
```

```
@* Contribute declarations for the 100% frame (e.g., opacity: 1; transform: scale(1);) *@
```

Results in CSS similar to:

```
@keyframes pulse{0%{opacity:0.2;transform:scale(0.9);}100%{opacity:1;transform:scale(1);}}
```

Remarks

- An ancestor component is expected to provide a cascading [AnimationName](#) (the name of the keyframes block).
- Child content contributes CSS declarations for this specific frame via the cascading [StyleBuilder](#) context supplied by this component.
- When composed, the collected declarations are emitted as a [KeyframeFrame](#) at the specified [Offset](#) inside the named keyframes.

Properties

AnimationName

The name of the `@keyframes` block to which this frame contributes.

```
[CascadingParameter(Name = "AnimationName")]
public string AnimationName { get; set; }
```

Property Value

[string](#) ↗

Remarks

Provided via cascading parameter by an ancestor animation component. When empty or whitespace, this component contributes nothing during composition.

ChildContent

Child content that contributes CSS declarations for this frame.

```
[Parameter]
[EditorRequired]
public required RenderFragment ChildContent { get; set; }
```

Property Value

[RenderFragment](#) ↗

Remarks

The child content receives a cascading [StyleBuilder](#) instance that targets this frame only. Use RTB Styled child components that write to the provided builder (e.g., components that call [Set\(string, string\)](#) or add nested rules) to define declarations for the frame.

Offset

The keyframe offset selector for this frame.

```
[Parameter]
[EditorRequired]
public required string Offset { get; set; }
```

Property Value

[string](#)

Remarks

Accepts standard keyframe selectors such as:

- "from" (equivalent to "0%")
- "to" (equivalent to "100%")
- Percentage values like "0%", "50%", "100%". The value is not validated; provide a valid CSS keyframe selector.

Methods

BuildRenderTree(RenderTreeBuilder)

Supplies a fixed cascading [StyleBuilder](#) to [ChildContent](#) so it can emit declarations for this keyframe.

```
protected override void BuildRenderTree(RenderTreeBuilder builder)
```

Parameters

builder [RenderTreeBuilder](#)

The Blazor [RenderTreeBuilder](#).

Remarks

The cascading value is marked as fixed to avoid unnecessary re-renders and guarantees that all child contributors target the same frame-local builder instance.

BuildStyle(StyleBuilder)

Contributes a [KeyframeFrame](#) into the named `@keyframes` block when any declarations were produced by `RTB.Blazor.Styled.Components.Keyframe._frameBuilder`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The ambient style builder composing the overall stylesheet.

Remarks

Workflow:

1. If [AnimationName](#) is null/whitespace, no work is performed.
2. Compose the frame builder to gather declarations from [ChildContent](#).
3. If the frame builder's base declarations are empty, skip emission.
4. Otherwise, add/modify the keyframes block named [AnimationName](#) and insert a frame at [Offset](#).
5. Clear the frame builder to avoid leaking declarations into subsequent compositions.

Class Margin

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS margin declarations to the current [StyleBuilder](#) scope.

```
public class Margin : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Margin

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Usage in a component that provides a cascading StyleBuilder:

```
<Margin All="Spacing.Rem(1)" />  
<Margin Horizontal="Spacing.Px(8)" Vertical="Spacing.Px(12)" />  
<Margin Vertical="Spacing.Px(12)" Top="Spacing.Px(4)" />
```

Remarks

Precedence (later entries override earlier ones):

1. [All](#) sets the "margin" shorthand for all sides.
2. [Horizontal/Vertical](#) set the "margin" shorthand as "`<vertical-or-0> <horizontal-or-0>`" (top/bottom, left/right). If only one of the two is provided, the missing counterpart defaults to `0`.
3. Side-specific properties ([Top](#), [Right](#), [Bottom](#), [Left](#)) override any previous shorthand. Null values are ignored.

Properties

All

Shorthand to set the same margin on all four sides (emits `margin: ...`). Overridden by [Horizontal](#), [Vertical](#), and side-specific properties when provided.

[Parameter]
`public Spacing? All { get; set; }`

Property Value

[Spacing?](#)

Bottom

Sets the bottom margin (emits `margin-bottom`). Overrides any previously emitted shorthand.

[Parameter]
`public Spacing? Bottom { get; set; }`

Property Value

[Spacing?](#)

Horizontal

Shorthand for left/right margins. When [Horizontal](#) or [Vertical](#) is set, emits `margin: <vertical-or-0> <horizontal-or-0>`.

```
[Parameter]
public Spacing? Horizontal { get; set; }
```

Property Value

[Spacing?](#)

Left

Sets the left margin (emits `margin-left`). Overrides any previously emitted shorthand.

```
[Parameter]
public Spacing? Left { get; set; }
```

Property Value

[Spacing?](#)

Right

Sets the right margin (emits `margin-right`). Overrides any previously emitted shorthand.

```
[Parameter]
public Spacing? Right { get; set; }
```

Property Value

[Spacing?](#)

Top

Sets the top margin (emits `margin-top`). Overrides any previously emitted shorthand.

```
[Parameter]
public Spacing? Top { get; set; }
```

Property Value

[Spacing?](#)

Vertical

Shorthand for top/bottom margins. When [Horizontal](#) or [Vertical](#) is set, emits `margin: <vertical-or-0> <horizontal-or-0>`.

[Parameter]

```
public Spacing? Vertical { get; set; }
```

Property Value

[Spacing?](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The style builder receiving declarations.

Remarks

Emission order ensures the precedence described in the class remarks:

- 1) [All](#), 2) [Horizontal/Vertical](#), 3) side-specific properties.

Class MarginExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Fluent extensions for applying CSS margin declarations using [StyleBuilder](#).

```
public static class MarginExtensions
```

Inheritance

[object](#) ← MarginExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Margin(StyleBuilder, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?)

Applies margin declarations to the builder.

```
public static StyleBuilder Margin(this StyleBuilder builder, Spacing? all = null, Spacing?  
top = null, Spacing? right = null, Spacing? bottom = null, Spacing? left = null, Spacing?  
horizontal = null, Spacing? vertical = null)
```

Parameters

builder [StyleBuilder](#)

The target style builder.

all [Spacing?](#)

Shorthand for all four sides.

top [Spacing?](#)

Top margin.

[right Spacing?](#)

Right margin.

[bottom Spacing?](#)

Bottom margin.

[left Spacing?](#)

Left margin.

[horizontal Spacing?](#)

Shorthand for left/right.

[vertical Spacing?](#)

Shorthand for top/bottom.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

Remarks

Precedence (later entries override earlier ones):

1. `all` sets the "margin" shorthand for all sides.
2. `horizontal/vertical` set the "margin" shorthand as "`<vertical-or-0> <horizontal-or-0>`". If only one of the two is provided, the missing counterpart defaults to `0`.
3. Side-specific parameters override any previous shorthand.

Class Media

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes a `@media` group to the ambient [StyleBuilder](#) using a provided [BreakPoint](#).

```
public class Media : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Media

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

This component establishes a private inner [StyleBuilder](#) that is exposed to its descendants via a [CascadingValue< TValue >](#). Children that contribute styles will write into this inner builder. During composition, the accumulated child styles are emitted as a single `@media` block on the parent builder, using the media query generated by [ToQuery\(\)](#).

The inner builder is cleared after each composition to avoid stale declarations across renders. The contribution is gated by [Condition](#).

Example:

```
<Media BreakPoint="new BreakPoint { Media = BreakPoint.MediaType.Screen, MinWidth = 992 }">  
  <StyleSet Set="b => b.Set("display", "grid")" />
```

```
</Media>
```

This results in:

```
@media screen and (min-width: 992px) {  
    .scope { display: grid; }  
}
```

Properties

BreakPoint

Describes the media query to apply. Converted to a CSS query string via [ToQuery\(\)](#).

```
[Parameter]  
[EditorRequired]  
public required BreakPoint BreakPoint { get; set; }
```

Property Value

[BreakPoint](#)

ChildContent

Child content that contributes styles to the media query via the inner [StyleBuilder](#).

```
[Parameter]  
[EditorRequired]  
public required RenderFragment ChildContent { get; set; }
```

Property Value

[RenderFragment](#) ↗

Methods

BuildRenderTree(RenderTreeBuilder)

Renders a fixed [CascadingValue< TValue >](#) that supplies the private RTB.Blazor.Styled.Components.Media._inner builder to descendants.

```
protected override void BuildRenderTree(RenderTreeBuilder renderBuilder)
```

Parameters

renderBuilder [RenderTreeBuilder](#)

The Blazor [RenderTreeBuilder](#).

Remarks

This component renders no visible DOM; it only provides a style scope to its children.

BuildStyle(StyleBuilder)

Composes child styles and appends them as a single `@media` group to the parent **builder**.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The ambient parent style builder receiving the media group.

Remarks

- Calls [Compose\(\)](#) to gather child contributions.
- Wraps the accumulated fragments in a `@media` group using [ToQuery\(\)](#).
- Clears the inner builder to prevent duplication across compositions.

Class Opacity

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes the CSS `opacity` declaration to the current [StyleBuilder](#).

```
public class Opacity : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← [Opacity](#)

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Basic usage in a styled scope:

```
<Opacity Value="0.5" />
```

Programmatic usage via [Opacity\(StyleBuilder, double?\)](#):

```
builder.Opacity(0.75);
```

Remarks

- Accepts values in the range [0, 1] and clamps out-of-range inputs to this interval.
- When [Value](#) is `null`, [NaN](#), or [IsInfinity\(double\)](#) is true, no declaration is emitted.
- Uses invariant culture formatting with up to three fractional digits.
Participation in style composition, registration, and conditional contribution is managed by [RTBStyleBase](#).

Properties

Value

The desired opacity value in the range [0, 1].

```
[Parameter]
public double? Value { get; set; }
```

Property Value

[double](#)?

Remarks

Values are clamped to [0, 1]. When `null`, no declaration is emitted.

Non-finite values ([NaN](#) or infinity) are ignored.

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The builder receiving the `opacity` declaration.

Remarks

Emits `opacity: <value>` using invariant culture with up to three fractional digits when the value is valid.

Class OpacityExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension helpers for configuring `opacity` on a [StyleBuilder](#).

```
public static class OpacityExtensions
```

Inheritance

[object](#) ← OpacityExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Opacity(StyleBuilder, double?)

Sets the CSS `opacity` declaration on the `builder`.

```
public static StyleBuilder Opacity(this StyleBuilder builder, double? value)
```

Parameters

`builder` [StyleBuilder](#)

The style builder to receive the declaration.

`value` [double](#)?

The desired opacity in the range `[0, 1]`. Values are clamped. When `null`, the builder is returned unchanged.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

Remarks

Uses invariant culture with up to three fractional digits.

Class Other

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

A generic style component that writes an arbitrary CSS declaration to the current style scope.

```
public class Other : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Other

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Component usage:

```
@* Emits: gap: 1rem; *@
```

```
@* Emits a CSS custom property *@
```

Imperative builder usage (equivalent behavior):

```
builder.Other("gap", "1rem");
builder.Other("--card-radius", "12px");
```

Remarks

- [Property](#) is required and should be a valid CSS property name (e.g., "gap", "background-color", "--my-var").
- When [Value](#) is null or whitespace, nothing is emitted.
- Whitespace-only or null property names are ignored.
- This component contributes to a cascading [StyleBuilder](#) provided by an ancestor style root.

Properties

Property

The CSS property name to emit (e.g., "gap", "background-color", "--my-var").

[Parameter]
[EditorRequired]
`public string Property { get; set; }`

Property Value

[string](#)

Remarks

Must not be null or whitespace. Marked as [EditorRequiredAttribute](#).

Value

The CSS value to assign to [Property](#) (e.g., "1rem", "#333", "var(--x)").

[Parameter]
`public string? Value { get; set; }`

Property Value

[string](#)

Remarks

If null or whitespace, no declaration is emitted.

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

Contributes a single CSS declaration if both [Property](#) and [Value](#) are set.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The style builder receiving the declaration.

Class OtherExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension helpers for emitting arbitrary CSS declarations using [StyleBuilder](#).

```
public static class OtherExtensions
```

Inheritance

[object](#) ← OtherExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Other(StyleBuilder, string, string?)

Adds a custom CSS property and value to the style builder.

```
public static StyleBuilder Other(this StyleBuilder builder, string property, string? value)
```

Parameters

builder [StyleBuilder](#)

The target [StyleBuilder](#).

property [string](#)

The CSS property name (e.g., "gap", "--my-var"). Ignored when null or whitespace.

value [string](#)

The CSS value to assign. Ignored when null or whitespace.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Examples

```
var css = StyleBuilder.Start  
    .Other("gap", "1rem")  
    .Other("--ring-color", "hsl(210 100% 50%)");
```

Remarks

No declaration is added when either `property` or `value` is null or whitespace.

Class Overflow

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

A style contributor that sets CSS overflow properties. Behavior:

- When all parameters ([X](#), [Y](#), [Value](#)) are null, this emits 'overflow: auto'.
- Otherwise, it emits any provided axis-specific values (overflow-x, overflow-y) and/or the shorthand (overflow). Note: If both axis-specific values and the shorthand are supplied, the emitted order is: overflow-x, overflow-y, overflow (last). In CSS, the shorthand can reset axis-specific values. Prefer setting either axis-specific values or the shorthand to avoid unintended overrides.

```
public class Overflow : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Overflow

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

Value

Sets the 'overflow' shorthand (both axes).

[Parameter]

```
public Overflow.OverflowMode? Value { get; set; }
```

Property Value

[Overflow.OverflowMode?](#)

Remarks

When null, no 'overflow' shorthand is emitted unless both [X](#) and [Y](#) are also null, in which case the component defaults to 'overflow: auto'.

X

Sets 'overflow-x' for the horizontal axis.

[Parameter]

```
public Overflow.OverflowMode? X { get; set; }
```

Property Value

[Overflow.OverflowMode?](#)

Remarks

When null, no 'overflow-x' declaration is emitted.

Y

Sets 'overflow-y' for the vertical axis.

[Parameter]

```
public Overflow.OverflowMode? Y { get; set; }
```

Property Value

[Overflow.OverflowMode?](#)

Remarks

When null, no 'overflow-y' declaration is emitted.

Methods

BuildStyle(StyleBuilder)

Contributes CSS declarations for overflow properties.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The style builder that receives declarations.

Remarks

- If all properties are null, emits 'overflow: auto'.
- Otherwise emits 'overflow-x' and/or 'overflow-y' for provided axes, and 'overflow' for [Value](#). The shorthand is emitted last and may override axis-specific values per CSS rules.

Enum Overflow.OverflowMode

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Represents allowed values for CSS overflow properties.

```
public enum Overflow.OverflowMode
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Auto = 3

Maps to 'auto'. Content is clipped and scrollbars appear when needed.

Hidden = 1

Maps to 'hidden'. Content is clipped, no scrollbars.

Scroll = 2

Maps to 'scroll'. Content is clipped and scrollbars are always shown.

Visible = 0

Maps to 'visible'. Content is not clipped, may render outside the element's box.

Class OverflowExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods on [StyleBuilder](#) for setting overflow-related CSS.

```
public static class OverflowExtensions
```

Inheritance

[object](#) ← OverflowExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Overflow(StyleBuilder, OverflowMode?)

Sets the 'overflow' CSS shorthand.

```
public static StyleBuilder Overflow(this StyleBuilder builder, Overflow.OverflowMode? value = null)
```

Parameters

builder [StyleBuilder](#)

The style builder.

value [Overflow.OverflowMode?](#)

The overflow mode; when null, no declaration is emitted.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

OverflowX(StyleBuilder, OverflowMode?)

Sets the 'overflow-x' CSS property (horizontal axis).

```
public static StyleBuilder OverflowX(this StyleBuilder builder, Overflow.OverflowMode? value = null)
```

Parameters

builder [StyleBuilder](#)

The style builder.

value [Overflow.OverflowMode?](#)

The overflow mode; when null, no declaration is emitted.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

OverflowY(StyleBuilder, OverflowMode?)

Sets the 'overflow-y' CSS property (vertical axis).

```
public static StyleBuilder OverflowY(this StyleBuilder builder, Overflow.OverflowMode? value = null)
```

Parameters

builder [StyleBuilder](#)

The style builder.

value [Overflow.OverflowMode?](#)

The overflow mode; when null, no declaration is emitted.

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) for chaining.

Class Padding

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS padding declarations to a cascading [StyleBuilder](#). Supports:

- A single value for all sides ([All](#)).
- Axis shorthands ([Vertical](#) for top/bottom and [Horizontal](#) for left/right).
- Individual sides ([Top](#), [Right](#), [Bottom](#), [Left](#)).

Precedence (from lowest to highest): 1) [All](#) (padding) 2) Axis shorthand via [Vertical](#)/[Horizontal](#) (padding: vertical horizontal) 3) Individual sides (padding-top/right/bottom/left) Later declarations override earlier ones when present.

Notes: - When only one axis is provided, the missing axis defaults to 0 (CSS zero). - Two-value padding shorthand uses the CSS form "padding: <vertical> <horizontal>", i.e., "top/bottom left/right".

```
public class Padding : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Padding

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

All

Shorthand padding for all sides (emits "padding: {value}"). Overridden by axis shorthand and any side-specific values when present.

[Parameter]

```
public Spacing? All { get; set; }
```

Property Value

[Spacing?](#)

Bottom

Sets "padding-bottom: {value}". Overrides [All](#) and [Vertical](#) for the bottom side.

[Parameter]

```
public Spacing? Bottom { get; set; }
```

Property Value

[Spacing?](#)

Horizontal

Axis shorthand for left and right. When used (alone or with [Vertical](#)), emits a two-value padding shorthand: "padding: <vertical> <horizontal>" where <horizontal> sets left/right. Missing axis defaults to 0.

[Parameter]

```
public Spacing? Horizontal { get; set; }
```

Property Value

[Spacing?](#)

Left

Sets "padding-left: {value}". Overrides [All](#) and [Horizontal](#) for the left side.

```
[Parameter]
public Spacing? Left { get; set; }
```

Property Value

[Spacing?](#)

Right

Sets "padding-right: {value}". Overrides [All](#) and [Horizontal](#) for the right side.

```
[Parameter]
public Spacing? Right { get; set; }
```

Property Value

[Spacing?](#)

Top

Sets "padding-top: {value}". Overrides [All](#) and [Vertical](#) for the top side.

```
[Parameter]
public Spacing? Top { get; set; }
```

Property Value

[Spacing?](#)

Vertical

Axis shorthand for top and bottom. When used (alone or with [Horizontal](#)), emits a two-value padding shorthand: "padding: <vertical> <horizontal>" where <vertical> sets top/bottom. Missing axis defaults

to 0.

```
[Parameter]  
public Spacing? Vertical { get; set; }
```

Property Value

[Spacing?](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

Emission order establishes precedence: 1) "padding" via [All](#) 2) Two-value "padding: vertical horizontal" via [Vertical/Horizontal](#) 3) Side-specific properties via [Top](#), [Right](#), [Bottom](#), [Left](#)

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

[builder](#) [StyleBuilder](#)

The style builder receiving padding declarations.

Class PaddingExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods on [StyleBuilder](#) for emitting padding declarations.

```
public static class PaddingExtensions
```

Inheritance

[object](#) ← PaddingExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

PaddingAll(StyleBuilder, Spacing?)

Emits "padding: {value}" when **value** is not null.

```
public static StyleBuilder PaddingAll(this StyleBuilder builder, Spacing? value)
```

Parameters

builder [StyleBuilder](#)

The target style builder.

value [Spacing?](#)

Spacing for all sides.

Returns

[StyleBuilder](#)

The same builder for chaining.

PaddingBottom(StyleBuilder, Spacing?)

Emits "padding-bottom: {value}" when `value` is not null.

```
public static StyleBuilder PaddingBottom(this StyleBuilder builder, Spacing? value)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`value` [Spacing?](#)

Bottom spacing.

Returns

[StyleBuilder](#)

The same builder for chaining.

PaddingLeft(StyleBuilder, Spacing?)

Emits "padding-left: {value}" when `value` is not null.

```
public static StyleBuilder PaddingLeft(this StyleBuilder builder, Spacing? value)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`value` [Spacing?](#)

Left spacing.

Returns

[StyleBuilder](#)

The same builder for chaining.

PaddingRight(StyleBuilder, Spacing?)

Emits "padding-right: {value}" when `value` is not null.

```
public static StyleBuilder PaddingRight(this StyleBuilder builder, Spacing? value)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`value` [Spacing?](#)

Right spacing.

Returns

[StyleBuilder](#)

The same builder for chaining.

PaddingTop(StyleBuilder, Spacing?)

Emits "padding-top: {value}" when `value` is not null.

```
public static StyleBuilder PaddingTop(this StyleBuilder builder, Spacing? value)
```

Parameters

`builder` [StyleBuilder](#)

The target style builder.

`value` [Spacing?](#)

Top spacing.

Returns

[StyleBuilder](#)

The same builder for chaining.

Class Positioned

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

A Blazor style contributor that emits CSS positioning declarations.

```
public class Positioned : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Positioned

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

This component contributes the following declarations into the current [StyleBuilder](#) scope: - Always emits `position: {mode}`. - Emits `top`, `right`, `bottom`, `left` only when their values are not null.

Place this inside a style scope that provides a cascading [StyleBuilder](#) (via [StyleBuilder](#)).

Razor usage:

Fluent usage:

```
var css = StyleBuilder.Start
    .Positioned(Positioned.PositionMode.Fixed, top: Size.Percent(10), right: Size.Px(16))
    .BuildScoped("my-class");
```

Properties

Bottom

CSS `bottom` offset. Emitted only when not null.

```
[Parameter]
public SizeExpression? Bottom { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Use [SizeExpression](#) helpers to create values (e.g., pixels, rem, percentages).

Left

CSS `left` offset. Emitted only when not null.

```
[Parameter]
public SizeExpression? Left { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Use [SizeExpression](#) helpers to create values (e.g., pixels, rem, percentages).

Position

CSS `position` mode. Defaults to [Absolute](#).

```
[Parameter]
public Positioned.PositionMode Position { get; set; }
```

Property Value

[Positioned.PositionMode](#)

Remarks

The chosen mode is always emitted; offsets are included only when provided.

Right

CSS `right` offset. Emitted only when not null.

```
[Parameter]
public SizeExpression? Right { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Use [SizeExpression](#) helpers to create values (e.g., pixels, rem, percentages).

Top

CSS `top` offset. Emitted only when not null.

```
[Parameter]
public SizeExpression? Top { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Use [SizeExpression](#) helpers to create values (e.g., pixels, rem, percentages).

Methods

BuildStyle(StyleBuilder)

Builds the component's style contribution by setting the positioning and optional offsets.

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The style builder receiving declarations.

Enum Positioned.PositionMode

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

The CSS positioning mode to emit.

```
public enum Positioned.PositionMode
```

Extension Methods

[CssEnumExtensions.ToCss\(Enumerable\)](#)

Fields

Absolute = 0

Emits `position: absolute.`

Fixed = 2

Emits `position: fixed.`

Relative = 1

Emits `position: relative.`

Sticky = 3

Emits `position: sticky.`

Class PositionedExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods for [StyleBuilder](#) related to CSS positioning.

```
public static class PositionedExtensions
```

Inheritance

[object](#) ← PositionedExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Examples

```
var css = StyleBuilder.Start
    .Positioned(Positioned.PositionMode.Relative, top: Size.Px(8))
    .BuildScoped("my-class");
```

Remarks

Designed for fluent usage. Null offsets are ignored. The [position](#) declaration is always emitted.

Methods

**Positioned(StyleBuilder, PositionMode, SizeExpression?,
SizeExpression?, SizeExpression?, SizeExpression?)**

Adds CSS [position](#) and optional offset declarations to the base declaration set.

```
public static StyleBuilder Positioned(this StyleBuilder builder, Positioned.PositionMode
position = PositionMode.Absolute, SizeExpression? top = null, SizeExpression? right = null,
SizeExpression? bottom = null, SizeExpression? left = null)
```

Parameters

`builder` [StyleBuilder](#)

The style builder to mutate.

`position` [Positioned.PositionMode](#)

The [Positioned.PositionMode](#) to emit; defaults to `absolute`.

`top` [SizeExpression](#)

Optional CSS `top` offset.

`right` [SizeExpression](#)

Optional CSS `right` offset.

`bottom` [SizeExpression](#)

Optional CSS `bottom` offset.

`left` [SizeExpression](#)

Optional CSS `left` offset.

Returns

[StyleBuilder](#)

The same `builder` to allow fluent chaining.

Examples

`StyleBuilder.Start`

```
.Positioned(Positioned.PositionMode.Sticky, top: Size.Rem(2))
.Selector("& > .badge", b => b.Set("z-index", "10"));
```

Class RTBStyleBase

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Base component for contributing styles to a [StyleBuilder](#) within a Blazor render tree.

```
public abstract class RTBStyleBase : ComponentBase, IComponent, IHandleEvent,  
IHandleAfterRender, IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← RTBStyleBase

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Derived

[Animation](#), [Background](#), [Border](#), [Color](#), [Flex](#), [Grid](#), [GridPlacement](#), [Keyframe](#), [Margin](#), [Media](#), [Opacity](#), [Other](#), [Overflow](#), [Padding](#), [Positioned](#), [Selector](#), [Size](#), [Transform](#), [Transition](#), [Visibility](#), [PreStyled](#)

Inherited Members

[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

RTBStyleBase participates in style composition via a cascading [StyleBuilder](#) parameter. When [Condition](#) is true, the instance auto-registers itself with the builder during initialization and parameter updates, and unregisters when the condition becomes false or the component is disposed.

Inheritors implement [BuildStyle\(StyleBuilder\)](#) to describe CSS declarations, selectors, groups, and keyframes. The contribution is gated by [Condition](#) and invoked by the builder during composition.

Properties

Condition

Controls whether this component is registered with the [StyleBuilder](#) and contributes styles.

```
[Parameter]  
public bool Condition { get; set; }
```

Property Value

[bool](#)

Remarks

- When set to true (default), the instance registers with the builder and participates in composition.
- When set to false, the instance unregisters and contributes nothing.
Toggling this parameter at runtime updates the registration accordingly.

StyleBuilder

The cascading style builder into which this component contributes styles.

```
[CascadingParameter(Name = "StyleBuilder")]  
public required StyleBuilder StyleBuilder { get; set; }
```

Property Value

[StyleBuilder](#)

Remarks

This value must be provided by an ancestor component that establishes a style scope (e.g., a style root). The registration lifecycle is managed automatically based on [Condition](#).

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected abstract void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The builder to receive declarations, selectors, groups, and keyframes.

Remarks

This method is called by the infrastructure when [Contribute\(StyleBuilder\)](#) is invoked by the owning [StyleBuilder](#) during composition, provided [Condition](#) is true. Typical usage:

- Use `builder.Set` to add base declarations.
- Use `builder.Selector` to add nested selector rules.
- Use `builder.Media`, `Supports`, or `Container` for group rules.
- Use `builder.Keyframes` to define animations.

Avoid long-running or stateful operations; this method may be invoked multiple times during recomposition.

DisposeAsync()

Unregisters this contributor from the [StyleBuilder](#) and suppresses finalization.

```
public ValueTask DisposeAsync()
```

Returns

[ValueTask](#)

Remarks

Safe to call multiple times. After disposal, no further contributions will occur.

OnInitialized()

Initializes the component and ensures registration state reflects the current [Condition](#).

```
protected override void OnInitialized()
```

OnParametersSet()

Ensures registration state reflects the current [Condition](#) whenever parameters change.

```
protected override void OnParametersSet()
```

Class Selector

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Blazor component that scopes child style contributions under a specific CSS selector/query.

```
public class Selector : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Selector

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

This component collects child styles into a private [StyleBuilder](#) instance and, during [BuildStyle\(StyleBuilder\)](#), appends them to the parent builder as a selector rule via [Selector\(string, Action<StyleBuilder>\)](#).

The [Query](#) supports the `&` placeholder to reference the current scope. When [Query](#) is null or whitespace, it implicitly becomes `&`, meaning "use the current scope as-is". Examples (assuming current scope ".root"):

- `Query="&:hover"` results in selector ".root:hover".
- `Query=".child"` results in selector ".root .child".
- `Query=""` (empty) results in selector ".root".
- `Query=".a, .b"` emits for both ".root .a" and ".root .b".

Lifecycle:

1. Child components contribute styles into the cascaded inner [StyleBuilder](#).
2. On composition, [BuildStyle\(StyleBuilder\)](#) calls `_inner.Compose()` to gather children.
3. The collected styles are appended to the parent via `builder.Selector(Query, sb => sb.Absorb(_inner))`.
4. `_inner.ClearAll()` resets the inner builder to avoid leaking state across compositions.

```
<Styled>
  <Selector Query="&:hover">
    <Set Prop="color" Value="red" />
  </Selector>
  <Selector Query=".title, .subtitle">
    <Set Prop="font-weight" Value="600" />
  </Selector>
</Styled>
```

[StyleBuilder Selector\(string, Action<StyleBuilder>\)](#) [RTBStyleBase SelectorRule](#)

Properties

ChildContent

The child content that contributes style declarations and fragments to this selector's inner builder.

```
[Parameter]
[EditorRequired]
public required RenderFragment ChildContent { get; set; }
```

Property Value

[RenderFragment](#) ↗

Remarks

The content receives a cascaded [StyleBuilder](#) instance (private to this component) so that any nested style components write into the selector-scoped builder instead of the parent.

Query

The CSS selector/query to scope the child styles under.

[Parameter]
`public string Query { get; set; }`

Property Value

[string](#)

Remarks

- Supports the `&` placeholder for the current scope, primarily when the selector starts with `&`.
- When null or whitespace, defaults to `&`, i.e., "use the current scope".
- Supports comma-delimited lists (e.g., `.a, .b`).

Methods

BuildRenderTree(RenderTreeBuilder)

Renders a fixed [`CascadingValue< TValue >`](#) that supplies the inner [`StyleBuilder`](#) to [`ChildContent`](#) so nested style components write into this selector's scope.

`protected override void BuildRenderTree(RenderTreeBuilder builder)`

Parameters

`builder` [`RenderTreeBuilder`](#)

The render tree builder.

Remarks

Uses `IsFixed=true` to avoid re-rendering the cascade reference when not necessary.

BuildStyle(StyleBuilder)

Contributes this component's styles to the provided `builder`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The builder to receive declarations, selectors, groups, and keyframes.

Remarks

- Resolves [Query](#) (defaults to `&` when empty).
- Composes child contributions into the inner builder.
- Appends them to the parent under the resolved selector via [Selector\(string, Action<StyleBuilder>\)](#).
- Clears the inner builder to prevent state leakage.

Class Size

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Contributes CSS size-related declarations (width/height and their min/max variants) to the current [Style Builder](#).

```
public class Size : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Size

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

The component can be used inside a context where a [StyleBuilder](#) is cascading:

```
<!-- In a Razor file where StyleBuilder is provided -->  
<Size FullWidth="true" MinHeight="someSizeExpression" />
```

Or programmatically through the extension methods:

```
builder.Width(value: widthExpr, min: minWidthExpr, max: maxWidthExpr)
    .Height(value: heightExpr, min: minHeightExpr, max: maxHeightExpr);
```

Remarks

- If [FullWidth](#) is true, a base declaration `width: 100%` is emitted. If [Width](#) is also provided, it will override the earlier 100% declaration, because it is applied afterward.
- The same precedence applies to [FullHeight](#) and [Height](#).
- Use [SizeExpression](#) values to ensure valid CSS (e.g., px, rem, %, vw, vh).

Properties

FullHeight

If true, emits `height: 100%` before applying [Height](#). If [Height](#) is set, it will override this declaration.

```
[Parameter]
public bool FullHeight { get; set; }
```

Property Value

[bool](#) ↗

FullWidth

If true, emits `width: 100%` before applying [Width](#). If [Width](#) is set, it will override this declaration.

```
[Parameter]
public bool FullWidth { get; set; }
```

Property Value

[bool](#) ↗

Height

The preferred height as a [SizeExpression](#).

```
[Parameter]
public SizeExpression? Height { get; set; }
```

Property Value

[SizeExpression](#)

MaxHeight

The maximum height as a [SizeExpression](#).

```
[Parameter]
public SizeExpression? MaxHeight { get; set; }
```

Property Value

[SizeExpression](#)

MaxWidth

The maximum width as a [SizeExpression](#).

```
[Parameter]
public SizeExpression? MaxWidth { get; set; }
```

Property Value

[SizeExpression](#)

MinHeight

The minimum height as a [SizeExpression](#).

```
[Parameter]
public SizeExpression? MinHeight { get; set; }
```

Property Value

[SizeExpression](#)

MinWidth

The minimum width as a [SizeExpression](#).

[Parameter]

```
public SizeExpression? MinWidth { get; set; }
```

Property Value

[SizeExpression](#)

Width

The preferred width as a [SizeExpression](#) (e.g., 100px, 10rem, 50%).

[Parameter]

```
public SizeExpression? Width { get; set; }
```

Property Value

[SizeExpression](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

Applies size-related CSS declarations to the provided `builder`.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

`builder` [StyleBuilder](#)

The [StyleBuilder](#) to receive the CSS declarations.

Remarks

Order of application:

1. If [FullWidth](#) is true, set `width: 100%`.
2. Apply [Width](#), [MinWidth](#), [MaxWidth](#).
3. If [FullHeight](#) is true, set `height: 100%`.
4. Apply [Height](#), [MinHeight](#), [MaxHeight](#). Later declarations override earlier ones for the same property.

Class SizeExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extension methods for [StyleBuilder](#) to compose size-related declarations.

```
public static class SizeExtensions
```

Inheritance

[object](#) ← SizeExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Height(StyleBuilder, SizeExpression?, SizeExpression?, SizeExpression?)

Set height-related declarations on the [StyleBuilder](#).

```
public static StyleBuilder Height(this StyleBuilder builder, SizeExpression? value = null,  
SizeExpression? min = null, SizeExpression? max = null)
```

Parameters

builder [StyleBuilder](#)

The builder that receives the declarations.

value [SizeExpression](#)

Optional height value (maps to `height`).

min [SizeExpression](#)

Optional minimum height (maps to `min-height`).

`max` [SizeExpression](#)

Optional maximum height (maps to `max-height`).

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Width(StyleBuilder, SizeExpression?, SizeExpression?, SizeExpression?)

Set width-related declarations on the [StyleBuilder](#).

```
public static StyleBuilder Width(this StyleBuilder builder, SizeExpression? value = null,  
SizeExpression? min = null, SizeExpression? max = null)
```

Parameters

`builder` [StyleBuilder](#)

The builder that receives the declarations.

`value` [SizeExpression](#)

Optional width value (maps to `width`).

`min` [SizeExpression](#)

Optional minimum width (maps to `min-width`).

`max` [SizeExpression](#)

Optional maximum width (maps to `max-width`).

Returns

[StyleBuilder](#)

The same [StyleBuilder](#) instance for chaining.

Class Styled

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

A component that provides a scoped CSS class and style builder context to its children.

```
public class Styled : ComponentBase, IComponent, IHandleEvent, IHandleAfterRender
```

Inheritance

[object](#) ← [ComponentBase](#) ← [Styled](#)

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#)

Inherited Members

[ComponentBase.OnInitialized\(\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSet\(\)](#), [ComponentBase.OnParametersSetAsync\(\)](#),
[ComponentBase.StateHasChanged\(\)](#), [ComponentBase.ShouldRender\(\)](#),
[ComponentBase.OnAfterRender\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

ChildContent

Child content that receives the resolved CSS class as a parameter.

```
[Parameter]  
public RenderFragment<string>? ChildContent { get; set; }
```

Property Value

[RenderFragment](#)<[string](#)>

Class

Optional additional class name(s) to append to the resolved class.

```
[Parameter]
public string? Class { get; set; }
```

Property Value

[string](#)

Classname

Optional externally provided class name to use instead of generating a new one.

```
[Parameter]
public string? Classname { get; set; }
```

Property Value

[string](#)

ClassnameChanged

Event callback that is invoked when the resolved class name changes.

```
[Parameter]
public EventCallback<string?> ClassnameChanged { get; set; }
```

Property Value

[EventCallback](#) <[string](#)>

Configure

An optional action to configure the StyleBuilder used by this component.

```
[Parameter]
public Action<StyleBuilder>? Configure { get; set; }
```

Property Value

[Action](#) <[StyleBuilder](#)>

Methods

BuildRenderTree(RenderTreeBuilder)

Render a CascadingValue that provides the StyleBuilder to descendants,

```
protected override void BuildRenderTree(RenderTreeBuilder builder)
```

Parameters

builder [RenderTreeBuilder](#)

DisposeAsync()

Dispose the component by releasing the acquired class from the registry.

```
public ValueTask DisposeAsync()
```

Returns

[ValueTask](#)

OnAfterRenderAsync(bool)

After the component has rendered, configure the StyleBuilder, build the scoped CSS,

```
protected override Task OnAfterRenderAsync(bool firstRender)
```

Parameters

firstRender [bool](#) ↗

Returns

[Task](#) ↗

Class Transform

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Modern transform component: compose transforms via Parts and set optional Origin.

```
public class Transform : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Transform

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

Origin

Optional "transform-origin": e.g. "center", "left top", "20px 50%".

```
[Parameter]  
public string? Origin { get; set; }
```

Property Value

[string](#)

Parts

List of transform parts like "translateY(8px)", "rotate(3deg)". Joined by spaces.

```
[Parameter]
public IEnumerable<string>? Parts { get; set; }
```

Property Value

[IEnumerable](#) <[string](#)>

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

PartPerspective(SizeUnit)

Creates a perspective part. E.g. "perspective(500px)".

```
public static string PartPerspective(SizeUnit d)
```

Parameters

d [SizeUnit](#)

Returns

[string ↗](#)

PartRotate(double)

Creates a rotate part. E.g. "rotate(30deg)".

```
public static string PartRotate(double deg)
```

Parameters

[deg double ↗](#)

Returns

[string ↗](#)

PartRotate3D(double, double, double, double)

Creates a rotate3d part. E.g. "rotate3d(1, 0, 0, 30deg)" to rotate 30 degrees around the X axis.

```
public static string PartRotate3D(double x, double y, double z, double deg)
```

Parameters

[x double ↗](#)

[y double ↗](#)

[z double ↗](#)

[deg double ↗](#)

Returns

[string ↗](#)

PartRotateX(double)

Creates a rotateX/Y/Z part. E.g. "rotateX(30deg)".

```
public static string PartRotateX(double deg)
```

Parameters

deg [double](#)

Returns

[string](#)

PartRotateY(double)

Creates a rotateY part. E.g. "rotateY(30deg)".

```
public static string PartRotateY(double deg)
```

Parameters

deg [double](#)

Returns

[string](#)

PartRotateZ(double)

Creates a rotateZ part. E.g. "rotateZ(30deg)".

```
public static string PartRotateZ(double deg)
```

Parameters

deg [double](#)

Returns

[string](#)

PartScale(double)

Creates a scale part. E.g. "scale(1.5)" or "scale(1.5, 2)".

```
public static string PartScale(double s)
```

Parameters

s [double](#)

Returns

[string](#)

PartScale(double, double)

Creates a scale part with separate X and Y factors. E.g. "scale(1.5, 2)".

```
public static string PartScale(double sx, double sy)
```

Parameters

sx [double](#)

sy [double](#)

Returns

[string](#)

PartScale3D(double, double, double)

Creates a scale3d part. E.g. "scale3d(1.5, 2, 1)".

```
public static string PartScale3D(double sx, double sy, double sz)
```

Parameters

sx [double](#)

sy [double](#)

sz [double](#)

Returns

[string](#)

PartScaleX(double)

Creates a scaleX part. E.g. "scaleX(1.5)".

```
public static string PartScaleX(double sx)
```

Parameters

sx [double](#)

Returns

[string](#)

PartScaleY(double)

Creates a scaleY part. E.g. "scaleY(2)".

```
public static string PartScaleY(double sy)
```

Parameters

sy [double](#)

Returns

[string ↗](#)

PartScaleZ(double)

Creates a scaleZ part. E.g. "scaleZ(1.2)".

```
public static string PartScaleZ(double sz)
```

Parameters

[sz double ↗](#)

Returns

[string ↗](#)

PartSkew(double, double)

Creates a skew part. E.g. "skew(10deg, 20deg)".

```
public static string PartSkew(double xDeg, double yDeg)
```

Parameters

[xDeg double ↗](#)

[yDeg double ↗](#)

Returns

[string ↗](#)

PartSkewX(double)

Creates a skewX part. E.g. "skewX(10deg)".

```
public static string PartSkewX(double deg)
```

Parameters

deg [double](#)

Returns

[string](#)

PartSkewY(double)

Creates a skewY part. E.g. "skewY(20deg)".

```
public static string PartSkewY(double deg)
```

Parameters

deg [double](#)

Returns

[string](#)

PartTranslate(SizeUnit, SizeUnit)

Creates a translate part. E.g. "translate(10px, 20px)".

```
public static string PartTranslate(SizeUnit x, SizeUnit y)
```

Parameters

x [SizeUnit](#)

y [SizeUnit](#)

Returns

[string ↗](#)

PartTranslate3D(SizeUnit, SizeUnit, SizeUnit)

Creates a translate3d part. E.g. "translate3d(10px, 20px, 30px)".

```
public static string PartTranslate3D(SizeUnit x, SizeUnit y, SizeUnit z)
```

Parameters

x [SizeUnit](#)

y [SizeUnit](#)

z [SizeUnit](#)

Returns

[string ↗](#)

PartTranslateX(SizeUnit)

Creates a translateX part. E.g. "translateX(10px)".

```
public static string PartTranslateX(SizeUnit x)
```

Parameters

x [SizeUnit](#)

Returns

[string ↗](#)

PartTranslateY(SizeUnit)

Creates a translateY part. E.g. "translateY(20px)".

```
public static string PartTranslateY(SizeUnit y)
```

Parameters

y [SizeUnit](#)

Returns

[string](#)

PartTranslateZ(SizeUnit)

Creates a translateZ part. E.g. "translateZ(30px)".

```
public static string PartTranslateZ(SizeUnit z)
```

Parameters

z [SizeUnit](#)

Returns

[string](#)

Class TransformExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Extensions for fluent StyleBuilder usage.

```
public static class TransformExtensions
```

Inheritance

[object](#) ← TransformExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Transform(StyleBuilder, params string[])

Sets the 'transform' property by joining the given parts with spaces.

```
public static StyleBuilder Transform(this StyleBuilder b, params string[] parts)
```

Parameters

b [StyleBuilder](#)

parts [string](#)[]

Returns

[StyleBuilder](#)

TransformNone(StyleBuilder)

Sets 'transform: none;' to reset any transforms.

```
public static StyleBuilder TransformNone(this StyleBuilder b)
```

Parameters

b [StyleBuilder](#)

Returns

[StyleBuilder](#)

TransformOrigin(StyleBuilder, string?)

Sets the 'transform-origin' property.

```
public static StyleBuilder TransformOrigin(this StyleBuilder b, string? origin)
```

Parameters

b [StyleBuilder](#)

origin [string](#) ↗

Returns

[StyleBuilder](#)

Class Transition

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Shorthand-first transitions with typed times and multi-item support.

```
public class Transition : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← Transition

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

Behavior

Optional: "normal" | "allow-discrete" (Transitions Level 2). Single value or comma list.

```
[Parameter]  
public string? Behavior { get; set; }
```

Property Value

[string](#) ↗

Delay

E.g. `TimeSpan.FromSeconds(0.3)` or `TimeSpan.FromMilliseconds(150)`. Default is 0 (no delay).

```
[Parameter]  
public TimeSpan Delay { get; set; }
```

Property Value

[TimeSpan](#) ↗

Duration

E.g. `TimeSpan.FromSeconds(0.3)` or `TimeSpan.FromMilliseconds(150)`. Default is 0 (no transition).

```
[Parameter]  
public TimeSpan Duration { get; set; }
```

Property Value

[TimeSpan](#) ↗

Items

When set, emits "transition: ..., ..." from these items.

```
[Parameter]  
public IEnumerable<TransitionItem>? Items { get; set; }
```

Property Value

[IEnumerable](#) ↗ <[TransitionItem](#)>

Property

Convenience single transition (emits shorthand). Ignored if [Items](#) is supplied.

```
[Parameter]
public string Property { get; set; }
```

Property Value

[string](#) ↗

TimingFunction

E.g. "ease", "linear", "ease-in-out", "cubic-bezier()", "steps()"

```
[Parameter]
public string TimingFunction { get; set; }
```

Property Value

[string](#) ↗

WillChange

Optional global "will-change: ..." hint.

```
[Parameter]
public string? WillChange { get; set; }
```

Property Value

[string](#) ↗

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

Class TransitionExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Style builder extensions for transitions.

```
public static class TransitionExtensions
```

Inheritance

[object](#) ← TransitionExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Transition(StyleBuilder, params TransitionItem[])

Sets a transition on an element, with support for multiple items.

```
public static StyleBuilder Transition(this StyleBuilder b, params TransitionItem[] items)
```

Parameters

b [StyleBuilder](#)

items [TransitionItem](#)[]

Returns

[StyleBuilder](#)

WillChange(StyleBuilder, string)

Sets a single transition on an element, using shorthand properties.

```
public static StyleBuilder WillChange(this StyleBuilder b, string value)
```

Parameters

b [StyleBuilder](#)

value [string](#) ↗

Returns

[StyleBuilder](#)

Class TransitionItem

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Single transition item for use in [Items](#).

```
public sealed class TransitionItem
```

Inheritance

[object](#) ← TransitionItem

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Behavior

Optional: "normal" | "allow-discrete" (Transitions Level 2). Single value.

```
public string? Behavior { get; set; }
```

Property Value

[string](#)

Delay

E.g. `TimeSpan.FromSeconds(0.3)` or `TimeSpan.FromMilliseconds(150)`. Default is 0 (no delay).

```
public TimeSpan Delay { get; set; }
```

Property Value

[TimeSpan](#)

Duration

E.g. `TimeSpan.FromSeconds(0.3)` or `TimeSpan.FromMilliseconds(150)`. Default is 0 (no transition).

```
public TimeSpan Duration { get; set; }
```

Property Value

[TimeSpan](#)

Property

E.g. "all", "opacity", "transform", "background-color", etc. Default is "all".

```
public string Property { get; set; }
```

Property Value

[string](#)

TimingFunction

E.g. "ease", "linear", "ease-in-out", "cubic-bezier()", "steps()". Default is "ease".

```
public string TimingFunction { get; set; }
```

Property Value

[string](#)

Class Visibility

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Sets the visibility of an element.

```
public class Visibility : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← [Visibility](#)

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Properties

Value

The visibility mode. Default is [Visible](#).

```
[Parameter]  
public Visibility.Mode Value { get; set; }
```

Property Value

[Visibility.Mode](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

Enum Visibility.Mode

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

The visibility mode.

```
public enum Visibility.Mode
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Collapse = 2

The element is hidden and does not take up space.

Hidden = 1

The element is hidden, but still takes up space.

Visible = 0

The element is visible.

Class VisibilityExtensions

Namespace: [RTB.Blazor.Styled.Components](#)

Assembly: RTB.Blazor.Styled.dll

Style builder extensions for visibility.

```
public static class VisibilityExtensions
```

Inheritance

[object](#) ← VisibilityExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Visibility(StyleBuilder, Mode)

Sets the visibility of an element.

```
public static StyleBuilder Visibility(this StyleBuilder builder, Visibility.Mode value)
```

Parameters

builder [StyleBuilder](#)

value [Visibility.Mode](#)

Returns

[StyleBuilder](#)

Namespace RTB.Blazor.Styled.Core

Classes

[DeclarationSet](#)

Represents a mutable set of CSS declarations (property/value pairs) destined for the writer's current selector.

[GroupRule](#)

Represents a CSS grouping at-rule (e.g., `@media`, `@supports`, `@container`, `@layer`) that wraps child [IStyleFragment](#) instances inside a single block.

[KeyframeFrame](#)

Represents a single frame within a `@keyframes` rule.

[Keyframes](#)

Represents a CSS `@keyframes` rule and its frames.

[ScopedWriter](#)

Provides a minimal, allocation-friendly writer for emitting scoped CSS strings. Maintains a selector stack to support nested emission contexts without copying strings.

[SelectorRule](#)

Represents a CSS-like selector rule that emits its own declarations and any child fragments relative to the current selector scope provided by a [ScopedWriter](#).

[StyleBuilder](#)

A builder for CSS styles, supporting base declarations, nested selectors, groups (media/supports/container), and keyframes.

Interfaces

[IStyleContributor](#)

Defines a participant that can contribute CSS to a [StyleBuilder](#) during composition.

[IStyleFragment](#)

Defines a minimal contract for a style fragment capable of emitting CSS into a [ScopedWriter](#).

Class DeclarationSet

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Represents a mutable set of CSS declarations (property/value pairs) destined for the writer's current selector.

```
public sealed class DeclarationSet : IStyleFragment, IEnumerable<KeyValuePair<string, string>>, IEnumerable
```

Inheritance

[object](#) ← DeclarationSet

Implements

[IStyleFragment](#), [IEnumerable](#)<[KeyValuePair](#)<[string](#), [string](#)>>, [IEnumerable](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Remarks

- Properties and values are trimmed on insert; null or whitespace-only inputs are ignored.
- Adding the same property more than once overwrites the previous value (last-write-wins).
- [Join\(IEnumerable<KeyValuePair<string, string>>\)](#) merges another sequence using the same add rules.
- [Emit\(ScopedWriter\)](#) writes a rule block for [CurrentSelector](#) and does nothing when empty.
- Emission order follows the dictionary's enumeration order. While .NET currently preserves insertion order for [Dictionary< TKey, TValue >](#), callers should not rely on a specific order.
- No validation or escaping of CSS is performed; callers must provide valid CSS identifiers and values.
- This type is not thread-safe.

Properties

IsEmpty

Gets a value indicating whether the set contains no declarations.

```
public bool IsEmpty { get; }
```

Property Value

[bool ↗](#)

Methods

Add(string, string)

Adds or replaces a CSS declaration.

```
public void Add(string prop, string value)
```

Parameters

prop [string ↗](#)

The CSS property name. Null or whitespace is ignored.

value [string ↗](#)

The CSS value. Null or whitespace is ignored.

Remarks

Both **prop** and **value** are trimmed. If either is null/whitespace, the call is ignored. When the same property is added multiple times, the last value wins.

Clear()

Removes all declarations from the set.

```
public void Clear()
```

Emit(ScopedWriter)

Emits a CSS rule block for the current selector into the provided writer.

```
public void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The writer that receives the CSS output. Must not be null.

Remarks

- No output is written if the set is empty.
- Uses [CurrentSelector](#) for the selector and writes all current declarations as-is.
- This method does not clear the set after emission; callers manage lifecycle as needed.

GetEnumerator()

Returns an enumerator that iterates through the declarations in this set.

```
public IEnumerator<KeyValuePair<string, string>> GetEnumerator()
```

Returns

[IEnumerator](#)<[KeyValuePair](#)<string, string>>

An enumerator over property/value pairs.

Remarks

The enumeration order matches the underlying dictionary's enumeration order.

Join(IEnumerable<KeyValuePair<string, string>>)

Merges a sequence of declarations into this set using last-write-wins semantics.

```
public void Join(IEnumerable<KeyValuePair<string, string>> source)
```

Parameters

source [IEnumerable](#)<[KeyValuePair](#)<string, string>>

The source sequence of property/value pairs. If null, the call is a no-op.

Remarks

Each pair is processed via [Add\(string, string\)](#); null or whitespace keys/values are ignored.

Class GroupRule

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Represents a CSS grouping at-rule (e.g., `@media`, `@supports`, `@container`, `@layer`) that wraps child [IStyleFragment](#) instances inside a single block.

```
public sealed class GroupRule : IStyleFragment
```

Inheritance

[object](#) ← GroupRule

Implements

[IStyleFragment](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- This fragment does not change the current selector scope of the [ScopedWriter](#); it only surrounds the emitted children with `{ ... }` after writing the at-rule header.
- If [Children](#) is empty, [Emit\(ScopedWriter\)](#) writes nothing.
- No validation or escaping is performed for [Kind](#) or [Prelude](#); callers must supply valid CSS.
- Whitespace is intentionally minimal to reduce allocations; a single space is inserted between [Kind](#) and [Prelude](#) only when [Prelude](#) is non-empty/non-whitespace.
- Output shape (when there are children): `{Kind}[{Prelude}]{{...children...}}`

Constructors

GroupRule(string, string)

Initializes a new [GroupRule](#) with the specified at-rule `kind` and `prelude`.

```
public GroupRule(string kind, string prelude)
```

Parameters

kind [string](#)

The at-rule keyword including the leading @ (e.g., "@media").

prelude [string](#)

The text following the at-keyword. May be null; null is treated as [Empty](#).

Properties

Children

Gets the child fragments to be emitted inside the group rule block, in order.

```
public List<IStyleFragment> Children { get; }
```

Property Value

[List](#)<[IStyleFragment](#)>

Remarks

Never null. If the collection is empty, nothing is emitted.

Kind

Gets the at-rule keyword including the leading @ (e.g., "@media", "@supports", "@container", "@layer").

```
public string Kind { get; }
```

Property Value

[string](#)

Remarks

This value is not validated; any string will be written as-is.

Prelude

Gets the at-rule prelude (the text following the at-keyword), such as `screen` and `(min-width: 992px)` for a media query.

```
public string Prelude { get; }
```

Property Value

`string` ↗

Remarks

Never null; if a null value is provided to the constructor it is normalized to [Empty](#). When empty or whitespace, no leading space is written between [Kind](#) and the opening brace.

Methods

Emit(ScopedWriter)

Emits the group rule and its children to the provided [ScopedWriter](#).

```
public void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The writer that accumulates CSS and maintains selector scope. Must be non-null.

Remarks

- If there are no children, the method returns without writing anything.
- Writes the at-rule header ([Kind](#) and optional [Prelude](#)) followed by a brace-enclosed block.
- Each child is emitted in sequence within the same selector scope as the caller.
- No validation or escaping is performed.

Interface IStyleContributor

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Defines a participant that can contribute CSS to a [StyleBuilder](#) during composition.

```
public interface IStyleContributor
```

Remarks

Usage:

- Register an implementation with [StyleBuilder.Register\(IStyleContributor\)](#).
- When [StyleBuilder.Compose\(\)](#) is invoked, the builder calls [Contribute\(StyleBuilder\)](#) on each registered contributor to aggregate base declarations, selectors, groups, and keyframes. Guidance:
- Implementations should be deterministic and idempotent (safe to call multiple times).
- Treat the builder as the sole output channel; avoid external side effects.
- Do not retain references to the provided [StyleBuilder](#) beyond the call.

Methods

Contribute(StyleBuilder)

Contributes style declarations and fragments to the provided [StyleBuilder](#).

```
void Contribute(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The style builder to mutate by adding base declarations, selector rules, group rules, and/or keyframes.
Must not be [null](#).

Exceptions

[ArgumentNullException](#)

Implementations may throw if `builder` is `null`.

Interface IStyleFragment

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Defines a minimal contract for a style fragment capable of emitting CSS into a [ScopedWriter](#).

```
public interface IStyleFragment
```

Remarks

- Fragments should write directly to the provided writer to minimize allocations.
- Use the writer's current selector scope; push nested scopes via [WithSelector\(string, Action\)](#) when needed.
- Do not capture or store the writer beyond the call to [Emit\(ScopedWriter\)](#).
- No validation or escaping is performed; callers should provide valid selectors and declarations.
- Thread-safety is not required; a single writer should not be used concurrently from multiple threads.

Methods

Emit(ScopedWriter)

Emits CSS for this fragment into the provided [ScopedWriter](#).

```
void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The writer that accumulates CSS and manages selector scope.

Remarks

Implementations should:

- Write directly via [Write\(string\)](#), [WriteRuleBlock\(string, IReadOnlyDictionary<string, string>\)](#), and [WriteDeclarations\(IReadOnlyDictionary<string, string>\)](#).
- Restore any selector scope they alter, preferably by using [WithSelector\(string, Action\)](#).

- Treat `w` as non-null; passing null is invalid and may result in an [ArgumentNullException](#).

Class KeyframeFrame

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Represents a single frame within a `@keyframes` rule.

```
public sealed class KeyframeFrame : IStyleFragment
```

Inheritance

[object](#) ← KeyframeFrame

Implements

[IStyleFragment](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

The [Offset](#) should be a valid CSS keyframe selector such as "from", "to", or a percentage like "0%", "50%", "100%". If [Declarations](#) is empty, this frame is not emitted.

Constructors

KeyframeFrame(string)

Creates a new [KeyframeFrame](#) for the specified offset.

```
public KeyframeFrame(string offset)
```

Parameters

offset [string](#)

The keyframe offset (not validated).

Properties

Declarations

The set of CSS declarations for this frame.

```
public DeclarationSet Declarations { get; }
```

Property Value

[DeclarationSet](#)

Remarks

Declarations are emitted in the enumeration order of the underlying set.

Offset

The keyframe offset selector (e.g., "0%", "50%", "100%", "from", or "to").

```
public string Offset { get; }
```

Property Value

[string](#)

Methods

Emit(ScopedWriter)

Emits this frame's declarations to the provided writer.

```
public void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The scoped writer to receive CSS output.

Remarks

Takes a snapshot of [Declarations](#) to avoid issues if the set is modified during iteration.

Class Keyframes

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Represents a CSS `@keyframes` rule and its frames.

```
public sealed class Keyframes : IStyleFragment
```

Inheritance

[object](#) ← Keyframes

Implements

[IStyleFragment](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- Emits compact CSS without validation or escaping.
- Nothing is emitted if [Name](#) is null/whitespace or if there are no frames with declarations.
- The order of frames in [Frames](#) is preserved on emission.

Constructors

Keyframes(string)

Creates a new [Keyframes](#) with the provided animation name.

```
public Keyframes(string name)
```

Parameters

`name` [string](#)

The animation name (CSS identifier). Not validated.

Properties

Frames

The ordered collection of frames contained in this `@keyframes` rule.

```
public List<KeyframeFrame> Frames { get; }
```

Property Value

[List](#) <[KeyframeFrame](#)>

Remarks

No validation is performed on offsets or duplicate entries.

Name

The animation name (CSS identifier) for the `@keyframes` rule.

```
public string Name { get; }
```

Property Value

[string](#)

Methods

Add(KeyframeFrame)

Adds a frame to this `@keyframes` rule.

```
public Keyframes Add(KeyframeFrame frame)
```

Parameters

frame [KeyframeFrame](#)

The frame to add. Ignored if null.

Returns

[Keyframes](#)

The current [Keyframes](#) instance for chaining.

Emit(ScopedWriter)

Emits this `@keyframes` rule and its frames to the provided writer.

```
public void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The scoped writer to receive CSS output.

Remarks

Skips emission if [Name](#) is null/whitespace or [Frames](#) is empty.

Class ScopedWriter

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Provides a minimal, allocation-friendly writer for emitting scoped CSS strings. Maintains a selector stack to support nested emission contexts without copying strings.

```
public sealed class ScopedWriter
```

Inheritance

[object](#) ← ScopedWriter

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- This type does not validate or escape CSS. Callers must provide valid CSS selectors and declarations.
- The root selector is pushed at construction and remains at the base of the stack until disposal/end of usage.
- Designed for high-throughput string building scenarios; all writes append directly to the provided [StringBuilder](#).

Constructors

ScopedWriter(StringBuilder, string)

Initializes a new instance of [ScopedWriter](#) using the provided [StringBuilder](#) and root selector.

```
public ScopedWriter(StringBuilder sb, string rootSelector)
```

Parameters

sb [StringBuilder](#)

The [StringBuilder](#) to which CSS will be appended. Must not be null.

`rootSelector` [string](#)

The root CSS selector to seed the selector stack. Must not be null.

Remarks

The constructor does not write anything immediately; it only establishes the initial scope.

Properties

CurrentSelector

Gets the selector at the top of the selector stack.

```
public string CurrentSelector { get; }
```

Property Value

[string](#)

Exceptions

[InvalidOperationException](#)

Thrown if accessed after the internal stack has been fully unwound (should not occur in normal usage).

Methods

WithSelector(string, Action)

Temporarily pushes a selector onto the stack, executes the provided `emit` action, then restores the previous selector.

```
public void WithSelector(string selector, Action emit)
```

Parameters

`selector` [string](#)

The selector to push for the duration of `emit`.

`emit Action`

An action that writes within the new selector scope.

Remarks

Uses a try/finally to guarantee the selector is popped even if `emit` throws.

Write(string)

Appends the specified string to the underlying [StringBuilder](#) without modification.

```
public void Write(string s)
```

Parameters

`s string`

The string to append.

Remarks

No validation or escaping is performed.

WriteDeclarations(IReadOnlyDictionary<string, string>)

Writes a sequence of CSS declarations in the form `prop:value;` for each pair.

```
public void WriteDeclarations(IReadOnlyDictionary<string, string> decls)
```

Parameters

`decls IReadOnlyDictionary<string, string>`

A read-only dictionary of CSS declarations (property/value pairs).

Remarks

- Declarations are emitted in the dictionary's enumeration order.
- No additional whitespace or escaping is applied.

WriteRuleBlock(string, IReadOnlyDictionary<string, string>)

Writes a CSS rule block of the form: `{selector}{k:v;...}`.

```
public void WriteRuleBlock(string selector, IReadOnlyDictionary<string, string> decls)
```

Parameters

`selector` [string](#)

The CSS selector for the rule block.

`decls` [IReadOnlyDictionary](#)<[string](#), [string](#)>

A read-only dictionary of CSS declarations (property/value pairs).

Remarks

- Keys are written as-is as property names; values are written as-is as property values.
- No whitespace or formatting is added beyond curly braces and semicolons to minimize allocations.

Class SelectorRule

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

Represents a CSS-like selector rule that emits its own declarations and any child fragments relative to the current selector scope provided by a [ScopedWriter](#).

```
public sealed class SelectorRule : IStyleFragment
```

Inheritance

[object](#) ← SelectorRule

Implements

[IStyleFragment](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- Supports comma-delimited selector lists (e.g., ".a, .b"). Each part is processed independently.
- Supports '&' as a placeholder for the current scope, but only when the selector text starts with '&' (e.g., "&:hover", "& > .child"). In that case, all '&' are replaced with the current scope.
- When the selector part is null, empty, or whitespace, the current scope is used as-is.
- A rule block is written only when [Declarations](#) is not empty; however, [Children](#) are still emitted within the resolved scope regardless of whether declarations exist.

Given a current scope ".parent": - Selector ".child" resolves to ".parent .child" - Selector "&:hover" resolves to ".parent:hover" - Selector "& > .child" resolves to ".parent > .child" - Selector "" (empty) resolves to ".parent" - Selector ".a, .b" resolves and emits for ".parent .a" and ".parent .b" independently

Constructors

SelectorRule(string)

Initializes a new instance of the [SelectorRule](#) class.

```
public SelectorRule(string selector)
```

Parameters

`selector` [string](#)

The selector to resolve against the current scope; when null, an empty string is used.

Properties

Children

Nested style fragments that will be emitted within the resolved selector scope.

```
public List<IStyleFragment> Children { get; init; }
```

Property Value

[List](#) <[IStyleFragment](#)>

Remarks

Emitted via [WithSelector\(string, Action\)](#) for each resolved selector part. This occurs even when [Declarations](#) is empty.

Declarations

The declarations to emit for the resolved selector(s).

```
public DeclarationSet Declarations { get; init; }
```

Property Value

[DeclarationSet](#)

Remarks

If [IsEmpty](#) is true, no rule block is written. Child fragments are still emitted within the resolved scope.

Selector

Gets the raw selector text for this rule. May contain a comma-delimited list and/or '&' placeholders.

```
public string Selector { get; }
```

Property Value

[string](#)

The unprocessed selector text. When null is provided to the constructor, this is set to [Empty](#).

Methods

Emit(ScopedWriter)

Emits CSS for this rule:

- Splits the selector into comma-delimited parts.
- Resolves each part against the current scope.
- Writes a rule block when declarations exist.
- Recursively emits children within the resolved scope.

```
public void Emit(ScopedWriter w)
```

Parameters

w [ScopedWriter](#)

The scoped writer that manages the selector stack and writes CSS.

Class StyleBuilder

Namespace: [RTB.Blazor.Styled.Core](#)

Assembly: RTB.Blazor.Styled.dll

A builder for CSS styles, supporting base declarations, nested selectors, groups (media/supports/container), and keyframes.

```
public sealed class StyleBuilder
```

Inheritance

[object](#) ← StyleBuilder

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Extension Methods

[AnimationStyleExtensions.AnimationDelay\(StyleBuilder, TimeSpan?\)](#) ,
[AnimationStyleExtensions.AnimationDirection\(StyleBuilder, Animation.AnimationDirection?\)](#) ,
[AnimationStyleExtensions.AnimationDuration\(StyleBuilder, TimeSpan?\)](#) ,
[AnimationStyleExtensions.AnimationFillMode\(StyleBuilder, Animation.AnimationFillMode?\)](#) ,
[AnimationStyleExtensions.AnimationIterationCount\(StyleBuilder, int?, bool\)](#) ,
[AnimationStyleExtensions.AnimationName\(StyleBuilder, string?\)](#) ,
[AnimationStyleExtensions.AnimationPlayState\(StyleBuilder, Animation.AnimationPlayState?\)](#) ,
[AnimationStyleExtensions.AnimationTiming\(StyleBuilder, string?\)](#) ,
[BackgroundExtensions.Background\(StyleBuilder, RTBColor?\)](#) ,
[BorderStyleExtensions.Border\(StyleBuilder, SizeUnit?, Border.BorderStyle, RTBColor?, Border.BorderSide?\)](#) ,

[BorderStyleExtensions.BorderAll\(StyleBuilder, SizeUnit?, RTBColor, Border.BorderStyle\)](#) ,
[BorderStyleExtensions.BorderNone\(StyleBuilder\)](#) ,
[BorderStyleExtensions.BorderRadius\(StyleBuilder, SizeUnit?, Border.BorderCorner?\)](#) ,
[BorderStyleExtensions.BorderRadiusAll\(StyleBuilder, SizeUnit?\)](#) ,
[BorderStyleExtensions.BorderRadiusNone\(StyleBuilder\)](#) , [ColorExtensions.Color\(StyleBuilder, RTBColor?\)](#) ,
[FlexExtensions.Flex\(StyleBuilder, Flex.AxisDirection?, Flex.WrapMode?, Flex.Justify?, Flex.Align?, Flex.Align?, Spacing?, int?, int?\)](#) ,
[GridExtensions.Grid\(StyleBuilder, string, string, Spacing?, Grid.Placement?\)](#) ,
[GridPlacementExtensions.GridPlacement\(StyleBuilder, int, int, int, int\)](#) ,

[MarginExtensions.Margin\(StyleBuilder, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?, Spacing?\)](#),
[OpacityExtensions.Opacity\(StyleBuilder, double?\)](#), [OtherExtensions.Other\(StyleBuilder, string, string?\)](#),
[OverflowExtensions.Overflow\(StyleBuilder, Overflow.OverflowMode?\)](#),
[OverflowExtensions.OverflowX\(StyleBuilder, Overflow.OverflowMode?\)](#),
[OverflowExtensions.OverflowY\(StyleBuilder, Overflow.OverflowMode?\)](#),
[PaddingExtensions.PaddingAll\(StyleBuilder, Spacing?\)](#),
[PaddingExtensions.PaddingBottom\(StyleBuilder, Spacing?\)](#),
[PaddingExtensions.PaddingLeft\(StyleBuilder, Spacing?\)](#),
[PaddingExtensions.PaddingRight\(StyleBuilder, Spacing?\)](#),
[PaddingExtensions.PaddingTop\(StyleBuilder, Spacing?\)](#),
[PositionedExtensions.Positioned\(StyleBuilder, Positioned.PositionMode, SizeExpression?, SizeExpression?, SizeExpression?, SizeExpression?\)](#),
[SizeExtensions.Height\(StyleBuilder, SizeExpression?, SizeExpression?, SizeExpression?\)](#),
[SizeExtensions.Width\(StyleBuilder, SizeExpression?, SizeExpression?, SizeExpression?\)](#),
[TransformExtensions.Transform\(StyleBuilder, params string\[\]\)](#),
[TransformExtensions.TransformNone\(StyleBuilder\)](#),
[TransformExtensions.TransformOrigin\(StyleBuilder, string?\)](#),
[TransitionExtensions.Transition\(StyleBuilder, params TransitionItem\[\]\)](#),
[TransitionExtensions.WillChange\(StyleBuilder, string\)](#),
[VisibilityExtensions.Visibility\(StyleBuilder, Visibility.Mode\)](#)

Properties

Base

Base declarations (the root level, no selector).

```
public DeclarationSet Base { get; }
```

Property Value

[DeclarationSet](#)

Start

Create a new StyleBuilder instance.

```
public static StyleBuilder Start { get; }
```

Property Value

[StyleBuilder](#)

Methods

Absorb(StyleBuilder)

Absorb another StyleBuilder's base declarations and fragments into this one.

```
public StyleBuilder Absorb(StyleBuilder other)
```

Parameters

other [StyleBuilder](#)

Returns

[StyleBuilder](#)

AddFragment(IStyleFragment)

Add a style fragment (selector, group, keyframes).

```
public void AddFragment(IStyleFragment f)
```

Parameters

f [IStyleFragment](#)

BuildScoped(string)

Build the complete CSS style as a string, scoped to the provided class name.

```
public string BuildScoped(string className)
```

Parameters

[className](#) [string](#)

Returns

[string](#)

Exceptions

[ArgumentException](#)

ClearAll()

Clear all base declarations and fragments.

```
public StyleBuilder ClearAll()
```

Returns

[StyleBuilder](#)

Compose()

Compose the style by clearing existing declarations and fragments,

```
public void Compose()
```

Container(string, Action<StyleBuilder>)

Add a group rule (media, supports, container) with nested declarations and optional child fragments.

```
public GroupRule Container(string prelude, Action<StyleBuilder> build)
```

Parameters

`prelude` [string](#)

`build` [Action](#)<[StyleBuilder](#)>

Returns

[GroupRule](#)

Group(string, string, Action<StyleBuilder>)

Add a group rule (media, supports, container) with nested declarations and optional child fragments.

```
public GroupRule Group(string kind, string prelude, Action<StyleBuilder> build)
```

Parameters

`kind` [string](#)

`prelude` [string](#)

`build` [Action](#)<[StyleBuilder](#)>

Returns

[GroupRule](#)

Keyframes(string, Action<Keyframes>)

Add or modify a keyframes block by name with nested frames.

```
public Keyframes Keyframes(string name, Action<Keyframes> build)
```

Parameters

`name` [string](#)

`build` [Action](#)<[Keyframes](#)>

Returns

[Keyframes](#)

Media(string, Action<StyleBuilder>)

Add a group rule (media, supports, container) with nested declarations and optional child fragments.

```
public GroupRule Media(string prelude, Action<StyleBuilder> build)
```

Parameters

prelude [string](#)

build [Action](#)<[StyleBuilder](#)>

Returns

[GroupRule](#)

Register(IStyleContributor)

Register a style contributor to participate in the next composition.

```
public void Register(IStyleContributor c)
```

Parameters

c [IStyleContributor](#)

Selector(string, Action<StyleBuilder>)

Add a selector rule with nested declarations and optional child fragments.

```
public SelectorRule Selector(string selector, Action<StyleBuilder> build)
```

Parameters

`selector` [string](#)

`build` [Action](#) <[StyleBuilder](#)>

Returns

[SelectorRule](#)

Set(string, string)

Set a CSS property to a value in the base declaration set.

```
public StyleBuilder Set(string prop, string value)
```

Parameters

`prop` [string](#)

`value` [string](#)

Returns

[StyleBuilder](#)

SetIf(string?, string?, bool)

Set a CSS property to a value in the base declaration set if both are not null or whitespace and the condition is true.

```
public StyleBuilder SetIf(string? prop, string? value, bool Condition)
```

Parameters

`prop` [string](#)

`value` [string](#)

Condition [bool](#)

Returns

[StyleBuilder](#)

SetIf(string?, string?, Func<bool>)

Set a CSS property to a value in the base declaration set if both are not null or whitespace and the condition is true.

```
public StyleBuilder SetIf(string? prop, string? value, Func<bool> Condition)
```

Parameters

prop [string](#)

value [string](#)

Condition [Func](#)<[bool](#)>

Returns

[StyleBuilder](#)

SetIfNotNull(string?, string?)

Set a CSS property to a value in the base declaration set if both are not null or whitespace.

```
public StyleBuilder SetIfNotNull(string? prop, string? value)
```

Parameters

prop [string](#)

value [string](#)

Returns

[StyleBuilder](#)

Supports(string, Action<StyleBuilder>)

Add a group rule (media, supports, container) with nested declarations and optional child fragments.

```
public GroupRule Supports(string prelude, Action<StyleBuilder> build)
```

Parameters

prelude [string](#)

build [Action](#)<[StyleBuilder](#)>

Returns

[GroupRule](#)

Unregister(IStyleContributor)

Unregister a style contributor.

```
public void Unregister(IStyleContributor c)
```

Parameters

c [IStyleContributor](#)

Namespace RTB.Blazor.Styled.Extensions

Classes

[ServiceCollectionExtension](#)

Provides extension methods for registering RTB.Styled services in the dependency injection container.

Class ServiceCollectionExtension

Namespace: [RTB.Blazor.Styled.Extensions](#)

Assembly: RTB.Blazor.Styled.dll

Provides extension methods for registering RTB.Styled services in the dependency injection container.

```
public static class ServiceCollectionExtension
```

Inheritance

[object](#) ← ServiceCollectionExtension

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

UseRTBStyled(IServiceCollection)

Registers RTB.Styled services in the provided [IServiceCollection](#).

```
public static IServiceCollection UseRTBStyled(this IServiceCollection collection)
```

Parameters

collection [IServiceCollection](#)

The service collection to which RTB.Styled services will be added.

Returns

[IServiceCollection](#)

The same [IServiceCollection](#) instance so that additional calls can be chained.

Examples

In Program.cs (Blazor or ASP.NET Core):

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.UseRTBStyled();
```

Remarks

This method registers:

- [IStyleRegistry](#) with a scoped lifetime using [StyleRegistry](#).

It is safe to call multiple times; services are only added if they have not already been registered.

Namespace RTB.Blazor.Styled.Helper

Classes

[AttributeSelector](#)

CSS attribute selector.

[BinarySelector](#)

Binary selector expression combining two selectors with a combinator/operator.

[BinarySizeExpression](#)

Represents a binary size expression composed of two operands and an operator, rendered as a CSS calc() expression.

[BreakPoint](#)

Represents a responsive CSS media query breakpoint.

[ClassSelector](#)

CSS selector for an element with a specific class name.

[CssEnumExtensions](#)

CSS enum extensions.

[ElementSelector](#)

CSS selector for an HTML element by name.

[FunctionalPseudoClass](#)

CSS functional pseudo-class selector, which can take arguments.

[GroupedSelector](#)

Group of selectors separated by commas.

[IdSelector](#)

CSS selector for an element with a specific ID.

[NumericLiteral](#)

Represents a numeric literal value (unit-less).

[ParentSelector](#)

Parent selector reference used in nested rules to refer to the current selector, similar to '&' in SCSS.
Useful for composing states or BEM-style suffixes.

[PseudoClass](#)

CSS pseudo-class selector (non-functional).

[PseudoElement](#)

CSS pseudo-element selector.

[RTBColors](#)

Common named CSS colors (CSS Color Module Level 4).

[RawLiteral](#)

Represents a raw literal size expression that is emitted as-is.

[RawSelector](#)

Raw CSS selector expression. The content is emitted without transformation.

[SelectorExpression](#)

Base type for a small, composable CSS selector DSL. Use provided factory helpers, static members, and operator overloads to build complex selectors in a type-safe way and render them to CSS with [To String\(\)](#).

[SizeExpression](#)

Base class for size expressions.

[SizeLiteral](#)

Represents a literal size value.

[Spacings](#)

Helper methods to compose spacing arrays in the order: top, right, bottom, left. Useful for mapping to CSS shorthand properties (e.g., margin/padding).

Structs

[RTBColor](#)

Represents a color in the sRGB RGBA color space (8-bit per channel). Aligns with common CSS color syntaxes per MDN (hex, rgb[a], hsl[a], named).

[SizeUnit](#)

Represents a CSS size value paired with a unit (px, rem, em, %, vw, vh).

[Spacing](#)

Represents a CSS spacing value, which can be a numeric value and unit or the keyword `auto`.
Supported units: [Px](#), [Em](#), [Rem](#), [Percent](#), [Vw](#), [Vh](#).

Enums

[BreakPoint.MediaType](#)

Supported CSS media types.

[BreakPoint.OrientationType](#)

Supported device orientation values.

[Place](#)

CSS place enum.

[Unit](#)

CSS related enums and extensions.

Class AttributeSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS attribute selector.

```
public sealed record AttributeSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<AttributeSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← AttributeSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[AttributeSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Remarks

Supported operators include '=', '~=' , '^=' , '\$=' , '*=' , and '|='.

Constructors

AttributeSelector(string, string?, string?)

CSS attribute selector.

```
public AttributeSelector(string Name, string? Value = null, string? Op = null)
```

Parameters

Name [string](#)

Attribute name.

Value [string ↗](#)

Optional attribute value; if null renders as [name].

Op [string ↗](#)

Optional operator (defaults to "=" when **Value** is provided).

Remarks

Supported operators include '=', '~=' , '^=' , '\$=' , '*=' , and '|='.

Properties

Name

Attribute name.

```
public string Name { get; init; }
```

Property Value

[string ↗](#)

Op

Optional operator (defaults to "=" when **Value** is provided).

```
public string? Op { get; init; }
```

Property Value

[string ↗](#)

Value

Optional attribute value; if null renders as [name].

```
public string? Value { get; init; }
```

Property Value

[string ↗](#)

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string ↗](#)

Class BinarySelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Binary selector expression combining two selectors with a combinator/operator.

```
public sealed record BinarySelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<BinarySelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← BinarySelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[BinarySelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

BinarySelector(SelectorExpression, string, SelectorExpression)

Binary selector expression combining two selectors with a combinator/operator.

```
public BinarySelector(SelectorExpression Left, string Operator, SelectorExpression Right)
```

Parameters

Left [SelectorExpression](#)

Left-hand selector.

Operator [string](#)

The operator (e.g., " ", ">", "+", "~").

Right [SelectorExpression](#)

Right-hand selector.

Properties

Left

Left-hand selector.

```
public SelectorExpression Left { get; init; }
```

Property Value

[SelectorExpression](#)

Operator

The operator (e.g., " ", ">", "+", "~").

```
public string Operator { get; init; }
```

Property Value

[string](#) ↗

Right

Right-hand selector.

```
public SelectorExpression Right { get; init; }
```

Property Value

[SelectorExpression](#)

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#)

Class BinarySizeExpression

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a binary size expression composed of two operands and an operator, rendered as a CSS calc() expression.

```
public sealed record BinarySizeExpression : SizeExpression, IEquatable<SizeExpression>,  
IEquatable<BinarySizeExpression>
```

Inheritance

[object](#) ← [SizeExpression](#) ← BinarySizeExpression

Implements

[IEquatable](#)<[SizeExpression](#)>, [IEquatable](#)<[BinarySizeExpression](#)>

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

BinarySizeExpression(SizeExpression, string, SizeExpression)

Represents a binary size expression composed of two operands and an operator, rendered as a CSS calc() expression.

```
public BinarySizeExpression(SizeExpression Left, string Operator, SizeExpression Right)
```

Parameters

Left [SizeExpression](#)

Left side of the expression.

Operator [string](#)

The operator symbol (+, -, *, /).

Right [SizeExpression](#)

Right side of the expression.

Properties

Left

Left side of the expression.

```
public SizeExpression Left { get; init; }
```

Property Value

[SizeExpression](#)

Operator

The operator symbol (+, -, *, /).

```
public string Operator { get; init; }
```

Property Value

[string](#) ↗

Right

Right side of the expression.

```
public SizeExpression Right { get; init; }
```

Property Value

[SizeExpression](#)

Methods

Render()

Renders the expression to CSS.

```
protected override string Render()
```

Returns

[string](#)

ToString()

Returns [Render\(\)](#) to ensure the expression is emitted as CSS.

```
public override string ToString()
```

Returns

[string](#)

Class BreakPoint

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a responsive CSS media query breakpoint.

```
public class BreakPoint
```

Inheritance

[object](#) ← BreakPoint

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

- Builds a media query string based on [Media](#), [MinWidth](#), and [MaxWidth](#).
- The generated string intentionally omits the leading "@media " keyword.
- [Orientation](#) is currently not emitted by [ToQuery\(\)](#) and is reserved for future use.
- [SizeExpression](#) instances render to valid CSS when converted to string; operations compose into CSS calc().

Properties

MaxWidth

Optional maximum width constraint for the media query, e.g., ([max-width: 1200px](#)).

```
public SizeExpression? MaxWidth { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Provide a [SizeExpression](#). Its string representation becomes the CSS value.

Media

The media type to target. Defaults to [Screen](#).

```
public BreakPoint.MediaType Media { get; set; }
```

Property Value

[BreakPoint\(MediaType\)](#)

Remarks

Emitted as a lower-cased token in [ToQuery\(\)](#) via an extension like [ToCss\(\)](#).

MinWidth

Optional minimum width constraint for the media query, e.g., `(min-width: 768px)`.

```
public SizeExpression? MinWidth { get; set; }
```

Property Value

[SizeExpression](#)

Remarks

Provide a [SizeExpression](#). Its string representation becomes the CSS value.

Orientation

Optional orientation constraint.

```
public BreakPoint.OrientationType? Orientation { get; set; }
```

Property Value

[BreakPoint.OrientationType?](#)

Remarks

Currently not emitted by [ToQuery\(\)](#). Set for completeness or future extension.

Methods

ToQuery()

Builds the media query condition string.

```
public string ToQuery()
```

Returns

[string](#)

A CSS media condition string without the leading "@media " prefix. Example: `screen` and `(min-width: 768px) and (max-width: 1200px)`.

Remarks

- The media token is lower-cased.
- Includes [MinWidth](#) and/or [MaxWidth](#) when provided.
- Does not include [Orientation](#) at this time.

Enum BreakPoint.MediaType

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Supported CSS media types.

```
public enum BreakPoint.MediaType
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

All = 0

All devices.

Print = 2

Print devices.

Screen = 1

Screens (default).

Enum BreakPoint.OrientationType

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Supported device orientation values.

```
public enum BreakPoint.OrientationType
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Landscape = 1

Landscape orientation.

Portrait = 0

Portrait orientation.

Remarks

Not currently included in the output of [ToQuery\(\)](#); reserved for future support.

Class ClassSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS selector for an element with a specific class name.

```
public sealed record ClassSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<ClassSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← ClassSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[ClassSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

ClassSelector(string)

CSS selector for an element with a specific class name.

```
public ClassSelector(string className)
```

Parameters

ClassName [string](#)

Class name without the leading '!'.

Properties

ClassName

Class name without the leading ''.

```
public string ClassName { get; init; }
```

Property Value

[string](#) ↗

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#) ↗

Class CssEnumExtensions

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS enum extensions.

```
public static class CssEnumExtensions
```

Inheritance

[object](#) ← CssEnumExtensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

ToCss(Enum)

Converts an enum value to a CSS-compatible string (kebab-case).

```
public static string ToCss(this Enum e)
```

Parameters

e [Enum](#)

Returns

[string](#)

Class ElementSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS selector for an HTML element by name.

```
public sealed record ElementSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<ElementSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← ElementSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[ElementSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

ElementSelector(string)

CSS selector for an HTML element by name.

```
public ElementSelector(string Name)
```

Parameters

Name [string](#)

HTML tag name.

Properties

Name

HTML tag name.

```
public string Name { get; init; }
```

Property Value

[string](#) ↗

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#) ↗

Class FunctionalPseudoClass

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS functional pseudo-class selector, which can take arguments.

```
public sealed record FunctionalPseudoClass : SelectorExpression,  
IEquatable<SelectorExpression>, IEquatable<FunctionalPseudoClass>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← FunctionalPseudoClass

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[FunctionalPseudoClass](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

FunctionalPseudoClass(string, IEnumerable<SelectorExpression>)

CSS functional pseudo-class selector, which can take arguments.

```
public FunctionalPseudoClass(string Name, IEnumerable<SelectorExpression> Arguments)
```

Parameters

Name [string](#)

Pseudo-class name without the leading ':'.

Arguments [IEnumerable](#)<[SelectorExpression](#)>

Arguments to the functional pseudo-class.

Properties

Arguments

Arguments to the functional pseudo-class.

```
public IEnumerable<SelectorExpression> Arguments { get; init; }
```

Property Value

[IEnumerable](#)<[SelectorExpression](#)>

Name

Pseudo-class name without the leading ':'.

```
public string Name { get; init; }
```

Property Value

[string](#)

Methods

Dir(string)

:dir(direction)

```
public static FunctionalPseudoClass Dir(string direction)
```

Parameters

`direction` [string](#)

Returns

[FunctionalPseudoClass](#)

Has(params SelectorExpression[])

:has(...)

```
public static FunctionalPseudoClass Has(params SelectorExpression[] args)
```

Parameters

`args` [SelectorExpression](#)[]

Returns

[FunctionalPseudoClass](#)

HasSlotted(params SelectorExpression[])

(Proposed) :has-slotted(...)

```
public static FunctionalPseudoClass HasSlotted(params SelectorExpression[] args)
```

Parameters

`args` [SelectorExpression](#)[]

Returns

[FunctionalPseudoClass](#)

HostContext(params SelectorExpression[])

:host-context(...)

```
public static FunctionalPseudoClass HostContext(params SelectorExpression[] args)
```

Parameters

args [SelectorExpression\[\]](#)

Returns

[FunctionalPseudoClass](#)

HostSelector(params SelectorExpression[])

:host(...)

```
public static FunctionalPseudoClass HostSelector(params SelectorExpression[] args)
```

Parameters

args [SelectorExpression\[\]](#)

Returns

[FunctionalPseudoClass](#)

Is(params SelectorExpression[])

:is(...)

```
public static FunctionalPseudoClass Is(params SelectorExpression[] args)
```

Parameters

args [SelectorExpression\[\]](#)

Returns

[FunctionalPseudoClass](#)

Lang(string)

:lang(code)

```
public static FunctionalPseudoClass Lang(string code)
```

Parameters

code [string](#)

Returns

[FunctionalPseudoClass](#)

Not(params SelectorExpression[])

:not(...)

```
public static FunctionalPseudoClass Not(params SelectorExpression[] args)
```

Parameters

args [SelectorExpression](#)[]

Returns

[FunctionalPseudoClass](#)

NthChild(string)

:nth-child(expr)

```
public static FunctionalPseudoClass NthChild(string expr)
```

Parameters

`expr` [string ↗](#)

Returns

[FunctionalPseudoClass](#)

NthLastChild(string)

:nth-last-child(expr)

```
public static FunctionalPseudoClass NthLastChild(string expr)
```

Parameters

`expr` [string ↗](#)

Returns

[FunctionalPseudoClass](#)

NthLastOfType(string)

:nth-last-of-type(expr)

```
public static FunctionalPseudoClass NthLastOfType(string expr)
```

Parameters

`expr` [string ↗](#)

Returns

[FunctionalPseudoClass](#)

NthOfType(string)

:nth-of-type(expr)

```
public static FunctionalPseudoClass NthOfType(string expr)
```

Parameters

expr [string](#)

Returns

[FunctionalPseudoClass](#)

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#)

Where(params SelectorExpression[])

:where(...)

```
public static FunctionalPseudoClass Where(params SelectorExpression[] args)
```

Parameters

args [SelectorExpression](#)[]

Returns

[FunctionalPseudoClass](#)

Class GroupedSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Group of selectors separated by commas.

```
public sealed record GroupedSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<GroupedSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← GroupedSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[GroupedSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

GroupedSelector(IEnumerable<SelectorExpression>)

Group of selectors separated by commas.

```
public GroupedSelector(IEnumerable<SelectorExpression> Selectors)
```

Parameters

Selectors [IEnumerable](#)<[SelectorExpression](#)>

Selectors to group.

Properties

Selectors

Selectors to group.

```
public IEnumerable<SelectorExpression> Selectors { get; init; }
```

Property Value

[IEnumerable](#) <[SelectorExpression](#)>

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#)

Class IdSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS selector for an element with a specific ID.

```
public sealed record IdSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<IdSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← IdSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[IdSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

IdSelector(string)

CSS selector for an element with a specific ID.

```
public IdSelector(string IdName)
```

Parameters

IdName [string](#)

ID without the leading '#'.

Properties

IdName

ID without the leading '#'.

```
public string IdName { get; init; }
```

Property Value

[string](#) ↗

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#) ↗

Class NumericLiteral

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a numeric literal value (unit-less).

```
public sealed record NumericLiteral : SizeExpression, IEquatable<SizeExpression>,  
IEquatable<NumericLiteral>
```

Inheritance

[object](#) ← [SizeExpression](#) ← NumericLiteral

Implements

[IEquatable](#)<[SizeExpression](#)>, [IEquatable](#)<[NumericLiteral](#)>

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

NumericLiteral(double)

Represents a numeric literal value (unit-less).

```
public NumericLiteral(double Value)
```

Parameters

Value [double](#)

The numeric value rendered in expressions.

Properties

Value

The numeric value rendered in expressions.

```
public double Value { get; init; }
```

Property Value

[double](#)

Methods

Render()

Renders the expression to CSS.

```
protected override string Render()
```

Returns

[string](#)

ToString()

Returns [Render\(\)](#) to ensure the expression is emitted as CSS.

```
public override string ToString()
```

Returns

[string](#)

Class ParentSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Parent selector reference used in nested rules to refer to the current selector, similar to '&' in SCSS.
Useful for composing states or BEM-style suffixes.

```
public sealed record ParentSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<ParentSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← ParentSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[ParentSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

ParentSelector()

Parent selector reference used in nested rules to refer to the current selector, similar to '&' in SCSS.
Useful for composing states or BEM-style suffixes.

```
public ParentSelector()
```

Properties

Parent

Gets a new instance representing the current selector "&".

```
public static ParentSelector Parent { get; }
```

Property Value

[ParentSelector](#)

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#)

Enum Place

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS place enum.

```
public enum Place
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Center = 2

Center

End = 1

End (flex-end)

Start = 0

Start (flex-start)

Stretch = 3

Stretch

Class PseudoClass

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS pseudo-class selector (non-functional).

```
public sealed record PseudoClass : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<PseudoClass>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← PseudoClass

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[PseudoClass](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

PseudoClass(string)

CSS pseudo-class selector (non-functional).

```
public PseudoClass(string Name)
```

Parameters

Name [string](#)

Pseudo-class name without the leading ':'.

Properties

Active

:active

```
public static PseudoClass Active { get; }
```

Property Value

[PseudoClass](#)

AnyLink

:any-link

```
public static PseudoClass AnyLink { get; }
```

Property Value

[PseudoClass](#)

Autofill

:autofill

```
public static PseudoClass Autofill { get; }
```

Property Value

[PseudoClass](#)

Blank

:blank

```
public static PseudoClass Blank { get; }
```

Property Value

[PseudoClass](#)

Buffering

:buffering

```
public static PseudoClass Buffering { get; }
```

Property Value

[PseudoClass](#)

Checked

:checked

```
public static PseudoClass Checked { get; }
```

Property Value

[PseudoClass](#)

Current

:current

```
public static PseudoClass Current { get; }
```

Property Value

[PseudoClass](#)

Default

:default

```
public static PseudoClass Default { get; }
```

Property Value

[PseudoClass](#)

Defined

:defined

```
public static PseudoClass Defined { get; }
```

Property Value

[PseudoClass](#)

Disabled

:disabled

```
public static PseudoClass Disabled { get; }
```

Property Value

[PseudoClass](#)

Empty

:empty

```
public static PseudoClass Empty { get; }
```

Property Value

[PseudoClass](#)

Enabled

:enabled

```
public static PseudoClass Enabled { get; }
```

Property Value

[PseudoClass](#)

FirstChild

:first-child

```
public static PseudoClass FirstChild { get; }
```

Property Value

[PseudoClass](#)

FirstOfType

:first-of-type

```
public static PseudoClass FirstOfType { get; }
```

Property Value

[PseudoClass](#)

Focus

:focus

```
public static PseudoClass Focus { get; }
```

Property Value

[PseudoClass](#)

FocusVisible

:focus-visible

```
public static PseudoClass FocusVisible { get; }
```

Property Value

[PseudoClass](#)

FocusWithin

:focus-within

```
public static PseudoClass FocusWithin { get; }
```

Property Value

[PseudoClass](#)

Fullscreen

:fullscreen

```
public static PseudoClass Fullscreen { get; }
```

Property Value

[PseudoClass](#)

Future

:future

```
public static PseudoClass Future { get; }
```

Property Value

[PseudoClass](#)

Host

:host

```
public static PseudoClass Host { get; }
```

Property Value

[PseudoClass](#)

Hover

:hover

```
public static PseudoClass Hover { get; }
```

Property Value

[PseudoClass](#)

InRange

:in-range

```
public static PseudoClass InRange { get; }
```

Property Value

[PseudoClass](#)

Indeterminate

:indeterminate

```
public static PseudoClass Indeterminate { get; }
```

Property Value

[PseudoClass](#)

Invalid

:invalid

```
public static PseudoClass Invalid { get; }
```

Property Value

[PseudoClass](#)

LastChild

:last-child

```
public static PseudoClass LastChild { get; }
```

Property Value

[PseudoClass](#)

LastOfType

:last-of-type

```
public static PseudoClass LastOfType { get; }
```

Property Value

[PseudoClass](#)

Link

:link

```
public static PseudoClass Link { get; }
```

Property Value

[PseudoClass](#)

LocalLink

:local-link

```
public static PseudoClass LocalLink { get; }
```

Property Value

[PseudoClass](#)

Modal

:modal

```
public static PseudoClass Modal { get; }
```

Property Value

[PseudoClass](#)

Muted

:muted

```
public static PseudoClass Muted { get; }
```

Property Value

[PseudoClass](#)

Name

Pseudo-class name without the leading ':'.

```
public string Name { get; init; }
```

Property Value

[string](#) ↗

OnlyChild

:only-child

```
public static PseudoClass OnlyChild { get; }
```

Property Value

[PseudoClass](#)

OnlyOfType

:only-of-type

```
public static PseudoClass OnlyOfType { get; }
```

Property Value

[PseudoClass](#)

Open

:open

```
public static PseudoClass Open { get; }
```

Property Value

[PseudoClass](#)

Optional

:optional

```
public static PseudoClass Optional { get; }
```

Property Value

[PseudoClass](#)

OutOfRange

:out-of-range

```
public static PseudoClass OutOfRange { get; }
```

Property Value

[PseudoClass](#)

Past

:past

```
public static PseudoClass Past { get; }
```

Property Value

[PseudoClass](#)

Paused

:paused

```
public static PseudoClass Paused { get; }
```

Property Value

[PseudoClass](#)

PictureInPicture

:picture-in-picture

```
public static PseudoClass PictureInPicture { get; }
```

Property Value

[PseudoClass](#)

PlaceholderShown

:placeholder-shown

```
public static PseudoClass PlaceholderShown { get; }
```

Property Value

[PseudoClass](#)

Playing

:playing

```
public static PseudoClass Playing { get; }
```

Property Value

[PseudoClass](#)

PopoverOpen

:popover-open

```
public static PseudoClass PopoverOpen { get; }
```

Property Value

[PseudoClass](#)

ReadOnly

:read-only

```
public static PseudoClass ReadOnly { get; }
```

Property Value

[PseudoClass](#)

ReadWrite

:read-write

```
public static PseudoClass ReadWrite { get; }
```

Property Value

[PseudoClass](#)

Required

:required

```
public static PseudoClass Required { get; }
```

Property Value

[PseudoClass](#)

Root

:root

```
public static PseudoClass Root { get; }
```

Property Value

[PseudoClass](#)

Scope

:scope

```
public static PseudoClass Scope { get; }
```

Property Value

[PseudoClass](#)

Seeking

:seeking

```
public static PseudoClass Seeking { get; }
```

Property Value

[PseudoClass](#)

Stalled

:stalled

```
public static PseudoClass Stalled { get; }
```

Property Value

[PseudoClass](#)

Target

:target

```
public static PseudoClass Target { get; }
```

Property Value

[PseudoClass](#)

TargetWithin

:target-within

```
public static PseudoClass TargetWithin { get; }
```

Property Value

[PseudoClass](#)

UserInvalid

:user-invalid

```
public static PseudoClass UserInvalid { get; }
```

Property Value

[PseudoClass](#)

UserValid

:user-valid

```
public static PseudoClass UserValid { get; }
```

Property Value

[PseudoClass](#)

Valid

:valid

```
public static PseudoClass Valid { get; }
```

Property Value

[PseudoClass](#)

Visited

:visited

```
public static PseudoClass Visited { get; }
```

Property Value

[PseudoClass](#)

VolumeLocked

:volume-locked

```
public static PseudoClass VolumeLocked { get; }
```

Property Value

[PseudoClass](#)

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#) ↗

Class PseudoElement

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS pseudo-element selector.

```
public sealed record PseudoElement : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<PseudoElement>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← PseudoElement

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[PseudoElement](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

PseudoElement(string)

CSS pseudo-element selector.

```
public PseudoElement(string Name)
```

Parameters

Name [string](#)

Pseudo-element name without the leading '::'.

Properties

After

::after

```
public static PseudoElement After { get; }
```

Property Value

[PseudoElement](#)

Backdrop

::backdrop

```
public static PseudoElement Backdrop { get; }
```

Property Value

[PseudoElement](#)

Before

::before

```
public static PseudoElement Before { get; }
```

Property Value

[PseudoElement](#)

Cue

::cue

```
public static PseudoElement Cue { get; }
```

Property Value

[PseudoElement](#)

FileSelectorButton

::file-selector-button

```
public static PseudoElement FileSelectorButton { get; }
```

Property Value

[PseudoElement](#)

FirstLetter

::first-letter

```
public static PseudoElement FirstLetter { get; }
```

Property Value

[PseudoElement](#)

FirstLine

::first-line

```
public static PseudoElement FirstLine { get; }
```

Property Value

[PseudoElement](#)

Marker

::marker

```
public static PseudoElement Marker { get; }
```

Property Value

[PseudoElement](#)

Name

Pseudo-element name without the leading '::'.

```
public string Name { get; init; }
```

Property Value

[string](#) ↗

Placeholder

::placeholder

```
public static PseudoElement Placeholder { get; }
```

Property Value

[PseudoElement](#)

Selection

::selection

```
public static PseudoElement Selection { get; }
```

Property Value

[PseudoElement](#)

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#)

Slotted(string)

::slotted(name)

```
public static PseudoElement Slotted(string name)
```

Parameters

name [string](#)

The slotted selector argument.

Returns

[PseudoElement](#)

Struct RTBColor

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a color in the sRGB RGBA color space (8-bit per channel). Aligns with common CSS color syntaxes per MDN (hex, rgb[a], hsl[a], named).

```
public readonly record struct RTBColor : IEquatable<RTBColor>
```

Implements

[IEquatable](#)<[RTBColor](#)>

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Remarks

- Hex: #RGB, #RGBA, #RRGGBB, #RRGGBBAA
- RGB: `rgb(255, 0, 0)`, `rgb(255 0 0)`, percentages, optional alpha via / or fourth value
- HSL: `hsl(120, 100%, 50%)`, space-separated, optional alpha via / or fourth value, hue supports deg
- Named: any CSS Level 4 named color (see [Named](#)) and [transparent](#)

Constructors

RTBColor(byte, byte, byte, byte)

Represents a color in the sRGB RGBA color space (8-bit per channel). Aligns with common CSS color syntaxes per MDN (hex, rgb[a], hsl[a], named).

```
public RTBColor(byte R, byte G, byte B, byte A)
```

Parameters

R [byte](#)

Red channel 0..255.

G [byte](#)

Green channel 0..255.

B [byte](#)

Blue channel 0..255.

A [byte](#)

Alpha channel 0..255 (255 = opaque).

Remarks

- Hex: #RGB, #RGBA, #RRGGBB, #RRGGBBAA
- RGB: `rgb(255,0,0)`, `rgb(255 0 0)`, percentages, optional alpha via / or fourth value
- HSL: `hsl(120, 100%, 50%)`, space-separated, optional alpha via / or fourth value, hue supports deg
- Named: any CSS Level 4 named color (see [Named](#)) and `transparent`

Properties

A

Alpha channel 0..255 (255 = opaque).

```
public byte A { get; init; }
```

Property Value

[byte](#)

Alpha

Alpha channel 0..255.

```
public byte Alpha { get; }
```

Property Value

[byte](#)

B

Blue channel 0..255.

```
public byte B { get; init; }
```

Property Value

[byte ↗](#)

Blue

Blue channel 0..255.

```
public byte Blue { get; }
```

Property Value

[byte ↗](#)

G

Green channel 0..255.

```
public byte G { get; init; }
```

Property Value

[byte ↗](#)

Green

Green channel 0..255.

```
public byte Green { get; }
```

Property Value

[byte](#)

HexRgb

Hex string in #RRGGBB format (no alpha).

```
public string HexRgb { get; }
```

Property Value

[string](#)

HexRgba

Hex string in #RRGGBBAA format.

```
public string HexRgba { get; }
```

Property Value

[string](#)

R

Red channel 0..255.

```
public byte R { get; init; }
```

Property Value

[byte](#)

Red

Red channel 0..255.

```
public byte Red { get; }
```

Property Value

[byte](#)

Methods

Blend(RTBColor, double)

Linear blend towards another color in RGBA space.

```
public RTBColor Blend(RTBColor target, double pct)
```

Parameters

target [RTBColor](#)

Target color.

pct [double](#)

Blend factor 0..1 (0 = self, 1 = target).

Returns

[RTBColor](#)

Blended color.

Examples

```
var mid = a.Blend(b, 0.5);
```

Darken(double)

Darken by subtracting from HSL lightness.

```
public RTBColor Darken(double pct)
```

Parameters

pct [double](#)

Delta in 0..1 (e.g. 0.1 to darken by 10%).

Returns

[RTBColor](#)

Desaturate(double)

Decrease HSL saturation.

```
public RTBColor Desaturate(double pct)
```

Parameters

pct [double](#)

Delta in 0..1.

Returns

[RTBColor](#)

FromCss(string)

Alias for [Parse\(string\)](#).

```
public static RTBColor FromCss(string css)
```

Parameters

`css string` ↗

CSS color string.

Returns

[RTBColor](#)

Parsed [RTBColor](#).

FromRgb(byte, byte, byte)

Create an opaque color from 8-bit RGB.

```
public static RTBColor FromRgb(byte r, byte g, byte b)
```

Parameters

`r byte` ↗

Red 0..255.

`g byte` ↗

Green 0..255.

`b byte` ↗

Blue 0..255.

Returns

[RTBColor](#)

Opaque [RTBColor](#) with alpha 255.

Examples

```
var c = RTBColor.FromRgb(255, 0, 0); // #FF0000FF
```

FromRgba(byte, byte, byte, byte)

Create a color from 8-bit RGBA.

```
public static RTBColor FromRgba(byte r, byte g, byte b, byte a)
```

Parameters

r [byte](#)

Red 0..255.

g [byte](#)

Green 0..255.

b [byte](#)

Blue 0..255.

a [byte](#)

Alpha 0..255 (255 = opaque).

Returns

[RTBColor](#)

[RTBColor](#).

Lighten(double)

Lighten by adding to HSL lightness.

```
public RTBColor Lighten(double pct)
```

Parameters

pct [double](#)

Delta in 0..1 (e.g. 0.1 to lighten by 10%).

Returns

[RTBColor](#)

Parse(string)

Parse a CSS color value. Supports hex, rgb[a], hsl[a], named colors, and `transparent`.

```
public static RTBColor Parse(string css)
```

Parameters

`css` [string](#)

CSS color string.

Returns

[RTBColor](#)

Parsed [RTBColor](#).

Examples

```
var a = RTBColor.Parse("#09f");           // short hex
var b = RTBColor.Parse("rgb(0 153 255)"); // space separated
var c = RTBColor.Parse("hsl(200 100% 50% / 50%)"); // with alpha
var d = RTBColor.Parse("royalblue");        // named
```

Exceptions

[FormatException](#)

If the string cannot be parsed.

Saturate(double)

Increase HSL saturation.

```
public RTBColor Saturate(double pct)
```

Parameters

pct [double](#)

Delta in 0..1.

Returns

[RTBColor](#)

ToString()

Returns [HexRgba](#) by default.

```
public override string ToString()
```

Returns

[string](#)

TryParse(string?, out RTBColor)

Try to parse a CSS color value safely.

```
public static bool TryParse(string? css, out RTBColor color)
```

Parameters

css [string](#)

CSS color string.

color [RTBColor](#)

Result on success; default on failure.

Returns

[bool](#)

`true` if parsed successfully; otherwise `false`.

WithAlpha(double)

Returns a copy with a new alpha.

```
public RTBColor WithAlpha(double alpha)
```

Parameters

`alpha` [double](#)

Alpha 0..1 (1 = opaque).

Returns

[RTBColor](#)

New color with updated alpha.

Operators

implicit operator string(RTBColor)

Implicitly convert [RTBColor](#) to `#RRGGBBAA` string.

```
public static implicit operator string(RTBColor c)
```

Parameters

`c` [RTBColor](#)

Returns

[string](#)

implicit operator RTBColor(string)

Implicitly parse a CSS color string to [RTBColor](#). Throws on failure.

```
public static implicit operator RTBColor(string css)
```

Parameters

`css` [string](#) ↗

Returns

[RTBColor](#)

Class RTBColors

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Common named CSS colors (CSS Color Module Level 4).

```
public static class RTBColors
```

Inheritance

[object](#) ← RTBColors

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Black

Pure black #000000FF.

```
public static RTBColor Black { get; }
```

Property Value

[RTBColor](#)

Blue

Blue #0000FFFF.

```
public static RTBColor Blue { get; }
```

Property Value

[RTBColor](#)

Gray

Neutral gray #808080FF.

```
public static RTBColor Gray { get; }
```

Property Value

[RTBColor](#)

Green

Green #00FF00FF.

```
public static RTBColor Green { get; }
```

Property Value

[RTBColor](#)

Magenta

Magenta #FF00FFFF.

```
public static RTBColor Magenta { get; }
```

Property Value

[RTBColor](#)

Named

Full CSS named colors dictionary (case-insensitive keys).

```
public static IReadOnlyDictionary<string, RTBColor> Named { get; }
```

Property Value

[IReadOnlyDictionary](#)<string, RTBColor>

Orange

Orange convenience color #FF8800FF (non-standard name mapping).

```
public static RTBColor Orange { get; }
```

Property Value

[RTBColor](#)

Red

Red #FF0000FF.

```
public static RTBColor Red { get; }
```

Property Value

[RTBColor](#)

Transparent

Fully transparent #00000000.

```
public static RTBColor Transparent { get; }
```

Property Value

[RTBColor](#)

White

Pure white #FFFFFF.

```
public static RTBColor White { get; }
```

Property Value

[RTBColor](#)

Yellow

Yellow #FFFF00FF.

```
public static RTBColor Yellow { get; }
```

Property Value

[RTBColor](#)

Methods

NamedColor(string)

Get a named CSS color by name.

```
public static RTBColor NamedColor(string name)
```

Parameters

name [string](#) ↗

Color name (any case). E.g. "royalblue".

Returns

[RTBColor](#)

Named [RTBColor](#).

Exceptions

[KeyNotFoundException](#) ↗

If the name is unknown.

Class RawLiteral

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a raw literal size expression that is emitted as-is.

```
public sealed record RawLiteral : SizeExpression, IEquatable<SizeExpression>,
IEquatable<RawLiteral>
```

Inheritance

[object](#) ← [SizeExpression](#) ← RawLiteral

Implements

[IEquatable](#)<[SizeExpression](#)>, [IEquatable](#)<[RawLiteral](#)>

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

RawLiteral(string)

Represents a raw literal size expression that is emitted as-is.

```
public RawLiteral(string Raw)
```

Parameters

Raw [string](#)

The raw CSS expression.

Properties

Raw

The raw CSS expression.

```
public string Raw { get; init; }
```

Property Value

[string ↗](#)

Methods

Render()

Renders the expression to CSS.

```
protected override string Render()
```

Returns

[string ↗](#)

ToString()

Returns [Render\(\)](#) to ensure the expression is emitted as CSS.

```
public override string ToString()
```

Returns

[string ↗](#)

Class RawSelector

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Raw CSS selector expression. The content is emitted without transformation.

```
public sealed record RawSelector : SelectorExpression, IEquatable<SelectorExpression>,  
IEquatable<RawSelector>
```

Inheritance

[object](#) ← [SelectorExpression](#) ← RawSelector

Implements

[IEquatable](#)<[SelectorExpression](#)>, [IEquatable](#)<[RawSelector](#)>

Inherited Members

[SelectorExpression.Element\(string\)](#), [SelectorExpression.Id\(string\)](#), [SelectorExpression.Class\(string\)](#),
[SelectorExpression.Attribute\(string, string, string\)](#), [SelectorExpression.PseudoClass\(string\)](#),
[SelectorExpression.PseudoElement\(string\)](#), [SelectorExpression.ToString\(\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

RawSelector(string)

Raw CSS selector expression. The content is emitted without transformation.

```
public RawSelector(string Raw)
```

Parameters

Raw [string](#)

Raw CSS selector text.

Properties

Raw

Raw CSS selector text.

```
public string Raw { get; init; }
```

Property Value

[string](#) ↗

Methods

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected override string Render()
```

Returns

[string](#) ↗

Class SelectorExpression

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Base type for a small, composable CSS selector DSL. Use provided factory helpers, static members, and operator overloads to build complex selectors in a type-safe way and render them to CSS with [ToString\(\)](#).

```
public abstract record SelectorExpression : IEquatable<SelectorExpression>
```

Inheritance

[object](#) ← SelectorExpression

Implements

[IEquatable](#)<[SelectorExpression](#)>

Derived

[AttributeSelector](#), [BinarySelector](#), [ClassSelector](#), [ElementSelector](#), [FunctionalPseudoClass](#),
[GroupedSelector](#), [IdSelector](#), [ParentSelector](#), [PseudoClass](#), [PseudoElement](#), [RawSelector](#)

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#)

Examples

```
var s = SelectorExpression.Element("ul") / SelectorExpression.Element("li") +  
SelectorExpression.PseudoClass("hover") > SelectorExpression.Element("a") &  
SelectorExpression.Class("primary"); // Renders: "ul li+ :hover > a~.primary" var css = s.ToString();
```

Methods

Attribute(string, string?, string?)

Creates a selector expression for an attribute of an element.

```
public static SelectorExpression Attribute(string name, string? value = null, string? op  
= null)
```

Parameters

name [string](#)

Attribute name.

value [string](#)

Optional attribute value. If omitted, renders as [name].

op [string](#)

Optional operator (e.g., "=", "~=", "^=", "\$=", "*=", "|="). Defaults to "=" when **value** is provided.

Returns

[SelectorExpression](#)

Class(string)

Creates a selector expression for an element with a specific class name.

```
public static SelectorExpression Class(string cls)
```

Parameters

cls [string](#)

Class name without the leading '!'.

Returns

[SelectorExpression](#)

Element(string)

Creates a selector expression for an HTML element by its name.

```
public static SelectorExpression Element(string name)
```

Parameters

name [string](#)

HTML tag name (e.g., "div", "button").

Returns

[SelectorExpression](#)

Id(string)

Creates a selector expression for an element with a specific ID.

```
public static SelectorExpression Id(string id)
```

Parameters

id [string](#)

ID without the leading '#'.

Returns

[SelectorExpression](#)

PseudoClass(string)

Creates a selector expression for a CSS pseudo-class.

```
public static SelectorExpression PseudoClass(string name)
```

Parameters

name [string](#)

Pseudo-class name without the leading ':'.

Returns

[SelectorExpression](#)

PseudoElement(string)

Creates a selector expression for a CSS pseudo-element.

```
public static SelectorExpression PseudoElement(string name)
```

Parameters

name [string](#)

Pseudo-element name without the leading '::'.

Returns

[SelectorExpression](#)

Render()

Renders the selector to CSS. Implemented by derived records.

```
protected abstract string Render()
```

Returns

[string](#)

ToString()

Renders the selector to CSS.

```
public override string ToString()
```

Returns

[string](#)

Operators

operator +(SelectorExpression, SelectorExpression)

Adjacent sibling combinator: selects the immediate following sibling.

```
public static SelectorExpression operator +(SelectorExpression a, SelectorExpression b)
```

Parameters

a [SelectorExpression](#)

b [SelectorExpression](#)

Returns

[SelectorExpression](#)

Examples

```
SelectorExpression.Class("item") + SelectorExpression.Class("badge") // ".item+.badge"
```

operator &(SelectorExpression, SelectorExpression)

General sibling combinator: selects any following sibling.

```
public static SelectorExpression operator &(SelectorExpression a, SelectorExpression b)
```

Parameters

a [SelectorExpression](#)

b [SelectorExpression](#)

Returns

[SelectorExpression](#)

Examples

SelectorExpression.Class("item") & SelectorExpression.Class("badge") // ".item~.badge"

operator |(SelectorExpression, SelectorExpression)

Grouping selector: combines selectors separated by commas.

```
public static SelectorExpression operator |(SelectorExpression a, SelectorExpression b)
```

Parameters

a [SelectorExpression](#)

b [SelectorExpression](#)

Returns

[SelectorExpression](#)

Examples

(SelectorExpression.Element("h1") | SelectorExpression.Element("h2")).ToString() // "h1, h2"

operator /(SelectorExpression, SelectorExpression)

Descendant combinator: selects any level descendant (whitespace).

```
public static SelectorExpression operator /(SelectorExpression parent,  
SelectorExpression descendant)
```

Parameters

parent [SelectorExpression](#)

Ancestor selector.

descendant [SelectorExpression](#)

Descendant selector.

Returns

[SelectorExpression](#)

A combined selector using a space.

Examples

SelectorExpression.Element("ul") / SelectorExpression.Element("li") // "ul li"

operator >(SelectorExpression, SelectorExpression)

Child combinator: selects direct children.

```
public static SelectorExpression operator >(SelectorExpression parent,  
SelectorExpression child)
```

Parameters

`parent` [SelectorExpression](#)

Left selector (parent).

`child` [SelectorExpression](#)

Right selector (direct child).

Returns

[SelectorExpression](#)

A combined selector using the '>' combinator.

Examples

SelectorExpression.Element("ul") > SelectorExpression.Element("li") // "ul>li"

implicit operator string?(SelectorExpression?)

Implicitly converts a [SelectorExpression](#) to its string (CSS) representation.

```
public static implicit operator string?(SelectorExpression? expr)
```

Parameters

expr [SelectorExpression](#)

Selector expression instance.

Returns

[string](#)

implicit operator SelectorExpression(string)

Implicitly converts a raw string to a [SelectorExpression](#). The string is used as-is.

```
public static implicit operator SelectorExpression(string literal)
```

Parameters

literal [string](#)

Raw CSS selector text.

Returns

[SelectorExpression](#)

operator <(SelectorExpression, SelectorExpression)

Not supported in CSS selector combinator.

```
[Obsolete("The '<' operator is not valid for CSS selectors.", true)]
public static SelectorExpression operator <(SelectorExpression parent,
SelectorExpression child)
```

Parameters

parent [SelectorExpression](#)

child [SelectorExpression](#)

Returns

[SelectorExpression](#)

Class SizeExpression

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Base class for size expressions.

```
public abstract record SizeExpression : IEquatable<SizeExpression>
```

Inheritance

[object](#) ← SizeExpression

Implements

[IEquatable](#) <[SizeExpression](#)>

Derived

[BinarySizeExpression](#), [NumericLiteral](#), [RawLiteral](#), [SizeLiteral](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Remarks

Instances render to valid CSS via [ToString\(\)](#) or [Render\(\)](#). Use operators to compose expressions; the output is wrapped in CSS calc().

Methods

Render()

Renders the expression to CSS.

```
protected abstract string Render()
```

Returns

[string](#)

ToString()

Returns [Render\(\)](#) to ensure the expression is emitted as CSS.

```
public override string ToString()
```

Returns

[string](#)

Operators

operator +(SizeExpression, SizeExpression)

Adds two expressions into a CSS calc() expression.

```
public static SizeExpression operator +(SizeExpression a, SizeExpression b)
```

Parameters

a [SizeExpression](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

operator /(SizeExpression, SizeExpression)

Divides two expressions into a CSS calc() expression.

```
public static SizeExpression operator /(SizeExpression a, SizeExpression b)
```

Parameters

a [SizeExpression](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

operator /(SizeExpression, double)

Divides an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator /(SizeExpression a, double b)
```

Parameters

a [SizeExpression](#)

b [double](#) ↗

Returns

[SizeExpression](#)

operator /(SizeExpression, int)

Divides an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator /(SizeExpression a, int b)
```

Parameters

a [SizeExpression](#)

b [int](#) ↗

Returns

[SizeExpression](#)

operator /(double, SizeExpression)

Divides a numeric literal by an expression into a CSS calc() expression.

```
public static SizeExpression operator /(double a, SizeExpression b)
```

Parameters

a [double](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

operator /(int, SizeExpression)

Divides a numeric literal by an expression into a CSS calc() expression.

```
public static SizeExpression operator /(int a, SizeExpression b)
```

Parameters

a [int](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

implicit operator string?(SizeExpression?)

Renders the expression to its CSS string representation.

```
public static implicit operator string?(SizeExpression? unit)
```

Parameters

unit [SizeExpression](#)

Returns

[string](#) ↗

implicit operator SizeExpression(SizeUnit)

Implicitly converts a [SizeUnit](#) to a [SizeExpression](#).

```
public static implicit operator SizeExpression(SizeUnit unit)
```

Parameters

unit [SizeUnit](#)

Returns

[SizeExpression](#)

implicit operator SizeExpression(double)

Implicitly converts a double to a pixel [SizeExpression](#).

```
public static implicit operator SizeExpression(double value)
```

Parameters

value [double](#) ↗

Returns

[SizeExpression](#)

implicit operator SizeExpression(int)

Implicitly converts an integer to a pixel [SizeExpression](#).

```
public static implicit operator SizeExpression(int value)
```

Parameters

`value` [int](#)

Returns

[SizeExpression](#)

implicit operator SizeExpression(string)

Implicitly converts a CSS size literal to a [SizeExpression](#).

```
public static implicit operator SizeExpression(string litteral)
```

Parameters

`litteral` [string](#)

Returns

[SizeExpression](#)

operator *(SizeExpression, SizeExpression)

Multiplies two expressions into a CSS calc() expression.

```
public static SizeExpression operator *(SizeExpression a, SizeExpression b)
```

Parameters

`a` [SizeExpression](#)

`b` [SizeExpression](#)

Returns

[SizeExpression](#)

operator *(SizeExpression, double)

Multiplies an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator *(SizeExpression a, double b)
```

Parameters

a [SizeExpression](#)

b [double](#) ↗

Returns

[SizeExpression](#)

operator *(SizeExpression, int)

Multiplies an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator *(SizeExpression a, int b)
```

Parameters

a [SizeExpression](#)

b [int](#) ↗

Returns

[SizeExpression](#)

operator *(double, SizeExpression)

Multiplies an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator *(double a, SizeExpression b)
```

Parameters

a [double](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

operator *(int, SizeExpression)

Multiplies an expression by a numeric literal into a CSS calc() expression.

```
public static SizeExpression operator *(int a, SizeExpression b)
```

Parameters

a [int](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

operator -(SizeExpression, SizeExpression)

Subtracts two expressions into a CSS calc() expression.

```
public static SizeExpression operator -(SizeExpression a, SizeExpression b)
```

Parameters

a [SizeExpression](#)

b [SizeExpression](#)

Returns

[SizeExpression](#)

Class SizeLiteral

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a literal size value.

```
public sealed record SizeLiteral : SizeExpression, IEquatable<SizeExpression>,  
IEquatable<SizeLiteral>
```

Inheritance

[object](#) ← [SizeExpression](#) ← SizeLiteral

Implements

[IEquatable](#)<[SizeExpression](#)>, [IEquatable](#)<[SizeLiteral](#)>

Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.ReferenceEquals\(object, object\)](#)

Constructors

SizeLiteral(SizeUnit)

Represents a literal size value.

```
public SizeLiteral(SizeUnit Unit)
```

Parameters

Unit [SizeUnit](#)

The underlying size value and unit.

Properties

Unit

The underlying size value and unit.

```
public SizeUnit Unit { get; init; }
```

Property Value

[SizeUnit](#)

Methods

Render()

Renders the expression to CSS.

```
protected override string Render()
```

Returns

[string](#)

ToString()

Returns [Render\(\)](#) to ensure the expression is emitted as CSS.

```
public override string ToString()
```

Returns

[string](#)

Struct SizeUnit

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a CSS size value paired with a unit (px, rem, em, %, vw, vh).

```
public readonly struct SizeUnit
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Examples

```
var w1 = SizeUnit.Px(12); // "12px" SizeUnit w2 = 1.5; // "1.5px" via implicit conversion SizeUnit w3 =  
"2rem"; // "2rem" SizeExpression e = w1 + "2rem"; // calc(12px + 2rem) string css = e.ToString(); //  
"calc(12px + 2rem)"
```

Remarks

- Defaults to pixels (px) when created from numeric literals or when the unit is omitted in string literals.
- Arithmetic between two [SizeUnit](#) values produces a [SizeExpression](#) rendered as a CSS calc() expression, preserving units without attempting unit normalization at runtime.
- Arithmetic between a [SizeUnit](#) and a numeric value keeps the original unit and returns a new [Size Unit](#).

Constructors

SizeUnit(double, Unit)

Represents a CSS size value paired with a unit (px, rem, em, %, vw, vh).

```
public SizeUnit(double value, Unit unit)
```

Parameters

value [double](#)

unit [Unit](#)

Examples

```
var w1 = SizeUnit.Px(12); // "12px" SizeUnit w2 = 1.5; // "1.5px" via implicit conversion SizeUnit w3 = "2rem"; // "2rem" SizeExpression e = w1 + "2rem"; // calc(12px + 2rem) string css = e.ToString(); // "calc(12px + 2rem)"
```

Remarks

- Defaults to pixels (px) when created from numeric literals or when the unit is omitted in string literals.
- Arithmetic between two [SizeUnit](#) values produces a [SizeExpression](#) rendered as a CSS calc() expression, preserving units without attempting unit normalization at runtime.
- Arithmetic between a [SizeUnit](#) and a numeric value keeps the original unit and returns a new [SizeUnit](#).

Properties

Unit

The unit associated with the value.

```
public Unit Unit { get; }
```

Property Value

[Unit](#)

Value

The numeric value of the size (unit-less).

```
public double Value { get; }
```

Property Value

[double](#) ↗

Methods

Em(double)

Creates a size in em.

```
public static SizeUnit Em(double v)
```

Parameters

v [double](#)

Returns

[SizeUnit](#)

Percent(double)

Creates a size in percent.

```
public static SizeUnit Percent(double v)
```

Parameters

v [double](#)

Returns

[SizeUnit](#)

Px(double)

Creates a size in pixels.

```
public static SizeUnit Px(double v)
```

Parameters

v [double](#)

Returns

[SizeUnit](#)

Rem(double)

Creates a size in root-em.

```
public static SizeUnit Rem(double v)
```

Parameters

v [double](#)

Returns

[SizeUnit](#)

ToString()

Renders the size as a CSS literal (e.g., "12px", "1.5rem").

```
public override string ToString()
```

Returns

[string](#)

Vh(double)

Creates a size in viewport height.

```
public static SizeUnit Vh(double v)
```

Parameters

v [double](#) ↗

Returns

[SizeUnit](#)

Vw(double)

Creates a size in viewport width.

```
public static SizeUnit Vw(double v)
```

Parameters

v [double](#) ↗

Returns

[SizeUnit](#)

Operators

operator +(SizeUnit, SizeUnit)

Adds two size expressions, returning a CSS calc() expression.

```
public static SizeExpression operator +(SizeUnit a, SizeUnit b)
```

Parameters

a [SizeUnit](#)

b [SizeUnit](#)

Returns

[SizeExpression](#)

operator +(SizeUnit, double)

Adds a numeric value to a size, preserving the unit.

```
public static SizeUnit operator +(SizeUnit a, double b)
```

Parameters

a [SizeUnit](#)

b [double](#) ↗

Returns

[SizeUnit](#)

operator +(SizeUnit, int)

Adds a numeric value to a size, preserving the unit.

```
public static SizeUnit operator +(SizeUnit a, int b)
```

Parameters

a [SizeUnit](#)

b [int](#) ↗

Returns

[SizeUnit](#)

operator +(double, SizeUnit)

Adds a numeric value to a size, preserving the unit.

```
public static SizeUnit operator +(double a, SizeUnit b)
```

Parameters

a [double](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator +(int, SizeUnit)

Adds a numeric value to a size, preserving the unit.

```
public static SizeUnit operator +(int a, SizeUnit b)
```

Parameters

a [int](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator /(SizeUnit, SizeUnit)

Divides two size expressions, returning a CSS calc() expression.

```
public static SizeExpression operator /(SizeUnit a, SizeUnit b)
```

Parameters

a [SizeUnit](#)

b [SizeUnit](#)

Returns

[SizeExpression](#)

operator /(SizeUnit, double)

Divides a size by a double, preserving the unit.

```
public static SizeUnit operator /(SizeUnit a, double b)
```

Parameters

a [SizeUnit](#)

b [double](#) ↗

Returns

[SizeUnit](#)

operator /(SizeUnit, int)

Divides a size by an integer, preserving the unit.

```
public static SizeUnit operator /(SizeUnit a, int b)
```

Parameters

a [SizeUnit](#)

b [int](#) ↗

Returns

[SizeUnit](#)

operator /(double, SizeUnit)

Divides a double by a size value, preserving the size unit on the result value.

```
public static SizeUnit operator /(double a, SizeUnit b)
```

Parameters

a [double](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator /(int, SizeUnit)

Divides an integer by a size value, preserving the size unit on the result value.

```
public static SizeUnit operator /(int a, SizeUnit b)
```

Parameters

a [int](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

implicit operator string(SizeUnit)

Implicitly renders a [SizeUnit](#) to its CSS string representation.

```
public static implicit operator string(SizeUnit unit)
```

Parameters

unit [SizeUnit](#)

Returns

[string](#)

implicit operator SizeUnit(double)

Implicitly converts a double to a pixel size.

```
public static implicit operator SizeUnit(double val)
```

Parameters

val [double](#)

Returns

[SizeUnit](#)

implicit operator SizeUnit(int)

Implicitly converts an integer to a pixel size.

```
public static implicit operator SizeUnit(int val)
```

Parameters

val [int](#)

Returns

[SizeUnit](#)

implicit operator SizeUnit(string)

Implicitly parses a CSS size literal (e.g., "12px", "1.5rem", "50%"). Defaults to pixels if the unit is omitted.

```
public static implicit operator SizeUnit(string literal)
```

Parameters

literal [string](#)

Returns

[SizeUnit](#)

Exceptions

[FormatException](#)

Thrown when the literal is not recognized.

operator *(SizeUnit, SizeUnit)

Multiplies two size expressions, returning a CSS calc() expression.

```
public static SizeExpression operator *(SizeUnit a, SizeUnit b)
```

Parameters

a [SizeUnit](#)

b [SizeUnit](#)

Returns

[SizeExpression](#)

operator *(SizeUnit, double)

Multiplies a size by a double, preserving the unit.

```
public static SizeUnit operator *(SizeUnit a, double b)
```

Parameters

a [SizeUnit](#)

b [double](#) ↗

Returns

[SizeUnit](#)

operator *(SizeUnit, int)

Multiplies a size by an integer, preserving the unit.

```
public static SizeUnit operator *(SizeUnit a, int b)
```

Parameters

a [SizeUnit](#)

b [int](#) ↗

Returns

[SizeUnit](#)

operator *(double, SizeUnit)

Multiplies a size by a double, preserving the unit.

```
public static SizeUnit operator *(double a, SizeUnit b)
```

Parameters

a [double](#) ↗

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator *(int, SizeUnit)

Multiplies a size by an integer, preserving the unit.

```
public static SizeUnit operator *(int a, SizeUnit b)
```

Parameters

a [int](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator -(SizeUnit, SizeUnit)

Subtracts two size expressions, returning a CSS calc() expression.

```
public static SizeExpression operator -(SizeUnit a, SizeUnit b)
```

Parameters

a [SizeUnit](#)

b [SizeUnit](#)

Returns

[SizeExpression](#)

operator -(SizeUnit, double)

Subtracts a numeric value from a size, preserving the unit.

```
public static SizeUnit operator -(SizeUnit a, double b)
```

Parameters

a [SizeUnit](#)

b [double](#) ↗

Returns

[SizeUnit](#)

operator -(SizeUnit, int)

Subtracts a numeric value from a size, preserving the unit.

```
public static SizeUnit operator -(SizeUnit a, int b)
```

Parameters

a [SizeUnit](#)

b [int](#) ↗

Returns

[SizeUnit](#)

operator -(double, SizeUnit)

Subtracts a size value from a numeric value, preserving the size unit on the result value.

```
public static SizeUnit operator -(double a, SizeUnit b)
```

Parameters

a [double](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

operator -(int, SizeUnit)

Subtracts a size value from a numeric value, preserving the size unit on the result value.

```
public static SizeUnit operator -(int a, SizeUnit b)
```

Parameters

a [int](#)

b [SizeUnit](#)

Returns

[SizeUnit](#)

Struct Spacing

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Represents a CSS spacing value, which can be a numeric value and unit or the keyword **auto**. Supported units: [Px](#), [Em](#), [Rem](#), [Percent](#), [Vw](#), [Vh](#).

```
public readonly struct Spacing : IEquatable<Spacing>
```

Implements

[IEquatable](#) <[Spacing](#)>

Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Auto

Gets a spacing representing the CSS keyword **auto**.

```
public static readonly Spacing Auto
```

Field Value

[Spacing](#)

Zero

Gets a spacing of zero pixels.

```
public static readonly Spacing Zero
```

Field Value

Properties

IsAuto

Gets a value indicating whether this spacing represents the CSS keyword `auto`.

```
public bool IsAuto { get; }
```

Property Value

[bool](#)

Unit

Gets the unit of measure. Meaningful only when [IsAuto](#) is `false`.

```
public Unit Unit { get; }
```

Property Value

[Unit](#)

Value

Gets the numeric value. Meaningful only when [IsAuto](#) is `false`.

```
public double Value { get; }
```

Property Value

[double](#)

Methods

Abs()

Returns a spacing with the absolute numeric value of this spacing.

```
public Spacing Abs()
```

Returns

[Spacing](#)

Clamp(Spacing, Spacing, Spacing)

Clamps a spacing between `min` and `max` (same unit required).

```
public static Spacing Clamp(Spacing value, Spacing min, Spacing max)
```

Parameters

`value` [Spacing](#)

`min` [Spacing](#)

`max` [Spacing](#)

Returns

[Spacing](#)

Em(double)

Creates an em spacing.

```
public static Spacing Em(double v)
```

Parameters

`v` [double](#) ↗

The value in em.

Returns

[Spacing](#)

Equals(Spacing)

Indicates whether the current object is equal to another [Spacing](#).

```
public bool Equals(Spacing other)
```

Parameters

other [Spacing](#)

Returns

[bool](#)

Equals(object?)

Determines whether the specified object is equal to the current [Spacing](#).

```
public override bool Equals(object? obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

GetHashCode()

Returns a hash code for the spacing.

```
public override int GetHashCode()
```

Returns

[int](#)

Max(Spacing, Spacing)

Returns the maximum of two spacing values (same unit required).

```
public static Spacing Max(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b [Spacing](#)

Returns

[Spacing](#)

Min(Spacing, Spacing)

Returns the minimum of two spacing values (same unit required).

```
public static Spacing Min(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b [Spacing](#)

Returns

[Spacing](#)

Parse(string)

Parses a CSS spacing literal (e.g., `10px`, `1.5rem`, `25%`, `auto`). If unit is omitted, pixels are assumed.

```
public static Spacing Parse(string text)
```

Parameters

`text` [string](#)

The input text.

Returns

[Spacing](#)

The parsed [Spacing](#).

Exceptions

[ArgumentNullException](#)

If `text` is null.

[FormatException](#)

If the text cannot be parsed or the unit is unknown.

Px(double)

Creates a pixel spacing.

```
public static Spacing Px(double v)
```

Parameters

`v` [double](#)

The value in pixels.

Returns

[Spacing](#)

Rem(double)

Creates a rem spacing.

```
public static Spacing Rem(double v)
```

Parameters

v [double](#)

The value in rem.

Returns

[Spacing](#)

ToString()

Returns the CSS string representation of the spacing (e.g., `10px`, `1.5rem`, `auto`).

```
public override string ToString()
```

Returns

[string](#)

Vh(double)

Creates a viewport height spacing.

```
public static Spacing Vh(double v)
```

Parameters

v [double](#)

The value in vh.

Returns

[Spacing](#)

Vw(double)

Creates a viewport width spacing.

```
public static Spacing Vw(double v)
```

Parameters

v [double](#)

The value in vw.

Returns

[Spacing](#)

Operators

operator +(Spacing, Spacing)

Adds two spacing values with the same unit.

```
public static Spacing operator +(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b [Spacing](#)

Returns

[Spacing](#)

operator /(Spacing, double)

Divides a spacing by a scalar.

```
public static Spacing operator /(Spacing a, double k)
```

Parameters

a [Spacing](#)

k [double](#) ↗

Returns

[Spacing](#)

operator ==(Spacing, Spacing)

Determines whether two [Spacing](#) instances are equal.

```
public static bool operator ==(Spacing left, Spacing right)
```

Parameters

left [Spacing](#)

right [Spacing](#)

Returns

[bool](#) ↗

operator >(Spacing, Spacing)

Returns [true](#) if a is greater than b (same unit required).

```
public static bool operator >(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b [Spacing](#)

Returns

[bool](#)

operator >=(Spacing, Spacing)

Returns [true](#) if a is greater than or equal to b (same unit required).

```
public static bool operator >=(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b [Spacing](#)

Returns

[bool](#)

implicit operator string(Spacing)

Implicitly converts the spacing to its CSS string representation.

```
public static implicit operator string(Spacing s)
```

Parameters

s [Spacing](#)

The spacing.

Returns

[string](#)

implicit operator Spacing(double)

Implicitly converts a [double](#) value to a pixel spacing.

```
public static implicit operator Spacing(double px)
```

Parameters

`px` [double](#)

The pixel value.

Returns

[Spacing](#)

implicit operator Spacing(int)

Implicitly converts an [int](#) value to a pixel spacing.

```
public static implicit operator Spacing(int px)
```

Parameters

`px` [int](#)

The pixel value.

Returns

[Spacing](#)

implicit operator Spacing(string)

Implicitly parses a CSS spacing literal (e.g., "10px", "1.5rem", "auto").

```
public static implicit operator Spacing(string literal)
```

Parameters

`literal` [string](#)

The CSS spacing literal.

Returns

[Spacing](#)

Exceptions

[ArgumentNullException](#)

Thrown when `literal` is null.

[FormatException](#)

Thrown when the literal cannot be parsed.

operator !=(Spacing, Spacing)

Determines whether two [Spacing](#) instances are not equal.

```
public static bool operator !=(Spacing left, Spacing right)
```

Parameters

`left` [Spacing](#)

`right` [Spacing](#)

Returns

[bool](#)

operator <(Spacing, Spacing)

Returns `true` if `a` is less than `b` (same unit required).

```
public static bool operator <(Spacing a, Spacing b)
```

Parameters

`a` [Spacing](#)

`b` [Spacing](#)

Returns

[bool](#) ↗

operator <=(Spacing, Spacing)

Returns `true` if `a` is less than or equal to `b` (same unit required).

```
public static bool operator <=(Spacing a, Spacing b)
```

Parameters

`a` [Spacing](#)

`b` [Spacing](#)

Returns

[bool](#) ↗

operator *(Spacing, double)

Multiplies a spacing by a scalar.

```
public static Spacing operator *(Spacing a, double k)
```

Parameters

a [Spacing](#)

k [double](#)

Returns

[Spacing](#)

operator *(double, Spacing)

Multiplies a spacing by a scalar.

```
public static Spacing operator *(double k, Spacing a)
```

Parameters

k [double](#)

a [Spacing](#)

Returns

[Spacing](#)

operator -(Spacing, Spacing)

Subtracts two spacing values with the same unit.

```
public static Spacing operator -(Spacing a, Spacing b)
```

Parameters

a [Spacing](#)

b Spacing

Returns

Spacing

Class Spacings

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

Helper methods to compose spacing arrays in the order: top, right, bottom, left. Useful for mapping to CSS shorthand properties (e.g., margin/padding).

```
public static class Spacings
```

Inheritance

[object](#) ← Spacings

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

All(Spacing)

Applies the same spacing to all sides.

```
public static Spacing[] All(Spacing s)
```

Parameters

s [Spacing](#)

Spacing value.

Returns

[Spacing\[\]](#)

An array [top, right, bottom, left].

Bottom(Spacing)

Applies spacing only to the bottom side.

```
public static Spacing[] Bottom(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing](#)[]

Horizontal(Spacing)

Applies spacing to left and right; top and bottom are zero.

```
public static Spacing[] Horizontal(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing](#)[]

Left(Spacing)

Applies spacing only to the left side.

```
public static Spacing[] Left(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing\[\]](#)

Only(Spacing?, Spacing?, Spacing?, Spacing?)

Applies spacing only to the specified sides; unspecified sides default to zero.

```
public static Spacing[] Only(Spacing? top = null, Spacing? right = null, Spacing? bottom = null, Spacing? left = null)
```

Parameters

top [Spacing?](#)

Optional top spacing.

right [Spacing?](#)

Optional right spacing.

bottom [Spacing?](#)

Optional bottom spacing.

left [Spacing?](#)

Optional left spacing.

Returns

[Spacing\[\]](#)

An array [top, right, bottom, left].

Right(Spacing)

Applies spacing only to the right side.

```
public static Spacing[] Right(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing](#)[]

Symmetric(Spacing, Spacing)

Applies separate vertical and horizontal spacing.

```
public static Spacing[] Symmetric(Spacing vertical, Spacing horizontal)
```

Parameters

vertical [Spacing](#)

Top and bottom spacing.

horizontal [Spacing](#)

Left and right spacing.

Returns

[Spacing](#)[]

Top(Spacing)

Applies spacing only to the top side.

```
public static Spacing[] Top(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing\[\]](#)

Vertical(Spacing)

Applies spacing to top and bottom; left and right are zero.

```
public static Spacing[] Vertical(Spacing s)
```

Parameters

s [Spacing](#)

Returns

[Spacing\[\]](#)

Enum Unit

Namespace: [RTB.Blazor.Styled.Helper](#)

Assembly: RTB.Blazor.Styled.dll

CSS related enums and extensions.

```
public enum Unit
```

Extension Methods

[CssEnumExtensions.ToCss\(Enum\)](#)

Fields

Em = 2

Em

Percent = 3

Percent

Px = 0

Pixels

Rem = 1

Root em

Vh = 5

Viewport height

Vw = 4

Viewport width

Namespace RTB.Blazor.Styled.Services

Classes

[StyleRegistry](#)

Registry for scoped CSS

Interfaces

[IStyleRegistry](#)

Registry for scoped CSS

Interface IStyleRegistry

Namespace: [RTB.Blazor.Styled.Services](#)

Assembly: RTB.Blazor.Styled.dll

Registry for scoped CSS

```
public interface IStyleRegistry
```

Methods

Acquire(string?)

Acquire (or create) a stable class name. If null, a GUID-based name is generated.

```
string Acquire(string? preferredClassName = null)
```

Parameters

preferredClassName [string](#)

Returns

[string](#)

ClearAll()

Clear all injected styles and reset the registry.

```
ValueTask ClearAll()
```

Returns

[ValueTask](#)

GenerateClassName(string)

Helper to generate a short, stable-looking class name.

```
string GenerateClassName(string prefix = "rtb-")
```

Parameters

`prefix` [string](#)

Returns

[string](#)

Release(string)

Release one reference; when it reaches zero, the rules are cleared and the class entry is removed.

```
ValueTask<bool> Release(string className)
```

Parameters

`className` [string](#)

Returns

[ValueTask](#)<[bool](#)>

UpsertScopedAsync(string, string)

Insert/update the fully-scoped CSS for the class. No-op if CSS unchanged.

```
ValueTask UpsertScopedAsync(string scopedCss, string className)
```

Parameters

`scopedCss` [string](#)

`className` [string](#)

Returns

[ValueTask](#)

Class StyleRegistry

Namespace: [RTB.Blazor.Styled.Services](#)

Assembly: RTB.Blazor.Styled.dll

Registry for scoped CSS

```
public sealed class StyleRegistry : IStyleRegistry
```

Inheritance

[object](#) ← StyleRegistry

Implements

[IStyleRegistry](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

StyleRegistry(IJSRuntime)

Registry for scoped CSS

```
public StyleRegistry(IJSRuntime js)
```

Parameters

js [IJSRuntime](#)

Methods

Acquire(string?)

Acquire (or create) a stable class name. If null, a GUID-based name is generated.

```
public string Acquire(string? preferredClassName = null)
```

Parameters

`preferredClassName` [string](#)

Returns

[string](#)

ClearAll()

Clear all injected styles and reset the registry.

```
public ValueTask ClearAll()
```

Returns

[ValueTask](#)

GenerateClassName(string)

Helper to generate a short, stable-looking class name.

```
public string GenerateClassName(string prefix = "rtb-")
```

Parameters

`prefix` [string](#)

Returns

[string](#)

Release(string)

Release one reference; when it reaches zero, the rules are cleared and the class entry is removed.

```
public ValueTask<bool> Release(string className)
```

Parameters

className [string](#)

Returns

[ValueTask](#) <[bool](#)>

UpsertScopedAsync(string, string)

Insert/update the fully-scoped CSS for the class. No-op if CSS unchanged.

```
public ValueTask UpsertScopedAsync(string scopedCss, string className)
```

Parameters

scopedCss [string](#)

className [string](#)

Returns

[ValueTask](#)

Namespace RTB.Blazor.Styles

Classes

[ButtonStyle](#)

Style model for buttons, extending [TextStyle](#) with button-specific colors.

[PreStyled](#)

A minimal [RTBStyleBase](#) implementation intended as a placeholder for pre-styled or externally composed styles.

[TabStyle](#)

Tab style settings.

[TextStyle](#)

Style class for text-related CSS properties.

Interfaces

[IStyle](#)

Defines a typed style that can be converted into a [StyleBuilder](#) for CSS emission.

Class ButtonStyle

Namespace: [RTB.Blazor.Styles](#)

Assembly: RTB.Blazor.dll

Style model for buttons, extending [TextStyle](#) with button-specific colors.

```
public class ButtonStyle : TextStyle, IStyle
```

Inheritance

[object](#) ← [TextStyle](#) ← ButtonStyle

Implements

[IStyle](#)

Inherited Members

[TextStyle.FontSize](#) , [TextStyle.FontWeight](#) , [TextStyle.LineHeight](#) , [TextStyle.Color](#) ,
[TextStyle.TextDecoration](#) , [TextStyle.TextAlign](#) , [TextStyle.WithColor\(RTBColor?\)](#) , [TextStyle.ToStyle\(\)](#) ,
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Examples

Example of using ButtonStyle with StyleBuilder:

```
var bs = new ButtonStyle
{
    BackgroundColor = RTBColor.FromRgb(33, 150, 243), // blue
    DisabledBackgroundColor = RTBColor.FromRgb(189, 189, 189), // grey
    DisabledColor = RTBColor.FromRgb(255, 255, 255) // white
}.WithColor(RTBColor.FromRgb(255, 255, 255)); // text color for normal state

var sb = bs.ToStyle()
    .SetIf("background-color", bs.BackgroundColor?.HexRgba, bs.BackgroundColor.HasValue)
    .SetIf("color", bs.Color?.HexRgba, bs.Color.HasValue)
    .Selector("&:disabled", b => b
        .SetIf("background-color", bs.DisabledBackgroundColor?.HexRgba,
        bs.DisabledBackgroundColor.HasValue)
        .SetIf("color", bs.DisabledColor?.HexRgba, bs.DisabledColor.HasValue));

```

Remarks

This type is a passive data model: it does not emit CSS by itself. Consumers (e.g., Blazor components) should translate its properties into CSS using [StyleBuilder](#) or similar mechanisms. A `null` property indicates that no explicit value should be emitted and theming/defaults should apply.

Properties

BackgroundColor

Background color for the button in its normal (enabled) state.

```
public RTBColor? BackgroundColor { get; set; }
```

Property Value

[RTBColor?](#)

Remarks

Use helpers like [FromRgb\(byte, byte, byte\)](#) or [Parse\(string\)](#) to construct values.

DisabledBackgroundColor

Background color for the button when it is disabled.

```
public RTBColor? DisabledBackgroundColor { get; set; }
```

Property Value

[RTBColor?](#)

Remarks

If `null`, consumers may fall back to [BackgroundColor](#) or theme defaults.

DisabledColor

Foreground text color to use when the button is disabled.

```
public RTBColor? DisabledColor { get; set; }
```

Property Value

[RTBColor?](#)

Remarks

If `null`, consumers should avoid emitting an explicit disabled text color and rely on default styles or theming.

Interface IStyle

Namespace: [RTB.Blazor.Styles](#)

Assembly: RTB.Blazor.dll

Defines a typed style that can be converted into a [StyleBuilder](#) for CSS emission.

```
public interface IStyle
```

Examples

Example:

```
public sealed class ButtonStyle : IStyle
{
    public bool Primary { get; init; }

    public StyleBuilder ToStyle()
    {
        var sb = StyleBuilder.Start
            .Set("padding", "0.5rem 1rem")
            .Set("border", "none")
            .Set("border-radius", "0.375rem")
            .SetIf("background", "dodgerblue", Primary)
            .SetIf("color", "white", Primary);

        // Nested selector relative to the component's scope
        sb.Selector("&:hover", b => b.Set("filter", "brightness(0.95)"));
        return sb;
    }
}
```

Remarks

Implementations typically translate component or application state into CSS declarations, selectors, groups (e.g., `@media`), and keyframes using [StyleBuilder](#). Consumers call [ToStyle\(\)](#) to obtain a builder, which can then be composed and emitted as scoped CSS where appropriate.

Methods

ToStyle()

Creates a [StyleBuilder](#) representing this style.

`StyleBuilder ToStyle()`

Returns

[StyleBuilder](#)

A configured [StyleBuilder](#) instance to be composed and emitted as CSS.

Remarks

Prefer returning a newly created builder per call to keep results deterministic and side-effect free.

See Also

[StyleBuilder](#)

Class PreStyled

Namespace: [RTB.Blazor.Styles](#)

Assembly: RTB.Blazor.dll

A minimal [RTBStyleBase](#) implementation intended as a placeholder for pre-styled or externally composed styles.

```
public class PreStyled : RTBStyleBase, IComponent, IHandleEvent, IHandleAfterRender,  
IStyleContributor, IAsyncDisposable
```

Inheritance

[object](#) ← [ComponentBase](#) ← [RTBStyleBase](#) ← PreStyled

Implements

[IComponent](#), [IHandleEvent](#), [IHandleAfterRender](#), [IStyleContributor](#), [IAsyncDisposable](#)

Inherited Members

[RTBStyleBase.StyleBuilder](#), [RTBStyleBase.Condition](#), [RTBStyleBase.OnInitialized\(\)](#),
[RTBStyleBase.OnParametersSet\(\)](#), [RTBStyleBase.DisposeAsync\(\)](#),
[ComponentBase.BuildRenderTree\(RenderTreeBuilder\)](#), [ComponentBase.OnInitializedAsync\(\)](#),
[ComponentBase.OnParametersSetAsync\(\)](#), [ComponentBase.StateHasChanged\(\)](#),
[ComponentBase.ShouldRender\(\)](#), [ComponentBase.OnAfterRender\(bool\)](#),
[ComponentBase.OnAfterRenderAsync\(bool\)](#), [ComponentBase.InvokeAsync\(Action\)](#),
[ComponentBase.InvokeAsync\(Func<Task>\)](#), [ComponentBase.DispatchExceptionAsync\(Exception\)](#),
[ComponentBase.SetParametersAsync\(ParameterView\)](#), [ComponentBase.RendererInfo](#),
[ComponentBase.Assets](#), [ComponentBase.AssignedRenderMode](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#), [object.ToString\(\)](#)

Examples

Usage as a placeholder within a style scope:

Remarks

- This component currently does not contribute any styles; [BuildStyle\(StyleBuilder\)](#) is a no-op.
- It can be used as a stable anchor in the render tree where a future style contribution may be injected or where a derived component can override behavior.

- The optional [Style](#) parameter is reserved for scenarios where an [IStyle](#) may later be absorbed into the cascading [StyleBuilder](#). In this implementation it is not consumed.

Properties

Style

An optional, pre-built style instance that could be forwarded or absorbed into the current style composition. Not used in this implementation.

```
[Parameter]  
public IStyle? Style { get; set; }
```

Property Value

[IStyle](#)

Methods

BuildStyle(StyleBuilder)

Implementors define style declarations and nested rules here.

```
protected override void BuildStyle(StyleBuilder builder)
```

Parameters

builder [StyleBuilder](#)

The cascading [StyleBuilder](#).

Class TabStyle

Namespace: [RTB.Blazor.Styles](#)

Assembly: RTB.Blazor.dll

Tab style settings.

```
public class TabStyle : IStyle
```

Inheritance

[object](#) ← TabStyle

Implements

[IStyle](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Color

The color of the tab text.

```
public RTBColor? Color { get; set; }
```

Property Value

[RTBColor?](#)

Methods

ToStyle()

Builds the style.

```
public StyleBuilder ToStyle()
```

Returns

[StyleBuilder](#)

Class TextStyle

Namespace: [RTB.Blazor.Styles](#)

Assembly: RTB.Blazor.dll

Style class for text-related CSS properties.

```
public class TextStyle : IStyle
```

Inheritance

[object](#) ← TextStyle

Implements

[IStyle](#)

Derived

[ButtonStyle](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Color

CSS color property.

```
public RTBColor? Color { get; set; }
```

Property Value

[RTBColor?](#)

FontSize

CSS font-size property.

```
public string? FontSize { get; set; }
```

Property Value

[string](#)

FontWeight

CSS font-weight property.

```
public string? FontWeight { get; set; }
```

Property Value

[string](#)

LineHeight

CSS line-height property.

```
public string? LineHeight { get; set; }
```

Property Value

[string](#)

TextAlign

CSS text-align property.

```
public Flex.Align? TextAlign { get; set; }
```

Property Value

[Flex.Align?](#)

TextDecoration

CSS text-decoration property.

```
public string? TextDecoration { get; set; }
```

Property Value

[string](#)

Methods

ToStyle()

Converts the TextStyle properties to a StyleBuilder instance.

```
public virtual StyleBuilder ToStyle()
```

Returns

[StyleBuilder](#)

WithColor(RTBColor?)

Sets the Color property and returns the current instance for chaining.

```
public TextStyle WithColor(RTBColor? color)
```

Parameters

color [RTBColor](#)?

Returns

[TextStyle](#)