Diagrams (Blzor)

# Quick Start

 Last updated: October 17th, 2020

# Installation

### Package installation

En primer lugar, deberá instalar el **paquete NuGet mediante el Administrador de paquetes de Visual Studio** o con los siguientes comandos:

#### Package Manager Console

Install-Package Z.Blazor.Diagrams

#### CLI

dotnet add package Z.Blazor.Diagrams

### Project setup

**Blazor.Diagrams necesita** algo de **JavaScript** y **CSS** para funcionar correctamente, agreguémoslos:

**<!-- in the head element -->**

**<link href="\_content/Z.Blazor.Diagrams/style.min.css" rel="stylesheet" />**

**<!-- if you want the default styling -->**

**<link href="\_content/Z.Blazor.Diagrams/default.styles.min.css" rel="stylesheet" />**

**<!-- in the body element -->**

**<script src="\_content/Z.Blazor.Diagrams/script.min.js"></script>**

#### Location (ubicación)

Si utilizamos la renderización del lado del Servidor (**Server Side Rendering**), debemos añadir el código anterior en el archivo **Pages/\_Host.cshtml**, si por el contrario es **Blazor** **WASM** su lugar es el archivo **wwwroot/index.html**.

### Diagram creation

### Diagram

Actualmente **Blazor.Diagrams** está muy orientado al código de objetos, lo que significa que toda la información (configuración) del diagrama debe ser proporcionada mediante código. Esto tiene algunos beneficios, como la fácil (des)serialización.

### Creating a diagram manager

Every diagram has a **Diagram**, go ahead and create one in the component where you want to show it:

Cada diagrama tiene una clase **Diagram**, defínalo y crer un objeto de este tipo dentro del componente donde desea mostrarlo:

**private Diagram Diagram { get; private set; }**

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**Diagram = new Diagram();**

**}**

### Changing options

Cada objeto **Diagram** dispone de una lista de opciones que podemos cambiar:

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**var options = new DiagramOptions**

**{**

**DeleteKey = "Delete", // What key deletes the selected nodes/links**

**DefaultNodeComponent = null, // Default component for nodes**

**AllowMultiSelection = true, // Whether to allow multi selection using CTRL**

**Links = new DiagramLinkOptions**

**{**

**// Options related to links**

**},**

**Zoom = new DiagramZoomOptions**

**{**

**Minimum = 0.5, // Minimum zoom value**

**Inverse = false, // Whether to inverse the direction of the zoom when using the wheel**

**// Other**

**}**

**};**

**Diagram = new Diagram(options);**

**}**

### Nodes

Para una mejor comprensión de este tutorial, agregaremos algunos **nodos** manualmente. Por lo general, estos se crearían a partir de los datos recibidos, por ejemplo, del **backend**

Tenga en cuenta que los **nodos** creados **tienen 4 puertos**, la alineación solo importa si está utilizando el estilo predeterminado, de lo contrario depende de usted decidir qué hacer con él.

**private void Setup()**

**{**

**var node1 = NewNode(50, 50);**

**var node2 = NewNode(300, 300);**

**var node3 = NewNode(300, 50);**

**Diagram.Nodes.Add(node1, node2, node3);**

**}**

**private NodeModel NewNode(double x, double y)**

**{**

**var node = new NodeModel(new Point(x, y));**

**node.AddPort(PortAlignment.Bottom);**

**node.AddPort(PortAlignment.Top);**

**node.AddPort(PortAlignment.Left);**

**node.AddPort(PortAlignment.Right);**

**return node;**

**}**

Avancemos y añadimos un método **Setup();** en le método **OnInitialized**

### Links

En **Blazor.Diagrams**, se pueden agregar vínculos desde un puerto de origen a un puerto de destino.  
Agreguemos uno:

**private void Setup()**

**{**

**var node1 = NewNode(50, 50);**

**var node2 = NewNode(300, 300);**

**var node3 = NewNode(300, 50);**

**Diagram.Nodes.Add(new [] { node1, node2, node3 });**

**Diagram.Links.Add(new LinkModel(node1.GetPort(PortAlignment.Right), node2.GetPort(PortAlignment.Left)));**

**}**

El método **AddLink** agrega un vínculo de tipo **LinkModel**, que es el tipo de vínculo predeterminado. También devuelve el link creado, por lo que podría obtenerlo y cambiar, por ejemplo, su tipo a una línea.

## Display

Finalmente, ¡mostremos el diagrama!

### \_Imports.razor

**@using Blazor.Diagrams.Core;**

**@using Blazor.Diagrams.Core.Models;**

**@using Blazor.Diagrams.Components;**

#### Adding the canvas

**<!-- Add this where you want to show the diagram -->**

**<CascadingValue Value="Diagram">**

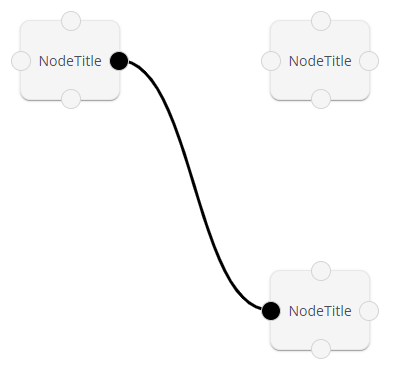
**<DiagramCanvas></DiagramCanvas>**

**</CascadingValue>**

#### Pay attention

When adding a **DiagramCanvas**, its parent needs to have an actual width/height, since all the layers in **Blazor.Diagrams** use **width: 100%; height: 100%** for them to be able to work properly.

Si hizo todo correctamente y usó los estilos predeterminados, debería ver esto:



# Customization

**Last updated: August 14th, 2020**

## Custom nodes

**En Z.Blazor.Diagrams**, todo es un **Modelo**, esto hace que sea más fácil manejar la (des)serialización y la personalización por herencia.

Para personalizar los nodos, tendrá que crear un nuevo modelo, así como el componente **Blazor** para él.

### Creating a model

Primero, crearemos un nuevo **modelo** para nuestro **nodo**, que contendrá todos los datos necesarios. Nuestro **nodo** personalizado se utilizará para sumar dos números.

**public class AddTwoNumbersNode : NodeModel**

**{**

**public AddTwoNumbersNode(Point position = null) : base(position) { }**

**public double FirstNumber { get; set; }**

**public double SecondNumber { get; set; }**

**// Here, you can put whatever you want, such as a method that does the addition**

**}**

### Creating a component

En segundo lugar, crearemos un componente **AddTwoNumbersComponent** que maneje nuestro nuevo tipo de **nodos**.

**@using Blazor.Diagrams.Components.Renderers;**

**<div class="card" style="width: 250px;">**

**<div class="card-body">**

**<h5 class="card-title">Add</h5>**

**<div class="form-group">**

**<input type="number" class="form-control" @bind-value="Node.FirstNumber" placeholder="Number 1" />**

**</div>**

**<div class="form-group">**

**<input type="number" class="form-control" @bind-value="Node.SecondNumber" placeholder="Number 2" />**

**</div>**

**</div>**

**@foreach (var port in Node.Ports)**

**{**

**<!-- Default ports -->**

**<PortRenderer Port="port" />**

**}**

**</div>**

**@code {**

**[Parameter]**

**public BotAnswerNode Node { get; set; }**

**}**

1. El componente se envolverá en un elemento **div.node** para posicionarlo.
2. Siempre es responsabilidad nuestra renderizar los puertos (You are always in charge of rendering the ports), y debes usar **PortRenderer**.
3. El **BotAnswerNode Node** vendrá del padre.

### Registering the custom node

Por último, registraremos el nodo personalizado y su componente en el **Diagram**:

**private Diagram Diagram { get; private set; }**

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**Diagram = new Diagram();**

**Diagram.RegisterModelComponent<AddTwoNumbersNode, AddTwoNumbersComponent>();**

**}**

Esto indica al **administrador de Diagram** que represente un **AddTwoNumbersComponent** para cada nodo de tipo **AddTwoNumbersNode**.

## Custom ports

Para personalizar los puertos, solo necesitará crear un nuevo modelo.  
Dado que los nodos controlan sus puertos, depende de ellos diseñar los puertos.

### Creating a model

Primero, crearemos un nuevo modelo para nuestro puerto, que contendrá todos los datos necesarios.

**public class ColoredPort : PortModel**

**{**

**public ColoredPort(NodeModel parent, PortAlignment alignment, bool isRed) : base(parent, alignment, null, null)**

**{**

**IsRed = isRed;**

**}**

**public bool IsRed { get; set; }**

**public override bool CanAttachTo(PortModel port)**

**{**

**// Checks for same-node/port attachements**

**if (!base.CanAttachTo(port))**

**return false;**

**// Only able to attach to the same port type**

**if (!(port is ColoredPort cp))**

**return false;**

**return IsRed == cp.IsRed;**

**}**

**}**

Como puede ver en el ejemplo, podemos sobrescribir el método **CanAttachTo** para agregar comprobaciones adicionales.

### Styling the custom ports

En segundo lugar, crearemos un componente de nodo personalizado **ColoredNodeWidget** que nos ayudará a diseñar nuestros puertos personalizados.

**<div class="colored-node @(Node.Selected ? " selected" : "")">**

**<div>N</div>**

**@foreach (var port in Node.Ports.Cast<ColoredPort>()) // Notice the cast**

**{**

**<PortRenderer Port="port" Class="@(port.IsRed ? "red" : "blue")">**

**@\* You can put custom content here (e.g. custom components) \*@**

**</PortRenderer>**

**}**

**</div>**

**@code {**

**[Parameter]**

**public NodeModel Node { get; set; }**

**}**

And for the CSS:

**.colored-node {**

**border: 2px solid black;**

**text-align: center;**

**}**

**.colored-node.selected {**

**border-color: blue;**

**}**

**.colored-node .port {**

**width: 30px;**

**height: 30px;**

**}**

**.colored-node .port.blue {**

**background-color: blue;**

**}**

**.colored-node .port.red {**

**background-color: red;**

**}**

### Using the custom ports

Por último, vamos a poner todo junto:

**private readonly Diagram \_diagram = new Diagram();**

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**\_diagram.Options.DefaultNodeComponent = typeof(ColoredNodeWidget);**

**var node1 = NewNode(50, 50);**

**var node2 = NewNode(300, 300);**

**\_diagram.Nodes.Add(node1);**

**\_diagram.Nodes.Add(node2);**

**\_diagram.Nodes.Add(NewNode(500, 50));**

**\_diagram.Links.Add(new LinkModel(node1.GetPort(PortAlignment.Top), node2.GetPort(PortAlignment.Top)));**

**}**

**private NodeModel NewNode(double x, double y)**

**{**

**var node = new NodeModel(new Point(x, y));**

**node.AddPort(new ColoredPort(node, PortAlignment.Top, true));**

**node.AddPort(new ColoredPort(node, PortAlignment.Left, false));**

**return node;**

**}**

Como puede ver, puede crear una instancia de sus puertos personalizados como desee y simplemente pasarlos a **AddPort**.

## Custom links

La creación de enlaces personalizados es más o menos lo mismo que los nodos personalizados.

### Creating a model

Primero, crearemos un nuevo modelo para nuestro enlace.

**public class ThickLink : LinkModel**

**{**

**public ThickLink(PortModel sourcePort, PortModel targetPort = null) : base(sourcePort, targetPort) { }**

**// Add relevant data here if you need to**

**}**

In **Z.Blazor.Diagrams**, links only hold information such as the source and target ports for now.

### Creating a component

En segundo lugar, crearemos un componente **ThickLinkWidget** que maneje nuestro nuevo tipo de enlaces.

**@code {**

**[Parameter]**

**public LinkModel Link { get; set; }**

**}**

**<line x1="@Link.GetMiddleSourceX().ToInvariantString()"**

**y1="@Link.GetMiddleSourceY().ToInvariantString()"**

**x2="@Link.GetTargetX().ToInvariantString()"**

**y2="@Link.GetTargetY().ToInvariantString()"**

**stroke="@(Link.Selected ? "red" : "blue")"**

**stroke-width="6" />**

* Todos los métodos **GetX** son métodos de extensión disponibles en **Blazor.Diagrams.Core.Extensions.LinkModelExtensions**.
* El método **ToInvariantString** se utiliza para asegurarse de que los números tienen formato con un punto.

### Registering the custom link

Por último, registraremos el enlace personalizado y su componente en el **Diagram**:

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**\_diagram.RegisterModelComponent<ThickLink, ThickLinkWidget>();**

**// Also usable: \_diagram.Options.Links.DefaultLinkComponent = typeof(ThickLink);**

**var node1 = NewNode(50, 50);**

**var node2 = NewNode(300, 300);**

**var node3 = NewNode(500, 50);**

**\_diagram.Nodes.Add(node1);**

**\_diagram.Nodes.Add(node2);**

**\_diagram.Nodes.Add(node3);**

**\_diagram.Links.Add(new ThickLink(node1.GetPort(PortAlignment.Right), node2.GetPort(PortAlignment.Left)));**

**\_diagram.Links.Add(new ThickLink(node2.GetPort(PortAlignment.Right), node3.GetPort(PortAlignment.Left)));**

**}**

**private NodeModel NewNode(double x, double y)**

**{**

**var node = new NodeModel(new Point(x, y));**

**node.AddPort(PortAlignment.Bottom);**

**node.AddPort(PortAlignment.Top);**

**node.AddPort(PortAlignment.Left);**

**node.AddPort(PortAlignment.Right);**

**return node;**

**}**

Como puede ver, puede establecer la opción **DefaultLinkComponent** para que cada nuevo vínculo se represente como **ThickLinkWidget** o especificar el tipo de vínculo cada vez que llame a **Diagram.AddLink**.

## Custom groups

Crear grupos personalizados es casi lo mismo que crear nodos personalizados, excepto por algunos detalles.

### Creating a model

Primero, crearemos un nuevo modelo para nuestro grupo.

**public class CustomGroupModel : GroupModel**

**{**

**public CustomGroupModel(Diagram diagram, NodeModel[] children, string title, byte padding = 30)**

**: base(diagram, children, padding)**

**{**

**Title = title;**

**}**

**public string Title { get; }**

**}**

Puede agregar las propiedades que necesite. También puede cambiar el valor del relleno, que se utiliza al calcular la posición/tamaño del grupo.

### Creating a component

En segundo lugar, crearemos un componente **CustomGroupWidget** que maneje nuestro nuevo tipo de enlaces.

**@using Blazor.Diagrams.Components.Renderers;**

**<GroupContainer Group="Group" Class="custom">**

**<span class="title">@Group.Title</span>**

**<GroupLinks></GroupLinks>**

**<GroupNodes></GroupNodes>**

**@foreach (var port in Group.Ports)**

**{**

**<PortRenderer Port="port" Class="group-port"></PortRenderer>**

**}**

**</GroupContainer>**

**@code {**

**[Parameter] public CustomGroupModel Group { get; set; }**

**}**

* It is mandatory to wrap everything in a **GroupContainer** component so that everything works properly, such as the group's position, moving it, etc..
* It is mandatory to add both **GroupLinks** and **GroupNodes** as direct children of the container. This will render the appropriate layers and populate/configure them properly.
* As with custom nodes, you are in charge of rendering the ports.

For any custom content, you can add it wherever you want, as long as it's inside the container like our title **span**.

Para cualquier contenido personalizado, puede agregarlo donde quiera, siempre que esté dentro del contenedor como nuestro **span** (**rango de)** título.

Como con todo lo demás, el componente se puede diseñar utilizando **CSS**. Aquí está el estilo del grupo personalizado en las demos:

**.group.custom {**

**outline: 2px solid black;**

**background-color: #6fbb6e;**

**}**

**.group.custom > span.title {**

**padding: 20px;**

**position: absolute;**

**left: 50%;**

**transform: translate(-50%, -50%);**

**background: #eee;**

**border: 2px solid black;**

**border-radius: 50%;**

**background-color: #6fbb6e;**

**font-weight: bold;**

**text-transform: uppercase;**

**}**

### Registering the custom group

Por último, registraremos el enlace personalizado y su componente en el **Diagram**:

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**\_diagram.RegisterModelComponent<CustomGroupModel, CustomGroupWidget>();**

**var node1 = NewNode(50, 50);**

**var node2 = NewNode(300, 300);**

**var node3 = NewNode(500, 100);**

**\_diagram.Nodes.Add(new [] { node1, node2, node3 });**

**\_diagram.AddGroup(new CustomGroupModel(new[] { node2, node3 }, "Group 1"));**

**\_diagram.Links.Add(new LinkModel(node1.GetPort(PortAlignment.Right), node2.GetPort(PortAlignment.Left)));**

**\_diagram.Links.Add(new LinkModel(node2.GetPort(PortAlignment.Right), node3.GetPort(PortAlignment.Bottom)));**

**}**

**private NodeModel NewNode(double x, double y)**

**{**

**var node = new NodeModel(new Point(x, y));**

**node.AddPort(PortAlignment.Bottom);**

**node.AddPort(PortAlignment.Top);**

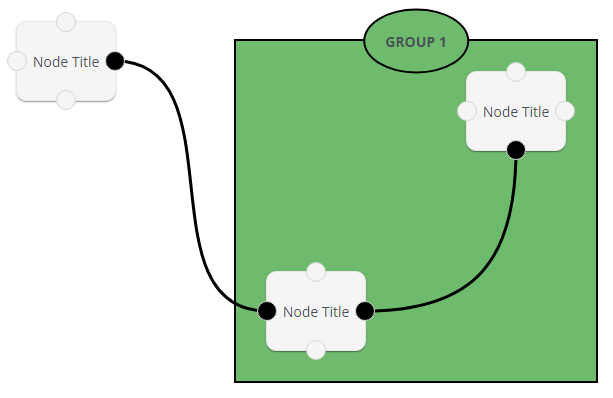
**node.AddPort(PortAlignment.Left);**

**node.AddPort(PortAlignment.Right);**

**return node;**

**}**

Here's how it would look like:



# Options

 Last updated: October 17th, 2020

## Diagram options

In **Z.Blazor.Diagrams**, you can customize a lot of things using the available diagram options.

### Table

Here are all the available options, as well as their default values:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Default | Description |
| **DeleteKey** | String | Delete | Key code for deleting entities |
| **DefaultNodeComponent** | Type |  | The default component for nodes |
| **GridSize** | Int32? |  | The grid size (grid-based snaping |
| **AllowMultiSelection** | Boolean | True | Whether to allow users to select multiple nodes at once using CTRL or not |
| **AllowPanning** | Boolean | True | Whether to allow panning or not |
| **EnableVirtualization** | Boolean | True | Only render visible nodes |
| **Zoom.Enabled** | Boolean | True | Whether to allow zooming or not |
| **Zoom.Inverse** | Boolean | False | Whether to inverse the zoom direction or not |
| **Zoom.Minimum** | Double | 0,1 | Minimum value allowed |
| **Zoom.Maximum** | Double | 2 | Maximum value allowed |
| **Zoom.ScaleFactor** | Double | 1,05 | Zoom Scale Factor. Should be between 1.01 and 2. Default is 1.05. |
| **Links.DefaultLinkComponent** | Type |  | The default component for links |
| **Links.DefaultColor** | String | black | The default color for links |
| **Links.DefaultSelectedColor** | String | rgb(110, 159, 212) | The default color for selected links |
| **Links.DefaultRouter** | Router | Blazor.Diagrams.Core.Router | Default Router for links |
| **Links.DefaultPathGenerator** | PathGenerator | Blazor.Diagrams.Core.PathGenerator | Default **PathGenerator** for links |
| **Links.EnableSnapping** | Boolean | False | Whether to enable link snapping |
| **Links.SnappingRadius** | Double | 50 | Link snapping radius |
| **Links.Factory** | LinkFactory | Blazor.Diagrams.Core.LinkFactory | Link model factory |
| **Groups.Enabled** | Boolean | False | Whether to allow users to group/ungroup nodes |
| **Groups.KeyboardShortcut** | Func`2 | System.Func`2[Microsoft.AspNetCore.Components.Web.KeyboardEventArgs,System.Boolean] | Keyboard shortcut (**CTRL+ALT+G** by default) |
| **Groups.Factory** | GroupFactory | Blazor.Diagrams.Core.GroupFactory | Group model factory |
| **Constraints.ShouldDeleteNode** | Func`2 | System.Func`2[Blazor.Diagrams.Core.Models.NodeModel,System.Boolean] | Decide if a node can/should be deleted |
| **Constraints.ShouldDeleteLink** | Func`2 | System.Func`2[Blazor.Diagrams.Core.Models.Base.BaseLinkModel,System.Boolean] | Decide if a link can/should be deleted |
| **Constraints.ShouldDeleteGroup** | Func`2 | System.Func`2[Blazor.Diagrams.Core.Models.GroupModel,System.Boolean] | Decide if a group can/should be deleted |

### Example

Here's an example usage:

**protected override void OnInitialized()**

**{**

**base.OnInitialized();**

**var options = new DiagramOptions**

**{**

**DeleteKey = "Delete",**

**InverseZoom = false,**

**DefaultNodeComponent = null,**

**GridSize = 40,**

**GroupingEnabled = true,**

**AllowMultiSelection = true,**

**AllowPanning = true,**

**AllowZooming = true,**

**Links = new DiagramLinkOptions**

**{**

**DefaultLinkType = LinkType.LineWithArrowToTarget,**

**DefaultLinkComponent = null,**

**DefaultColor = "blue",**

**DefaultSelectedColor = "red"**

**}**

**};**

**Diagram = new Diagram(options);**

**}**

[**Diagram Options**](https://blazor-diagrams.zhaytam.com/#diagram-options)[Table](https://blazor-diagrams.zhaytam.com/#diagram-options-table)[**Example**](https://blazor-diagrams.zhaytam.com/#diagram-options-example)