

[](https://www.tutorialr.com/tutorials/)

Windows App SDK

Dial Control





# Dial Control

**Dial Control** shows how to create a **Control** that can be used as a **Dial** using **Windows App SDK**

## Step 1

Follow **Setup and Start** on how to get **Setup** and **Install** what you need for **Visual Studio 2022** and **Windows App SDK**.

|  |  |
| --- | --- |
| In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it. | Text  Description automatically generated |
| Once **Visual Studio 2022** has started select **Create a new project**. | **Graphical user interface, text  Description automatically generated** |
| Then choose the **Blank App, Packages (WinUI in Desktop)** and then select **Next**. | **Graphical user interface, text  Description automatically generated** |
| After that in **Configure your new project** type in the **Project name** as *DialControl*, then select a Location and then select **Create** to start a new **Solution**. | **Graphical user interface, text, application, email  Description automatically generated** |

## Step 2

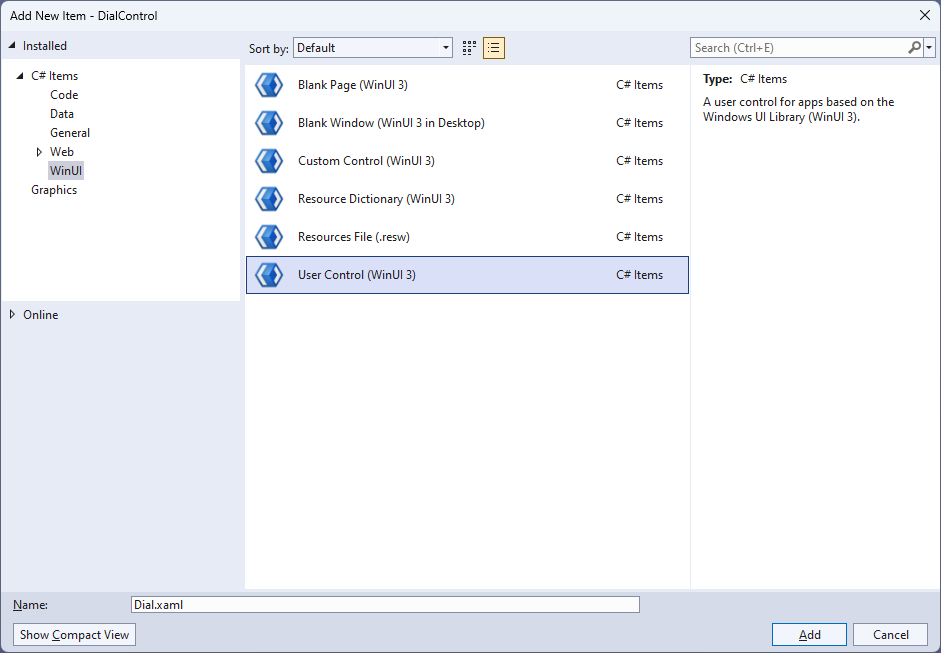
Then in **Visual Studio** within **Solution** **Explorer** for the **Solution**, right click on the **Project** shown below the **Solution** and then select **Add** then **New Item…**

Table

Description automatically generated with low confidence

## Step 3

Then in **Add New Item** from the **C# Items** list, select **WinUI** and then select **User Control (WinUI 3)** from the list next to this, then type in the name of *Dial.xaml* and then **Click** on **Add**.



## Step 4

|  |  |
| --- | --- |
| Then from **Solution** **Explorer** for the **Solution** double-click on **Dial.xaml** to see the **XAML** for the **User Control**. |  |

## Step 5

In the **XAML** for *Dial.xaml* there be some **XAML** for a **Grid**, above **</Grid>**, type in the following **XAML**:

<Grid Name="DialGrid" Loaded="Load">

<ContentPresenter Content="{x:Bind Face}"/>

<ContentPresenter Content="{x:Bind Knob}"

RenderTransformOrigin="0.5,0.5">

<ContentPresenter.RenderTransform>

<TransformGroup>

<RotateTransform x:Name="DialValue" />

</TransformGroup>

</ContentPresenter.RenderTransform>

</ContentPresenter>

</Grid>

This **XAML** contains a **Grid** with a **Loaded** event handler of **Load** along with a **ContentPresenter** for the **Face** and **Knob** of the **Dial** which also has a **RotateTransform** to show the correct indicator for the **Dial**.

## Step 6

|  |  |
| --- | --- |
| Then, within **Solution** **Explorer** for the **Solution** select the arrow next to **Dial.xaml** then double-click on **Dial.xaml.cs** to see the **Code** for the **User Control**. |  |

## Step 7

You will now be in the **View** for the **Code** of *Dial.xaml.cs*, type in the following **Code** below the end of the **Constructor** of **public Dial() { ... }**:

private bool \_hasCapture = false;

// Dependancy Properties

// Properties

// GetRotation, GetAngle & SetPosition Methods

// Load Method

The **class** for **Dial** represents the **User Control** for the **Dial** and includes a **bool** that will be used to know when the **Dial** is being interacted with.

## Step 8

While still in the **class** of **Dial** after the **Comment** of **// Dependency Properties** type the following **Dependency Properties**:

public static readonly DependencyProperty ValueProperty =

DependencyProperty.Register(nameof(Value), typeof(double),

typeof(Dial), null);

public static readonly DependencyProperty MinimumProperty =

DependencyProperty.Register(nameof(Minimum), typeof(double),

typeof(Dial), null);

public static readonly DependencyProperty MaximumProperty =

DependencyProperty.Register(nameof(Maximum), typeof(double),

typeof(Dial), null);

public static readonly DependencyProperty KnobProperty =

DependencyProperty.Register(nameof(Knob), typeof(UIElement),

typeof(Dial), null);

public static readonly DependencyProperty FaceProperty =

DependencyProperty.Register(nameof(Face), typeof(UIElement),

typeof(Dial), null);

There will also be some **Errors** as these refer to **Properties** that will be added in the next step.

These **Dependency Properties** refer to various **Properties** of the **Dial** that can be customised for the **User Control**.

## Step 9

While still in the **class** of **Dial** after the **Comment** of **// Properties** type the following **Properties**:

public double Value

{

get { return (double)GetValue(ValueProperty); }

set { SetValue(ValueProperty, value); }

}

public double Minimum

{

get { return (double)GetValue(MinimumProperty); }

set { SetValue(MinimumProperty, value); }

}

public double Maximum

{

get { return (double)GetValue(MaximumProperty); }

set { SetValue(MaximumProperty, value); }

}

public UIElement Knob

{

get { return (UIElement)GetValue(KnobProperty); }

set { SetValue(KnobProperty, value); }

}

public UIElement Face

{

get { return (UIElement)GetValue(FaceProperty); }

set { SetValue(FaceProperty, value); }

}

Any **Errors** should now be resolved, if you continue to get them check any previous steps to see if you have missed anything.

These **Properties** are for values for the **User Control** such as the **Minimum** or **Maximum** values for the **Dial**.

## Step 10

While still in the **class** of **Dial** after the **Comment** of **// GetRotation, GetAngle & SetPosition Methods** type the following **Methods**:

private double GetRotation(double width, double height, Point point)

{

double radius = width / 2;

Point centre = new(radius, height / 2);

double triangleTop = Math.Sqrt(Math.Pow(point.X - centre.X, 2)

+ Math.Pow(centre.Y - point.Y, 2));

double triangleHeight = (point.Y > centre.Y) ?

point.Y - centre.Y : centre.Y - point.Y;

return triangleHeight \* Math.Sin(90) / triangleTop \* 100;

}

private double GetAngle(Point point)

{

double diameter = DialGrid.ActualWidth;

double height = DialGrid.ActualHeight;

double radius = diameter / 2;

double rotation = GetRotation(diameter, height, point);

if ((point.X > radius) && (point.Y <= radius))

{

rotation = 90.0 + (90.0 - rotation);

}

else if ((point.X > radius) && (point.Y > radius))

{

rotation = 180.0 + rotation;

}

else if ((point.X < radius) && (point.Y > radius))

{

rotation = 270.0 + (90.0 - rotation);

}

return rotation;

}

private void SetPosition(double rotation)

{

if (Minimum >= 0 && Maximum > 0 && Minimum < 360 && Maximum <= 360)

{

if (rotation < Minimum) { rotation = Minimum; }

if (rotation > Maximum) { rotation = Maximum; }

}

DialValue.Angle = rotation;

Value = rotation;

}

These **Methods** will be used to determine the rotation for the **Dial** with **GetRotation** which will be used to set the angle with **GetAngle** and **SetPosition** will be used to constrain the angle as needed for the **Dial**.

## Step 11

While still in the **class** of **Dial** after the **Comment** of **// Load Method** type the following **Method**:

private void Load(object sender, RoutedEventArgs e)

{

if (Minimum > 0 && Minimum < 360)

SetPosition(Minimum);

DialGrid.PointerReleased += (object sender, PointerRoutedEventArgs e) =>

\_hasCapture = false;

DialGrid.PointerPressed += (object sender, PointerRoutedEventArgs e) =>

{

\_hasCapture = true;

SetPosition(GetAngle(e.GetCurrentPoint(DialGrid).Position));

};

DialGrid.PointerMoved += (object sender, PointerRoutedEventArgs e) =>

{

if (\_hasCapture)

SetPosition(GetAngle(e.GetCurrentPoint(DialGrid).Position));

};

DialGrid.PointerExited += (object sender, PointerRoutedEventArgs e) =>

\_hasCapture = false;

}

**Load** will be used to set up the **Event** **Handlers** for the **Dial** for when the mouse is released or pressed and when the mouse is moved or leaves the **Dial**.

## Step 12

|  |  |
| --- | --- |
| Within **Solution** **Explorer** for the **Solution** double-click on **MainWindow.xaml** to see the **XAML** for the **Main Window**. |  |

## Step 13

In the **XAML** for **MainWindow.xaml** there be some **XAML** for a **StackPanel**, this should be **Removed** by removing the following:

<StackPanel Orientation="Horizontal"

HorizontalAlignment="Center" VerticalAlignment="Center">

<Button x:Name="myButton" Click="myButton\_Click">Click Me</Button>

</StackPanel>

## Step 14

While still in the **XAML** for **MainWindow.xaml** above **</Window>**, type in the following **XAML**:

<local:Dial x:Name="Dial" Height="300" Width="300" Minimum="90.0" Maximum="275.0">

<local:Dial.Face>

<Ellipse Fill="{ThemeResource SystemControlHighlightAccentBrush}"/>

</local:Dial.Face>

<local:Dial.Knob>

<Rectangle Height="40" Width="150" Margin="5,0,145,0"

RadiusX="20" RadiusY="20"

Fill="{ThemeResource SystemControlBackgroundAltHighBrush}"/>

</local:Dial.Knob>

</local:Dial>

This **XAML** contains the **User Control** of **Dial** with the **Face** and **Knob** set along with other **Properties** such as the **Minimum** and **Maximum** values.

## Step 15

|  |  |
| --- | --- |
| Then, within **Solution** **Explorer** for the **Solution** select the arrow next to **MainWindow.xaml** then double-click on **MainWindow.xaml.cs** to see the **Code** for the **Main Window**. |  |

## Step 16

In the **Code** for **MainWindow.xaml.cs** there be a **Method** of **myButton\_Click(...)** this should be **Removed** by removing the following:

private void myButton\_Click(object sender, RoutedEventArgs e)

{

myButton.Content = "Clicked";

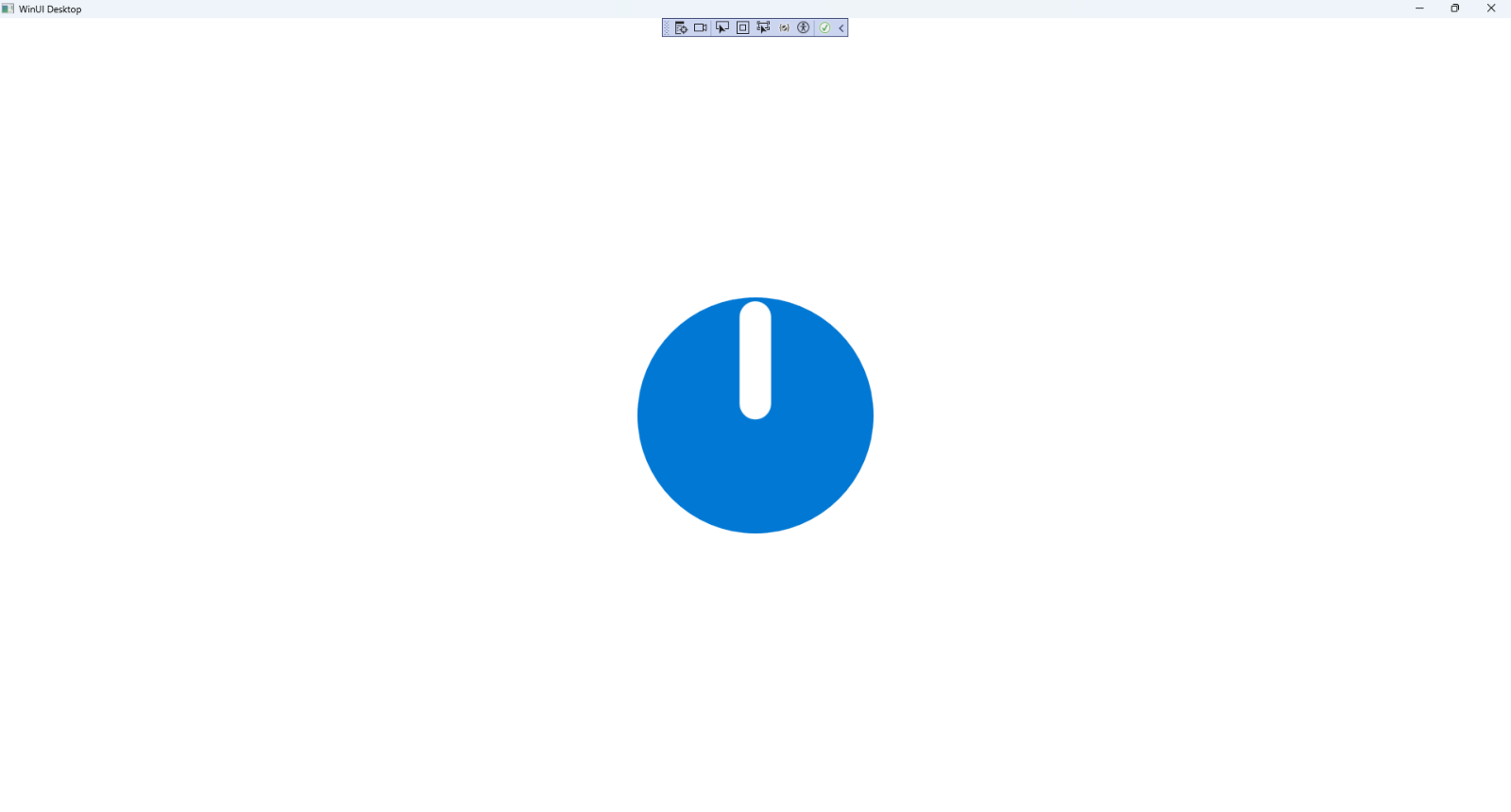
}

## Step 17

|  |  |
| --- | --- |
| That completes the **Windows App SDK** application. In **Visual Studio 2022** from the **Toolbar** select **DialControl (Package)** to **Start** the application. |  |

## Step 18

Once running you will see the **Dial Control** displayed, then you can rotate it to set the **Value** for the **Dial**.

****

## Step 19

|  |  |
| --- | --- |
| To **Exit** the **Windows App SDK** application, select the **Close** button from the top right of the application as that concludes this **Tutorial** for **Windows App SDK** from [tutorialr.com](https://tutorialr.com)! |  |