Tutorial – Custom Controls

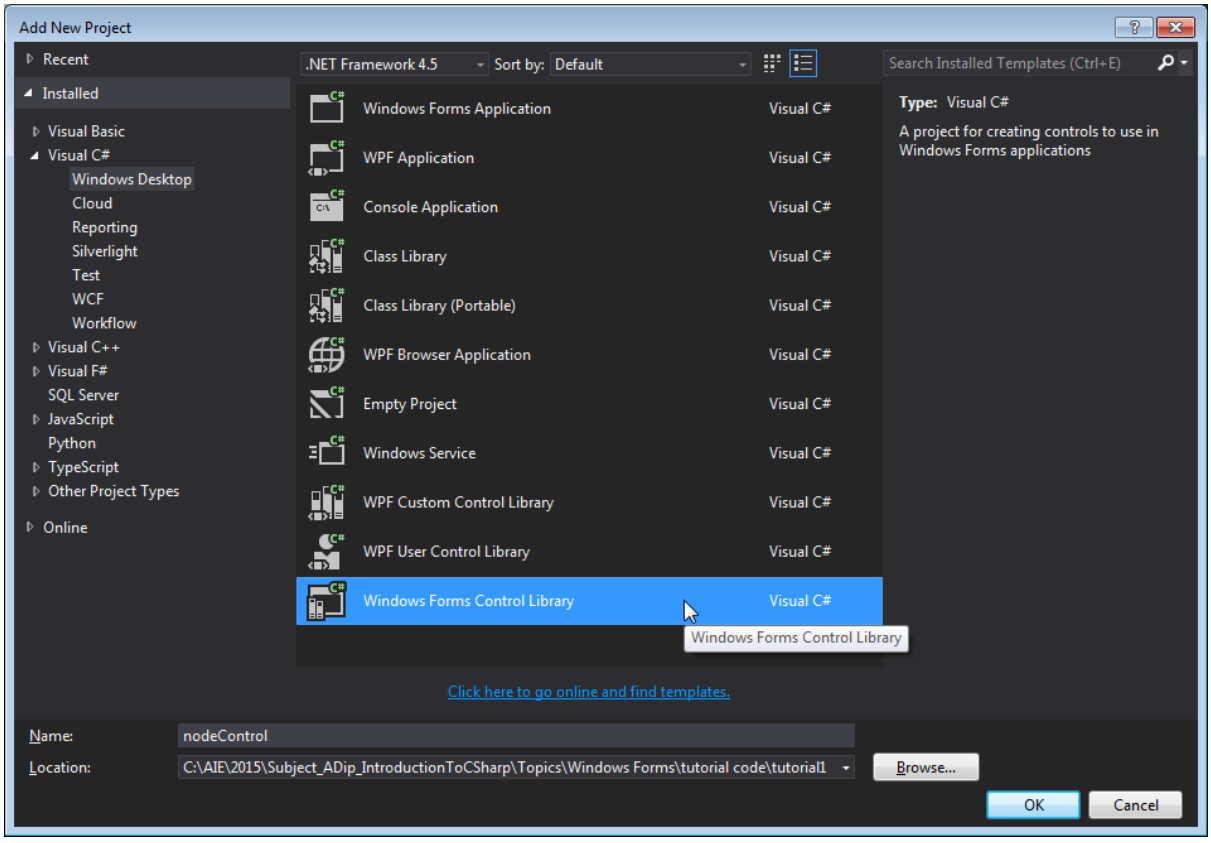
In this tutorial we are going to replace the *PictureBox* with a custom control. We'll use this control to draw a circle (node) corresponding to where the user clicked.

Creating a Control Library:

We will create our custom control inside its own library (dll).

1. From the *Solution* context menu select *Add*, *New Project*

   Fill in the details for your new control.



2. Don't worry about the Forms Designer, go straight to the source code.

   The first thing we must do is write a new OnPaint function, because all custom controls must implement their own paint method.

   Add the following code to the class:

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);

    e.Graphics.DrawArc(Pens.Black,
       new Rectangle(currentCoordinates, new Size(5, 5)), 0, 360);
}
```

This will draw a circle at the coordinates defined by the variable *currentCoordinates*.

3. Add the variable *currentCoordinates* to the class definition.

```
private Point currentCoordinates = new Point(0, 0);
```

4. Finally, we need some way to update the *currentCoordinates* value. We want to do that every time the user clicks on the control, so we need to override the *OnClick* function.

   Insert the following function into the class:

```
protected override void OnClick(EventArgs e)
{
    if(e.GetType() == typeof(MouseEventArgs))
    {
        MouseEventArgs me = e as MouseEventArgs;
        currentCoordinates = me.Location;
        Refresh();
    }

    base.OnClick(e);
}
```

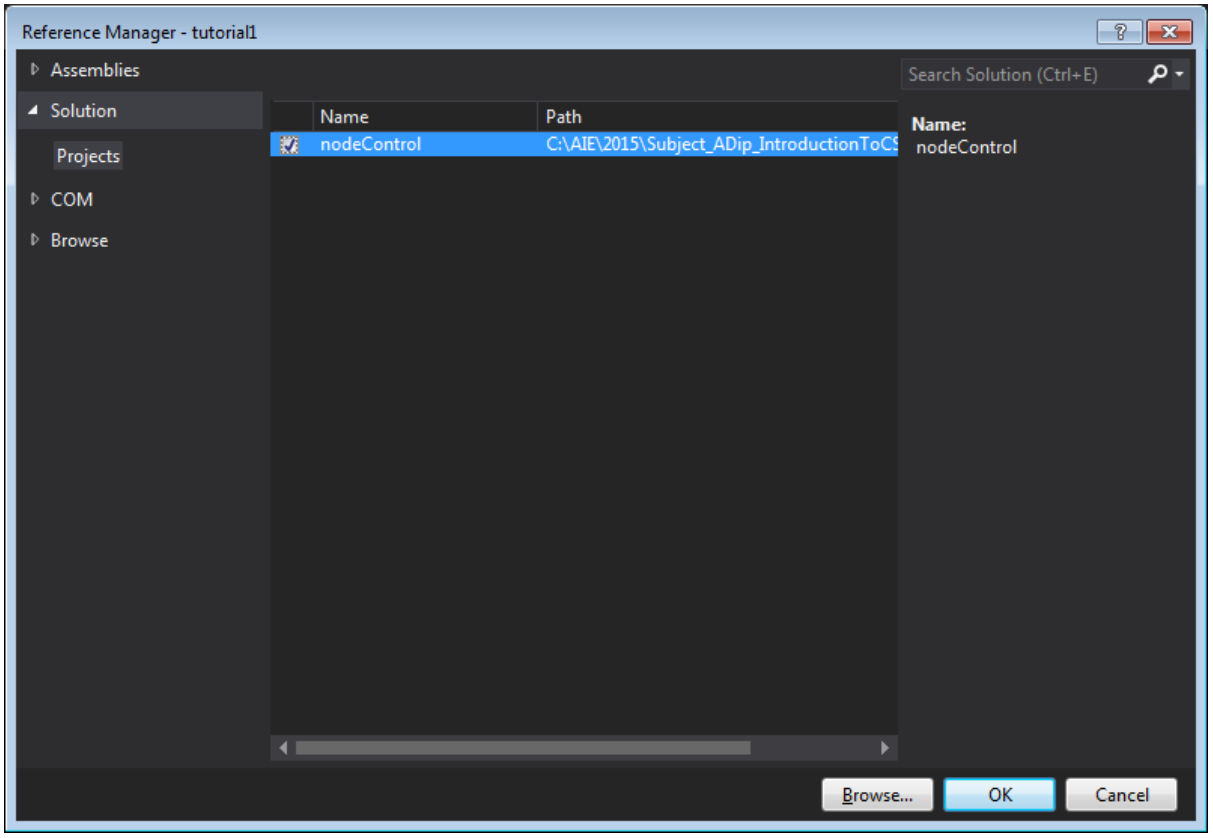
That's it. That's our basic control build. Later on you can add your point object list to this control and expose member functions to add and remove points to that list. But for now, this basic control will suffice.

Adding the Control to the Form:

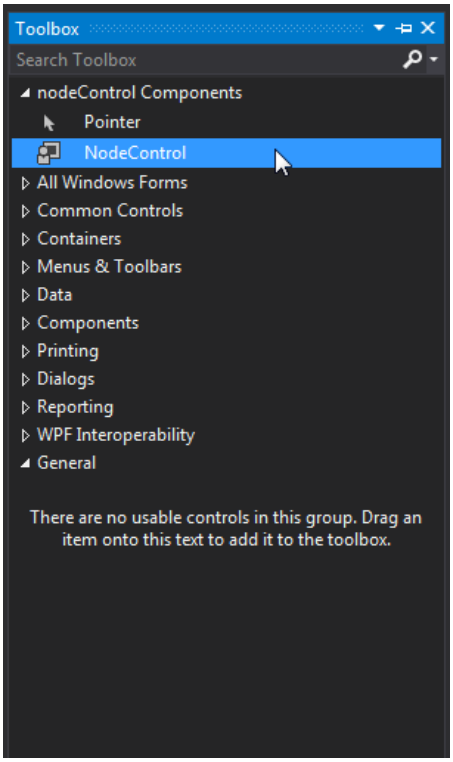
We need to add our new control to our existing form. This is actually relatively painless.

1. Add a reference to the project containing the custom control.

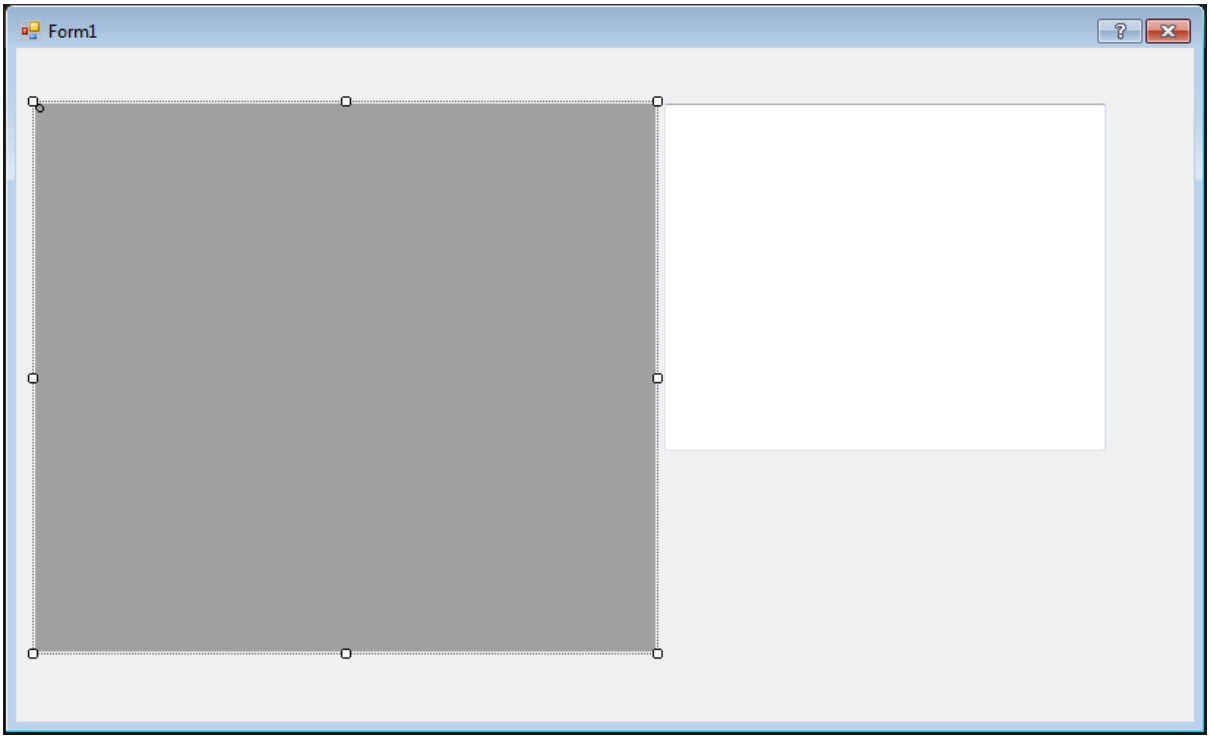
   Ensure the MDI Application is selected, and then from the *Project* menu select *Add Reference*
2. In the *Reference Manager* dialog, ensure that the custom control's project is checked and press *OK*



3. Rebuild your solution. You won't be able to see your control in the *Toolbox* until you rebuild.
4. Select your control from the *Toolbox* and replace the *PictureBox* control with your custom control.



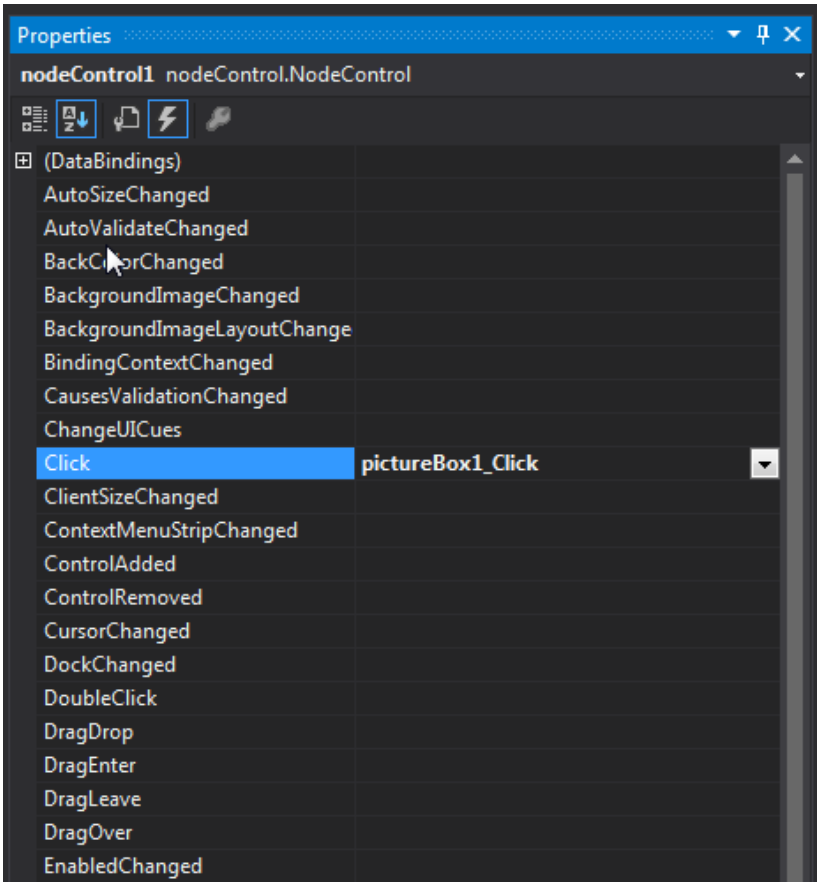
If you want to change the name of your control, go back to the control's project, open the *Properties* window in the Forms Designer and change the *Name* property. Don't forget to rebuild the solution.



Set the *BackColor* of your control to something nice.

5. Now we need to add the *Click* event handler. We've already written this function for the *PictureBox*, so we can reuse that function.

   In the *Properties* window, select the *Events* and scroll down to the *Click* action. In the drop-down list to the right, select the event handler that we wrote for the *PictureBox*



Run your program. If everything went as expected a circle should be drawn at the last position you clicked and the coordinates should appear in the *TextBox* (actually, everything else should function as before).

If you added help bindings to the old *PictureBox* control you'll need to reset those, along with any other properties or events you may have modified.

Congratulations, you've now got the foundations for a pretty decent node path editing program.

Exercises:

1. Move your point list into the custom control. Expose member functions for adding and removing points to and from the list. Update your MDI application so that all the points in the custom control's list are written to the TextBox any time the list changes.

2. Extend your control so that lines are drawn between nodes. Use your creativity when defining how the user will specify which nodes should be connected by lines.