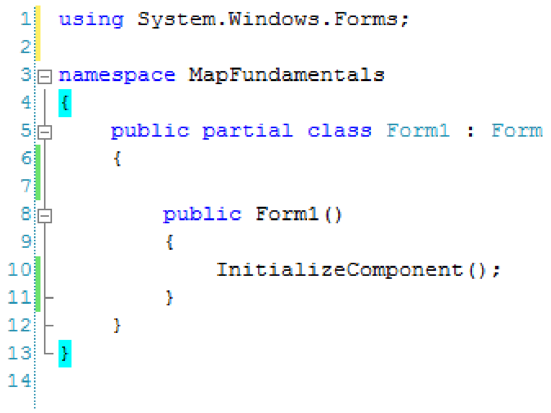
Draw On the Map

Ok, so this tutorial assumes that you successfully got your first project started from the help in “Add a Map control to a Form”. Having done this, it might be nice to be able to show a watermark or image on top of the map. Perhaps this is a fast moving object like a GPS point that needs to be updated frequently. In this section we look at some different alternatives for drawing content on the map. This section, however, does not attempt to draw georeferenced content, which can be illustrated later.

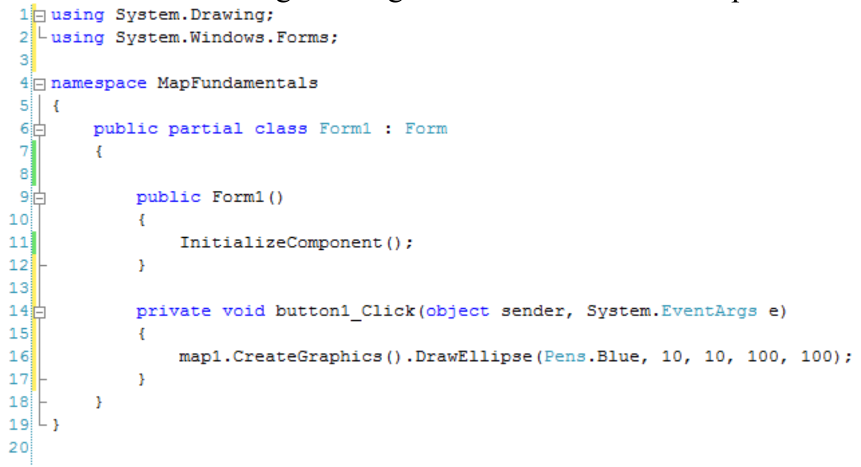
If we look at the code behind the form, we see that Microsoft has hidden most of the form creation code away in the designer, leaving us with a basically empty class for Form1. Figure 1 shows the code that was in the Form1 class before we add anything.



```
using System.Windows.Forms;

namespace MapFundamentals
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

Figure 1: Initial Form Code

We have carefully added a button called button1. Double click on that button in the designer view, and you should automatically create an event handler for when that button is clicked. In that event handler, we will forceably draw a circle on the form by using the CreateGraphics method that is common to all windows controls. Notice the additional reference that we need to System.Drawing in order to make use of the Pens enumeration. Figure 2 shows the code including our single method to draw the ellipse.



```
using System.Drawing;
using System.Windows.Forms;

namespace MapFundamentals
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            map1.CreateGraphics().DrawEllipse(Pens.Blue, 10, 10, 100, 100);
        }
    }
}

```

Figure 2: Draw Circle using CreateGraphics

Running this code at first seems like it does what we want. It shows a white map, and when we click the button, we can see a blue circle.

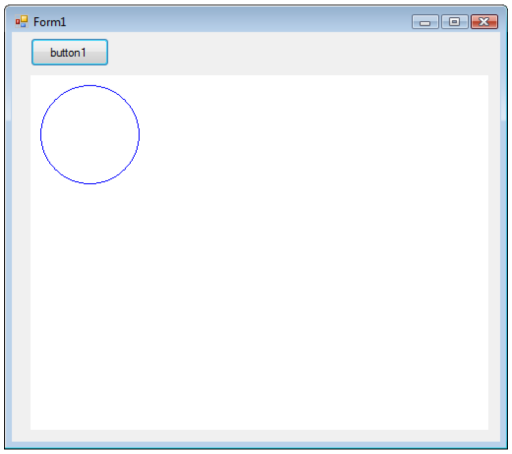


Figure 3: Blank map with blue circle

However, the instant we move the form, you should notice that the blue circle vanishes again. If we drag a shapefile onto the map as demonstrated in "Add a Map control to a Form" we can see another interesting fact. Figure 4 shows that pressing the button will draw a circle above any data that is currently drawn to the map. However, as mentioned before, the instant that you use the pan or zoom features, or resize or even move the form, the blue circle will disappear again.

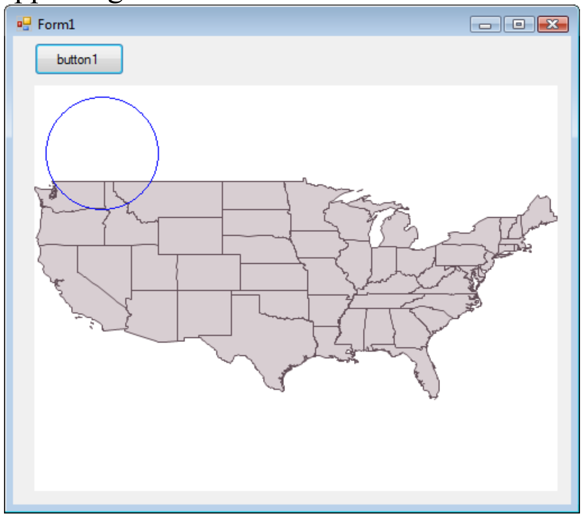


Figure 4: State map with Circle

To make the circle become more permanent, once we press the button, we need to basically re-draw the circle every time the screen updates. To do this, we are going to need to be drawing the circle in multiple places. Therefore, I pulled the draw circle method out into a single method. The reason for this is if we update the drawing code in the future, we would rather just update one function. Since we don't want to draw the code until the button is pressed, I am using a class level Boolean value called _showCircle. I used an underscore before the variable name to indicate that the variable is a class level variable. It is possible to access class level variables from within any method in the class. Using an underscore consistently also groups together your class level variables in intellisense so that you can remember what you named them, even if the variable declaration is well out of view.

```
using System.Drawing;
using System.Windows.Forms;

namespace MapFundamentals
{
    public partial class Form1 : Form
    {
        private bool _showCircle;

        public Form1()
        {
            InitializeComponent();
            map1.Paint += map1_Paint;
        }
        private void button1_Click(object sender, System.EventArgs e)
        {
            _showCircle = true;
            Graphics g = map1.CreateGraphics();
            DrawCircle(g);
            g.Dispose();
        }
        void map1_Paint(object sender, PaintEventArgs e)
        {
            DrawCircle(e.Graphics);
        }
        private void DrawCircle(Graphics g)
        {
            if(!_showCircle) return;
            g.DrawEllipse(Pens.Blue, 10, 10, 100, 100);
        }
    }
}
```

Figure 5: Permanent Circle Code

I also updated the code in button1_click so that we can not only create the graphics object, but that we clean up after ourselves and dispose the graphics objects that we create. The rule is that if the graphics object is handed to you through a parameter, you don't need to dispose it because the creator will dispose of the variable after the method is finished. If, on the other hand, you create the graphics object, then you should dispose the graphics object. The code in Figure 5 will re-draw the circle even as you add data to the map or redraw content, without having to continually press the button. The graphics surface is a very powerful tool and can be used to draw images, lines, ellipses, polygons as well as filling those shapes. It is, in fact, these very methods that MapWindow 6 uses to do all of its rendering.