Path & Geometries

2D Graphics in WPF

Agenda

- Path and Geometries

2D Graphics – the different layers

- WPF has three layers of 2D graphics:
  - **Shapes**
    - Are UIElements that represent drawings like:

```
<Ellipse Fill="Cyan"  Width="50" Height="20" />
```

  - **Drawings** (aka Visuals)
    - Are a an object-oriented view of drawing instructions like:

```
GeometryDrawing d = new GeometryDrawing(Brushes.Red,
null,
new EllipseGeometry(new rect(7, 7, 15, 15)));
```

  - **Drawing Instructions** (aka Direct rendering)
    - Are low-level commands to the drawing-engine like:

```
drawingContext.DrawRectangle(Brushes.Red, null, new
Rect(0, 0, 100, 50));
```

PATH AND GEOMETRIES

A Path Contains Geometry Objects

- Path is a Shape-derived class that has the ability to contain:
  - any simple shape
  - curves
  - groups of shapes
- The Path class has a property, named Data, that accepts a **Geometry** object that defines the shape (or shapes) the path includes
- The Path object has also the Stroke and Fill brushes used to paint the Path
- The Path class also includes the features it inherits from the UIElement infrastructure, such as mouse and keyboard handling

Geometry Classes

- Geometry is an abstract base class
- Geometry desendents:
  - LineGeometry
  - RectangleGeometry
  - EllipseGeometry
  - GeometryGroup
  - CombinedGeometry
  - PathGeometry
  - StreamGeometry
- A Geometry object defines details such as the coordinates and size of a shape – not brushes and pens
- The geometry classes can also be used to define drawings that you can apply through a brush, which gives you an easy way to paint complex content that doesn't need the user-interactivity features of the Path class (Drawings)

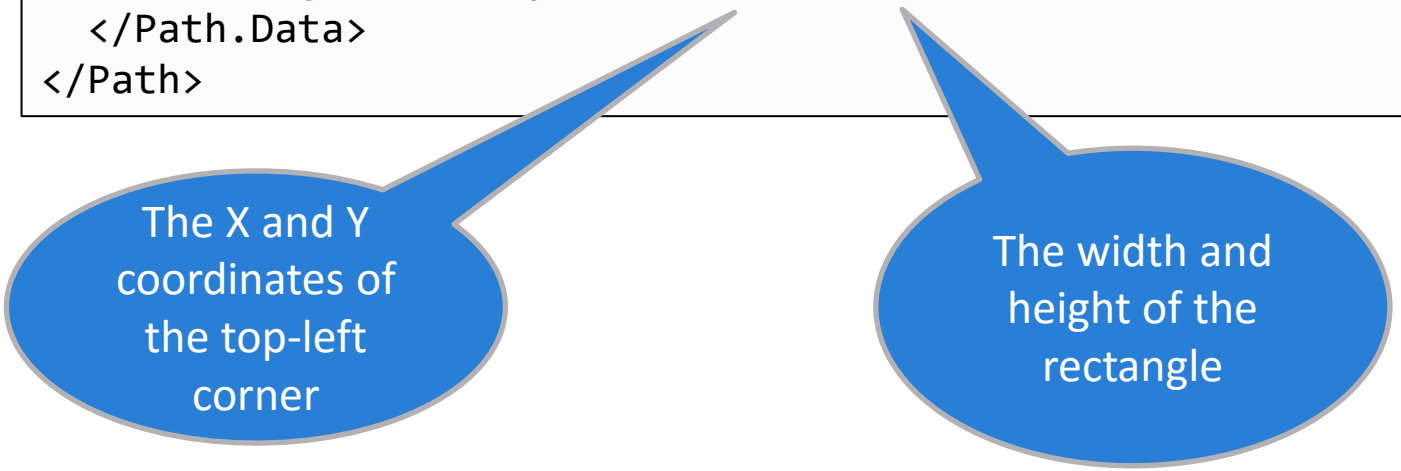
Rectangle Geometry

- The RectangleGeometry maps directly to the Rectangle shape.

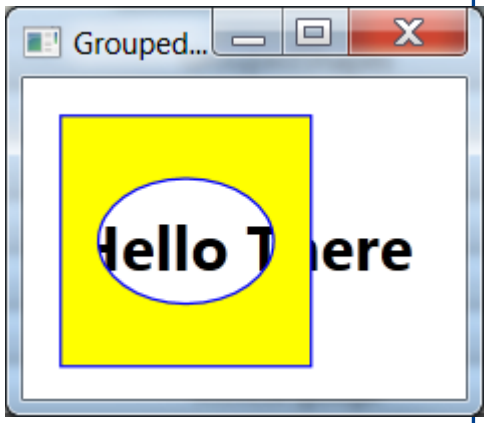
```
<Rectangle Fill="Yellow" Stroke="Blue"
           Width="100" Height="50“
           />
```

Maps to this markup that uses the Path element:

```
<Path Fill="Yellow" Stroke="Blue">
  <Path.Data>
    <RectangleGeometry Rect="0,0 100,50" />
  </Path.Data>
</Path>
```



GeometryGroup



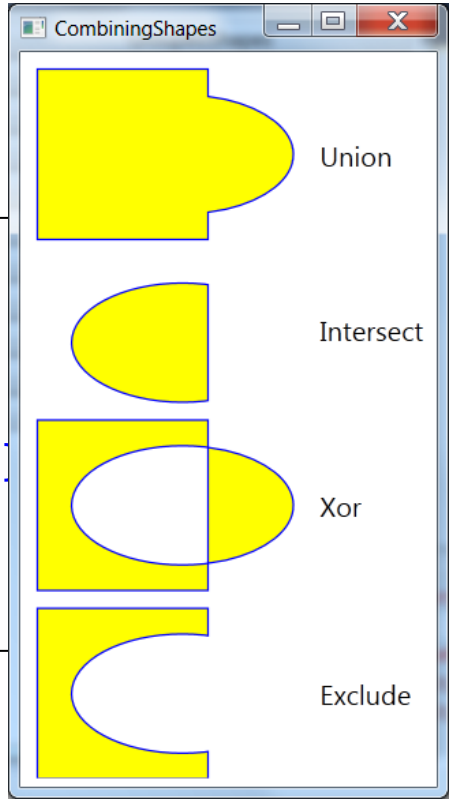
- The simplest way to combine geometries is to use the GeometryGroup
  - The effect of this markup is the same as if you had supplied two Path elements
- The advantage:
  - A window that uses a smaller number of elements with more complex geometries will perform faster than a window that has a large number of elements with simpler geometries

```
<TextBlock Canvas.Top="50" Canvas.Left="20" Text="Hello There" />
<Path Fill="Yellow" Stroke="Blue" Margin="5"
      Canvas.Top="10" Canvas.Left="10" >
  <Path.Data>
    <GeometryGroup FillRule="EvenOdd">
      <RectangleGeometry Rect="0 0 100 100" />
      <EllipseGeometry Center="50 50" RadiusX="35" RadiusY="25" />
    </GeometryGroup>
  </Path.Data>
</Path>
```

CombinedGeometry

- Is used to combine shapes that overlap, and where neither shape contains the other completely
- Takes just two geometries, which you supply using the Geometry1 and Geometry2 properties
  - But you can use nested CombinedGeometry objects

```
<Path Fill="Yellow" Stroke="Blue" Margin="5">
  <Path.Data>
    <CombinedGeometry GeometryCombineMode="Union"
       CombinedGeometry.Geometry1="{StaticResource rect}"
       CombinedGeometry.Geometry2="{StaticResource ellipse}
    </CombinedGeometry>
  </Path.Data>
</Path>
```



The PathGeometry

- Can draw anything that the other geometries can, and more
  - The only drawback is a lengthier and more complex syntax
- Is built out of one or more PathFigure objects
  - which are stored in thePathGeometry.Figures collection
- Each PathFigure is a continuous set of connected lines and curves that can be closed or open
- The PathFigure class has four key properties:

StartPoint	This is a point that indicates where the line for the figure begins.
Segments	This is a collection of PathSegment objects that are used to draw the figure.
IsClosed	If true, WPF adds a straight line to connect the starting and ending points (if they aren't the same).
IsFilled	If true, the area inside the figure is filled in using the Path.Fill brush.

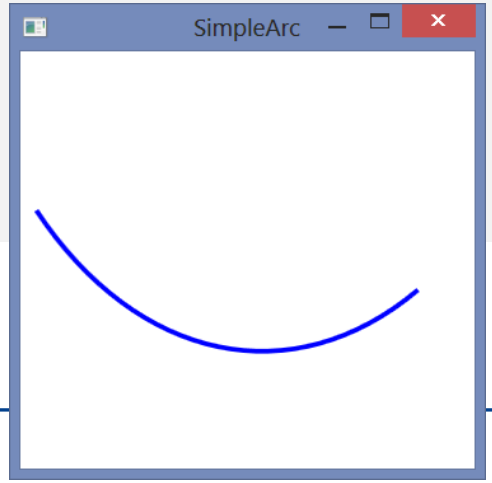
PathSegment Classes

Name	Description
LineSegment	Creates a straight line between two points.
ArcSegment	Creates an elliptical arc between two points.
BezierSegment	Creates a Bézier curve between two points.
QuadraticBezierSegment	Creates a simpler form of Bézier curve that has one control point instead of two, and is faster to calculate.
PolyLineSegment	Creates a series of straight lines.
PolyBezierSegment	Creates a series of Bézier curves.
PolyQuadraticBezierSegment	Creates a series of simpler quadratic Bézier curves.

Arcs

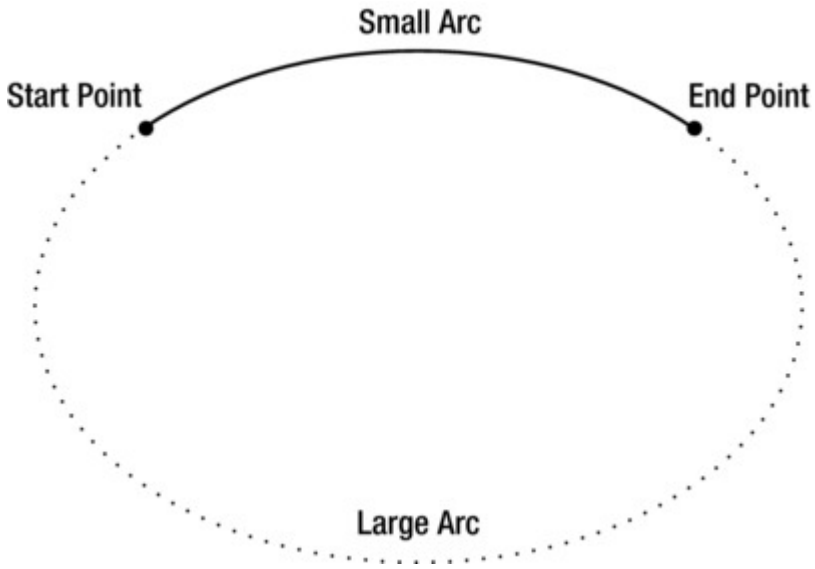
- You identify the end point of the line by using the ArcSegment.Point property
- And you indicate the size of the imaginary ellipse that's being used to draw the arc by using the ArcSegment.Size property
  - which supplies the X radius and the Y radius of the ellipse

```
<Path Stroke="Blue" StrokeThickness="3">
  <Path.Data>
    <PathGeometry>
      <PathFigure IsClosed="False" StartPoint="10,100" >
        <ArcSegment Point="250,150" Size="200,300" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

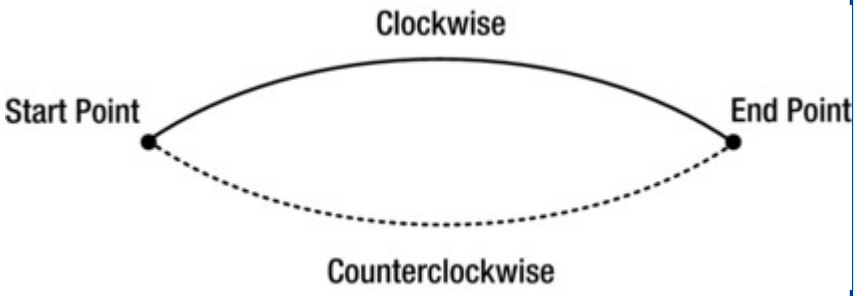


Arcs – the other way

- Two ways to trace a curve along an ellipse:
  - The `ArcSegment.IsLargeArc` property can be true or false



- The curve could be stretched down and then up, or it could be flipped so that it curves up and then down:
  - the `ArcSegment.SweepDirection` property, which can be Counterclockwise (the default) or Clockwise



Bézier Curves

- Connect two line segments by using a complex mathematical formula that incorporates two *control points* that determine how the curve is shaped

```
<PathFigure StartPoint="10,10">
    <BezierSegment Point1="130,30"
                   Point2="40,140"
                   Point3="150,150">
    </BezierSegment>
</PathFigure>
```