Layer Inheritance and Casting

The most basic idea of a layer is that it combines the data being displayed with the symbology (or scheme) that should be used to represent that dataset. MapWindow 4.x just placed all the properties and methods directly on a single class, but this can create confusion from the fact that most of the methods available on the class would not actually do anything with the specific layer in hand. The model we are using in MapWindow 6.0 uses inheritance to allow the layer classes to share some of the content while the methods, properties or events that only work in some cases are placed in sub-classes. The diagram below illustrates the hierarchy for layers and also shows some example code for casting a layer from the map to the appropriate subclass.

Cannot resolve image macro, invalid image name or id.

```
private void SomeBasicsForAllLayers()
{
    // A layer combines data, symbology, and drawing functions.
    // Some things can be controlled directly with the layer:
    ILayer layer = map1.Layers[0];

    // Feature Layers only share some general symbology, but give
    // full access to the vector featrueset on the dataset.
    IFeatureLayer fl = layer as IFeatureLayer;

    // Safe casting may produce a null value
    if(fl != null)
    {
        // The dataset at this stage is automatically an IFeatureSet.
        IFeatureSet fs = fl.DataSet;

        // Further Subtypes include Line, Point, and Polygon layers
        IPointLayer pointl = fl as IPointLayer;
        if(pointl != null)
        {
            // point schemes control multiple categories of description with
filter expressions for each category
            IPointScheme pscheme = pointl.Symbology;
            //    A    point    symbolizers    are    a    shortcut    to
Symbology.Categories[0].Symbolizer
            IPointSymbolizer psymb = pointl.Symbolizer;
        }
        // The same works for Lines
        ILineLayer ll = fl as ILineLayer;
        if(ll != null)
        {
            ILineScheme lscheme = ll.Symbology;
            ILineSymbolizer ls = ll.Symbolizer;
        }
        // And Polygons
        IPolygonLayer pl = fl as IPolygonLayer;
        if(pl != null)
        {
            IPolygonScheme pscheme = pl.Symbology;
            IPolygonSymbolizer ps = pl.Symbolizer;
        }
        //  You  will  not  find  label  layers  directly  in  the  map  layers
collection.
        // Instead they belong directly to a feature layer.
        fl.LabelLayer.Symbology.Categories[0].Expression      =      "["      +
fs.DataTable.Columns[0].ColumnName + "]";
    }
    //  Raster  Layers  have  an  IRaster  for  the  DataSet,  instead  of  an
IFeatureSet
    IRasterLayer rl = layer as IRasterLayer;
    if(rl != null)
    {
        // The IRaster interface allows easy access to the data values.
        IRaster r = rl.DataSet;
        //  There  is  only  one  raster  symbolizer  and  it  controls  the  basic
properties
        IRasterSymbolizer symbolizer = rl.Symbolizer;
        // The color scheme lists the color settings
        IColorScheme scheme = symbolizer.Scheme;
    }

    // Image layers work
    IImageLayer il = layer as IImageLayer;
    if(il != null)
    {
        //  Image  data  controls  the  data  access  for  the  colors  at  a  pixel
location,
        //  or  else  provides  direct  access  to  a  byte  array  representing  the
image.
        IImageData id = il.DataSet;

        // A bitmap retrieval method automatically scales the representation
        //  for  the  specified  geographic  extent  so  that  it  works  with  the
pixel
        // size of the specified client rectangle.
        Bitmap bmp = id.GetBitmap(map1.Extents, map1.ClientRectangle);
        // At the moment the symbolizer here is just a placeholder for potential future uses.
    }
}
```

Create Random Points in C#

Points represent exact locations, though in many cases they are simply reference locations where the actual shape or area of the feature being represented is much less important than the approximate location of that item. These are also usually features that are numerous. An example would be cities of the united states, where each city is very small compared to the entire country. Since points don't have a size characteristic, generally, no matter how much you zoom in, points are always represented by a symbol that is the same size. In this example the geographic points each are also associated with an independent elevation. This elevation is considered ancillary to the geographic data, and so is stored in an Attribute. If many points should be considered a single entity, a MultiPoint shapefile can be created. Unlike MultiLineStrings and MultiPolygons, however, MultiPoints require a different feature type that Points because of the shapefile specification.

Cannot resolve image macro, invalid image name or id.

```
using MapWindow.Data;
using MapWindow.Geometries;
using MapWindow.Projections;

private void BuildPoints()
{
    // Setup a new "shapefile" by using the featureset object
    FeatureSet fs = new FeatureSet(FeatureTypes.Point);
    // You can control things like the projection with this object
    fs.Projection                                                        =
KnownCoordinateSystems.Projected.UtmNad1983.NAD1983UTMZone11N;
    // The DataTable is a standard .Net DataTable, so you can add columns the
normal way and use with DataGrid controls
    fs.DataTable.Columns.Add("Elevation", typeof(int));

    // Set up the specs for creating random points.  If you already know the
point values from a file this is not necessary.
    Random rnd = new Random(DateTime.Now.Millisecond);
    const int YMIN = 0;
    const int YSPAN = 1000;
    const int XMIN = 0;
    const int XSPAN = 1000;

    //  In  a  loop  we  are  creating  100  different  points  using  the  box
established above.
    for (int i = 0; i < 100; i++)
    {
        // A coordinate is the simple X and Y location
        Coordinate c = new Coordinate(XMIN + rnd.NextDouble() * XSPAN, YMIN +
rnd.NextDouble() * YSPAN);
```

```
        // A point has geoemtry capabilities like testing intersection with
polygons etc.
        Point pt = new Point(c);
        // A feature also has attributes related to the featureset
        // Features can be created directly, passing the point into the
constructor, but there is a glitch
        // right now that may not update the DataRow property of the feature
correctly once it is added.
        IFeature currentFeature = fs.AddFeature(pt);

        // Working with the current feature allows you to control attribute
content as well as the feature content.
        currentFeature.DataRow["Elevation"] = rnd.Next(0, 100);
    }
    fs.SaveAs(@"C:\test.shp", true);
}
```

Create Random LineStrings

A LineString is a kind of ogc feature that represents a continuous set of points connected by linear segments. A single LineString will not be broken into multiple parts. They are usually used to represent roads, railroads or rivers. Creating a new Shapefile that contains some randomly created lines is shown below. Features represent a coupling of geometric shapes with non-geometric attributes. In this case the elevation value corresponds to an attribute of the integer data type. Each feature has exactly one measured elevation value, regardless of the number of coordinates in the geometric shape.

Cannot resolve image macro, invalid image name or id.

```
using MapWindow.Data;
using MapWindow.Geometries;
using MapWindow.Projections;
private void BuildLines()
{
    // Setup a new "shapefile" by using the featureset object
    FeatureSet fs = new FeatureSet(FeatureTypes.Line);
    // You can control things like the projection with this object
    fs.Projection                                                       =
KnownCoordinateSystems.Projected.UtmNad1983.NAD1983UTMZone11N;
    // The DataTable is a standard .Net DataTable, so you can add columns the
normal way and use with DataGrid controls
    fs.DataTable.Columns.Add("Elevation", typeof(int));

    // Set up the specs for creating random points.  If you already know the
point values from a file this is not necessary.
    Random rnd = new Random(DateTime.Now.Millisecond);
    const int YMIN = 0;
    const int YSPAN = 1000;
    const int XMIN = 0;
    const int XSPAN = 1000;

    //  In  a  loop  we  are  creating  100  different  points  using  the  box
established above.
    for (int i = 0; i < 10; i++)
    {
        // An array of coordinates that defines a single, continuous line
        Coordinate[] coords = new Coordinate[10];
        for(int j = 0; j < 10; j++)
        {
            // A coordinate is the simple X and Y location
            coords[j] = new Coordinate(XMIN + rnd.NextDouble() * XSPAN, YMIN
+ rnd.NextDouble() * YSPAN);
        }

        Coordinate c = new Coordinate(XMIN + rnd.NextDouble() * XSPAN, YMIN +
rnd.NextDouble() * YSPAN);
        // A point has geoemtry capabilities like testing intersection with
polygons etc.
        LineString ls = new LineString(coords);
        // A feature also has attributes related to the featureset
        // Features can be created directly, passing the point into the
constructor, but there is a glitch
        // right now that may not update the DataRow property of the feature
correctly once it is added.
        IFeature currentFeature = fs.AddFeature(ls);

        // Working with the current feature allows you to control attribute
content as well as the feature content.
        currentFeature.DataRow["Elevation"] = rnd.Next(0, 100);
    }
    fs.SaveAs(@"C:\test.shp", true);
}
```

Create a Random MultiLineString

A MultiLineString is a kind of ogc feature that represents a continuous set of points connected by linear segments, but can have several, discontinuous parts associated with a single entity. They are usually used to represent roads, railroads or rivers. Creating a new Shapefile that contains some randomly created MultiLineStrings is shown below. Features represent a coupling of geometric shapes with non-geometric attributes. In this case the elevation value corresponds to an attribute of the integer data type. Each feature has exactly one measured elevation value, regardless of the number of coordinates in the geometric shape.

Cannot resolve image macro, invalid image name or id.

```
using MapWindow.Data;
using MapWindow.Geometries;
using MapWindow.Projections;
/// <summary>
/// MultiLineString represents several, disconnected lines.  MultiPoint
/// and MultiPolygon work the same way, so this example covers all of these.
/// For MultiPoint, make sure the feature specification is multi-point or
else
/// the shapefile may not save correctly.  The other multi-formats are
supported
/// as multi-part lines/polygons in the shapefile.
/// </summary>
private void BuildMultiLineString()
{
    // Setup a new "shapefile" by using the featureset object
    FeatureSet fs = new FeatureSet(FeatureTypes.Line);
    // You can control things like the projection with this object
    fs.Projection                                                       =
KnownCoordinateSystems.Projected.UtmNad1983.NAD1983UTMZone11N;
    // The DataTable is a standard .Net DataTable, so you can add columns the
normal way and use with DataGrid controls
    fs.DataTable.Columns.Add("Elevation", typeof(int));

    // Set up the specs for creating random points.  If you already know the
point values from a file this is not necessary.
    Random rnd = new Random(DateTime.Now.Millisecond);
    const int YMIN = 0;
    const int YSPAN = 1000;
    const int XMIN = 0;
    const int XSPAN = 1000;

    LineString[] strings = new LineString[10];
    //  In  a  loop  we  are  creating  100  different  points  using  the  box
established above.
    for (int i = 0; i < 10; i++)
    {
        // An array of coordinates that defines a single, continuous line
        Coordinate[] coords = new Coordinate[10];
        for (int j = 0; j < 10; j++)
        {
            // A coordinate is the simple X and Y location
            coords[j] = new Coordinate(XMIN + rnd.NextDouble() * XSPAN, YMIN
+ rnd.NextDouble() * YSPAN);
        }

        Coordinate c = new Coordinate(XMIN + rnd.NextDouble() * XSPAN, YMIN +
rnd.NextDouble() * YSPAN);
        //  A  point  has  geoemtry  capabilities  like  testing  intersection  with
polygons etc.
        LineString ls = new LineString(coords);

        // collect all the LineStrings we create in a single array.
        strings[i] = ls;

    }

    MultiLineString mls = new MultiLineString(strings);

    // A feature also has attributes related to the featureset
    IFeature currentFeature = fs.AddFeature(mls);

    //  Working  with  the  current  feature  allows  you  to  control  attribute
content as well as the feature content.
    currentFeature.DataRow["Elevation"] = rnd.Next(0, 100);

    fs.SaveAs(@"C:\test.shp", true);
}
```

Create Random Circular Polygons

A Polygon is a kind of ogc feature that represents a continuous geographic area that is identified by a closed curve that is comprised of linear segments. A Polygon with mutliple parts is a MultiPolygon. They are usually used to represent areas like states, counties, lakes or continents. The code below chooses a random central location and radius and then generates a circular polygon described by 36 separate points for each polygon feature. Features represent a coupling of geometric shapes with non-geometric attributes. In this case the elevation value corresponds to an attribute of the integer data type. Each feature has exactly one measured elevation value, regardless of the number of coordinates in the geometric shape.

Cannot resolve image macro, invalid image name or id.

```
using MapWindow.Data;
using MapWindow.Geometries;
using MapWindow.Projections;
private void BuildPolygon()
{
    // Setup a new "shapefile" by using the featureset object
    FeatureSet fs = new FeatureSet(FeatureTypes.Line);
    // You can control things like the projection with this object
    fs.Projection                                                      =
KnownCoordinateSystems.Projected.UtmNad1983.NAD1983UTMZone11N;
    // The DataTable is a standard .Net DataTable, so you can add columns the
normal way and use with DataGrid controls
    fs.DataTable.Columns.Add("Elevation", typeof(int));

    // Set up the specs for creating random points.  If you already know the
point values from a file this is not necessary.
    Random rnd = new Random(DateTime.Now.Millisecond);
    const int YMIN = 0;
    const int YSPAN = 1000;
    const int XMIN = 0;
    const int XSPAN = 1000;

    //  In  a  loop  we  are  creating  100  different  points  using  the  box
established above.
    for (int i = 0; i < 10; i++)
    {
        // random center
        Coordinate  center  =  new  Coordinate(XMIN  +  rnd.NextDouble()  *  XSPAN,
YMIN + rnd.NextDouble() * YSPAN);
        // random radius from 10 to 100
        double radius = 10 + rnd.Next()*90;
        // An array of coordinates that defines a single, continuous line
        Coordinate[] coords = new Coordinate[36];
        // Filled regions are defined by ArcGIS as clockwise
        for (int j = 35; j >= 0; j--)
        {
            // A coordinate is the simple X and Y location
            coords[j]  =  new  Coordinate(center.X  +  radius  *  Math.Cos(j  *
Math.PI/18 ), center.Y + radius * Math.Sin(j * Math.PI/18));
        }


        //  A  point  has  geoemtry  capabilities  like  testing  intersection  with
polygons etc.
        Polygon pg = new Polygon(coords);
        // A feature also has attributes related to the featureset
        //  Features  can  be  created  directly,  passing  the  point  into  the
constructor, but there is a glitch
        // right now that may not update the DataRow property of the feature
correctly once it is added.
        IFeature currentFeature = fs.AddFeature(pg);

        //  Working  with  the  current  feature  allows  you  to  control  attribute
content as well as the feature content.
        currentFeature.DataRow["Elevation"] = rnd.Next(0, 100);
    }
    fs.SaveAs(@"C:\test.shp", true);
}
```

Create Random Circular Polygons With Holes

A Polygon is a kind of ogc feature that represents a continuous geographic area that is identified by a closed curve that is comprised of linear segments. A Polygon with mutliple parts is a MultiPolygon. While a Polygon is not allowed to have multiple separate parts, it can have any number of holes. A hole is defined by a linear ring and describes an area that will not be filled. A Dough-nut shape would be an example of a polygon with a single hole. The holes should not overlap. Examples of polygons with holes might be a lake with an island, or a . The code below chooses a random central location and radius and then generates a circular polygon described by 36 separate points for each polygon feature. A smaller circle is then cut out of each polygon by a single hole. Features represent a coupling of geometric shapes with non-geometric attributes. In this case the elevation value corresponds to an attribute of the integer data type. Each feature has exactly one measured elevation value, regardless of the number of coordinates in the geometric shape.

Cannot resolve image macro, invalid image name or id.

```
using MapWindow.Data;
using MapWindow.Geometries;
using MapWindow.Projections;
private void BuildPolygonWithHole()
{
    // Setup a new "shapefile" by using the featureset object
    FeatureSet fs = new FeatureSet(FeatureTypes.Line);
    // You can control things like the projection with this object
    fs.Projection                                                            =
KnownCoordinateSystems.Projected.UtmNad1983.NAD1983UTMZone11N;
    // The DataTable is a standard .Net DataTable, so you can add columns the
normal way and use with DataGrid controls
    fs.DataTable.Columns.Add("Elevation", typeof(int));



    // Set up the specs for creating random points.  If you already know the
point values from a file this is not necessary.
    Random rnd = new Random(DateTime.Now.Millisecond);
    const int YMIN = 0;
    const int YSPAN = 1000;
    const int XMIN = 0;
    const int XSPAN = 1000;

    //  In  a  loop  we  are  creating  100  different  points  using  the  box
established above.
```

```
    for (int i = 0; i < 10; i++)
    {
        // random center
        Coordinate center = new Coordinate(XMIN + rnd.NextDouble() * XSPAN,
YMIN + rnd.NextDouble() * YSPAN);
        // random radius from 10 to 100
        double radius = 10 + rnd.Next() * 90;
        // An array of coordinates that defines a single, continuous line
        Coordinate[] coords = new Coordinate[36];
        // Filled regions are defined by Shapefile specs to be clockwise
        for (int j = 35; j >= 0; j--)
        {
            // A coordinate is the simple X and Y location
            coords[j] = new Coordinate(center.X + radius * Math.Cos(j *
Math.PI / 18), center.Y + radius * Math.Sin(j * Math.PI / 18));
        }

        Coordinate[] hole = new Coordinate[36];
        // Holes are defined as counter clockwise by shapefile spec.
        // In fact rendering rules are as long as it is in the opposite
direction
        // and overlaps the shell it will be drawn as a hole.
        for (int j = 0; j < 36; j++)
        {
            // A coordinate is the simple X and Y location
            hole[j] = new Coordinate(center.X + radius/2 * Math.Cos(j *
Math.PI / 18), center.Y + radius/2 * Math.Sin(j * Math.PI / 18));
        }
        // Create a LinearRing for the hole
        LinearRing ring = new LinearRing(hole);
        // Since multiple holes are supported, make this an array with one
member.
        LinearRing[] rings = new LinearRing[1];
        rings[0] = ring;

        // Temporarilly there is no constructor overload that just takes an
array
        // of coordinates but also has rings, so this creates the linear ring
that
        // represents the shell.
        LinearRing shell = new LinearRing(coords);

        // A point has geoemtry capabilities like testing intersection with
polygons etc.
        Polygon pg = new Polygon(shell, rings);
        // A feature also has attributes related to the featureset
        // Features can be created directly, passing the point into the
constructor, but there is a glitch
        // right now that may not update the DataRow property of the feature
correctly once it is added.
        IFeature currentFeature = fs.AddFeature(pg);

        // Working with the current feature allows you to control attribute
content as well as the feature content.
        currentFeature.DataRow["Elevation"] = rnd.Next(0, 100);
    }
    fs.SaveAs(@"C:\test.shp", true);
}
```