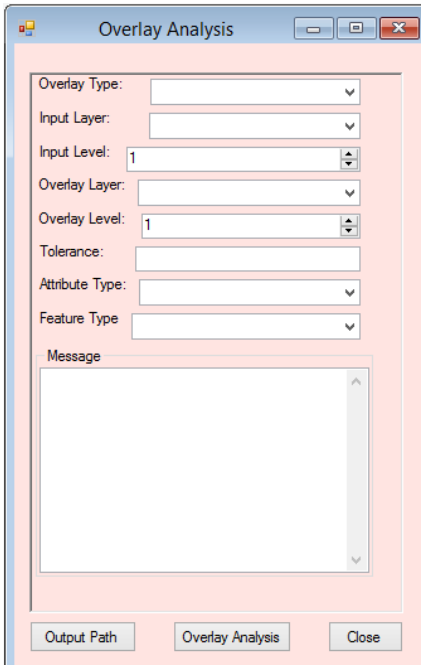
Chapter 13 Overlay Analysis

1. Add a Windows Form “OverlayAnalysis” to the project:



GUI implementation detail could be found at “OverlayAnalysis.Designer.cs” file.

2. Implementation of OverlayAnalysis:

- Class member and Construction method:

```
IHookHelper m_hookHelper;
IActiveView m_activeView;
IMap m_map;
public OverlayAnalysis(IHookHelper hookHelper)
{
    InitializeComponent();
    m_hookHelper = hookHelper;
    m_activeView = m_hookHelper.ActiveView;
    m_map = m_hookHelper.FocusMap;
}
```

- Add GetLayers, CboAddItems, CboAddItemsForSymDiff, GetFeatureLayer, Load event of form, Click Event of “Cancel”:

```
private IEnumLayer GetLayers()
{
    UID uid = new UIDClass();
    uid.Value = "{40A9E885-5533-11d0-98BE-00805F7CED21}";
    if (m_map.LayerCount != 0)
    {
        IEnumLayer layers = m_map.get_Layers(uid, true);
        return layers;
    }
    return null;
}
```

```
private void CboAddItems(ComboBox cbo)
{
    IEnumLayer layers = GetLayers();
    if (layers == null) return;
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
        if (layer is IFeatureLayer)
        {
            cbo.Items.Add(layer.Name);
        }
        layer = layers.Next();
    }
}

private void CboAddItems(ComboBox cbo, esriGeometryType geometryType)
{
    IFeatureLayer featureLayer;
    IEnumLayer layers = GetLayers();
    if (layers == null) return;
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
        if (layer is IFeatureLayer)
        {
            featureLayer = layer as IFeatureLayer;
            if (featureLayer.FeatureClass.ShapeType == geometryType)
            {
                cbo.Items.Add(layer.Name);
            }
        }
        layer = layers.Next();
    }
}

private void CboAddItems(ComboBox cbo, esriGeometryType geometryType, IFeatureLayer inputLayer)
{
    string inputName = inputLayer.Name;
    IFeatureLayer featureLayer;
    IEnumLayer layers = GetLayers();
    if (layers == null) return;
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
        if (layer is IFeatureLayer)
        {
            featureLayer = layer as IFeatureLayer;
            if (featureLayer.Name == inputName) goto label1;
            if (featureLayer.FeatureClass.ShapeType == geometryType)
            {
                cbo.Items.Add(layer.Name);
            }
        }
    label1:layer = layers.Next();
    }
}
```

```
 private void CboAddItems(ComboBox cbo, IFeatureLayer inputLayer)
 {
     string inputName = inputLayer.Name;
     IFeatureLayer featureLayer;
     IEnumLayer layers = GetLayers();
     if (layers == null) return;
     layers.Reset();
     ILayer layer = layers.Next();
     while (layer != null)
     {
         if (layer is IFeatureLayer)
         {
             featureLayer = layer as IFeatureLayer;
             if (featureLayer.Name == inputName) goto label1;
             cbo.Items.Add(layer.Name);
         }
     label1:layer = layers.Next();
     }
 }

private void CboAddItemsForSymDiff(ComboBox cbo, IFeatureLayer inputLayer)
{
    esriGeometryType inputGeometryType = inputLayer.FeatureClass.ShapeType;
    string inputName = inputLayer.Name;
    IFeatureLayer featureLayer;
    IEnumLayer layers = GetLayers();
    if (layers == null) return;
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
        if (layer is IFeatureLayer)
        {
            featureLayer = layer as IFeatureLayer;
            if (featureLayer.Name == inputName) goto label1;
            esriGeometryType overlayGeometryType = featureLayer.FeatureClass.ShapeType;
            if (overlayGeometryType == inputGeometryType)
            {
                cbo.Items.Add(layer.Name);
            }
        }
    label1: layer = layers.Next();
    }
}

private IFeatureLayer GetFeatureLayer(string layerName)
{
    if (GetLayers() == null) return null;
    IEnumLayer layers = GetLayers();
    layers.Reset();

    ILayer layer = null;
    while ((layer = layers.Next()) != null)
    {
        if (layer.Name == layerName)
            return layer as IFeatureLayer;
    }
    return null;
}
```

```
private void OverlayAnalysis_Load(object sender, EventArgs e)
{
    CboAddItems(cboSelectInputLayer);
    cboAttributeType.Enabled = false;
    cboFeatureType.Enabled = false;
    lblAttributeType.Enabled = false;
    lblFeatureType.Enabled = false;
}


private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}
```

- selectedIndexChanged event of cboOverlayerType , cboSelectInputLayer, and cboOverlayLayer:

```
private void cboOveralyerType_SelectedIndexChanged(object sender, EventArgs e)
 {
  strOveralyerType = cboOveralyerType.SelectedItem.ToString();
     switch (strOveralyerType)
     {
         case "Intersect":
             cboSelectInputLayer.Items.Clear();
             CboAddItems(cboSelectInputLayer);
             cboAttributeType.Enabled = true;
             lblAttributeType.Enabled = true;
             cboFeatureType.Enabled = true;
             lblFeatureType.Enabled = true;
             numUpDownInputLevel.Enabled = true;
             lblInputLevel.Enabled = true;
             numUpDownOverlayLevel.Enabled = true;
             lblOverlayLevel.Enabled = true;
             break;

         case "Union":
             cboSelectInputLayer.Items.Clear();
             CboAddItems(cboSelectInputLayer,
esriGeometryType.esriGeometryPolygon);
             cboAttributeType.Enabled = true;
             lblAttributeType.Enabled = true;
             cboFeatureType.Enabled = false;
             lblFeatureType.Enabled = false;
             numUpDownInputLevel.Enabled = true;
             lblInputLevel.Enabled = true;
             numUpDownOverlayLevel.Enabled = true;
             lblOverlayLevel.Enabled = true;
             break;

         case "Different":
             cboSelectInputLayer.Items.Clear();
             CboAddItems(cboSelectInputLayer);
             cboAttributeType.Enabled = true;
             lblAttributeType.Enabled = true;
             cboFeatureType.Enabled = false;
             lblFeatureType.Enabled = false;
             numUpDownInputLevel.Enabled = false;
```

```
                    lblInputLevel.Enabled = false;
                    numUpDownOverlayLevel.Enabled = false;
                    lblOverlayLevel.Enabled = false;
                    break;
                default:
                    break;
            }
            cboSelectInputLayer.Text = "";
            cboOverlayLayer.Text = "";
        }

        private void cboSelectInputLayer_SelectedIndexChanged(object sender, EventArgs e)
        {
            strInputLayer = cboSelectInputLayer.SelectedItem.ToString();
            IFeatureLayer inputLayer = GetFeatureLayer(strInputLayer);
            if (inputLayer == null) return;
            switch (strOveralyerType)
            {
                case "Intersect":
                    cboOverlayLayer.Items.Clear();
                    CboAddItems(cboOverlayLayer, inputLayer);
                    break;
                case "Union":
                    cboOverlayLayer.Items.Clear();
                    CboAddItems(cboOverlayLayer, esriGeometryType.esriGeometryPolygon,
inputLayer);
                    break;
                case "Different":
                    cboOverlayLayer.Items.Clear();
                    CboAddItemsForSymDiff(cboOverlayLayer, inputLayer);
                    break;
                default:
                    break;
            }
        }
        string strOverLayer;

        private void cboOverlayLayer_SelectedIndexChanged(object sender, EventArgs e)
        {
            strOverLayer = cboOverlayLayer.SelectedItem.ToString();
        }
```

- Leave event of txtBufferDistance:

```
double bufferDistance = 10;
object bufferDistanceField;

1 reference | Yuhui Wu, 14 hours ago | 1 change
private void txtBufferDistance_Leave(object sender, EventArgs e)
{
    if (rdoBufferDistance.Checked)
    {
        if (Information.IsNumeric(txtBufferDistance.Text))
        {
            bufferDistance = Convert.ToDouble(txtBufferDistance.Text);
            bufferDistanceField = bufferDistance;
        }
    }
}
```

- Click Event of btnOutputPath:

```
string strOutputPath = System.IO.Path.GetTempPath();
private void btnOutpuPath_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        strOutputPath = folderBrowserDialog1.SelectedPath;
    }
}
```

- IsDouble and Leave Event of txtTolerance:

```
private bool IsDouble(string s)
{
    try
    {
        Double.Parse(s);
    }
    catch
    {
        return false;
    }
    return true;
}
double tolerance = 0.1;
private void txtTolerance_Leave(object sender, EventArgs e)
{
    if (IsDouble(txtTolerance.Text))
        tolerance = Convert.ToDouble(txtTolerance.Text);
}
```

- ValueChanged event of numUpDownOverlayLevel and numUpDownInputLevel

```
int overlayLevel = 1;
private void numUpDownOverlayLevel_ValueChanged(object sender, EventArgs e)
{
    overlayLevel = (int)numUpDownOverlayLevel.Value;
}
int inputLevel = 1;
private void numUpDownInputLevel_ValueChanged(object sender, EventArgs e)
{
    inputLevel = (int)numUpDownInputLevel.Value;
}
```

- SelectedIndexChanged Event of cboAttributeType and cboFeatureType:

```
string strJoinAttributeType = "ALL";
1 reference | Yuhui Wu, 15 hours ago | 1 change
private void cboAttributeType_SelectedIndexChanged(object sender, EventArgs e)
{
    string attributeType = cboAttributeType.SelectedItem.ToString();
    switch (attributeType)
    {
        case "All Attributes":
            strJoinAttributeType = "ALL";
            break;
        case "Not Include FID":
            strJoinAttributeType = "NO_FID";
            break;
        case "Include only FID":
            strJoinAttributeType = "ONLY_FID";
            break;
        default:
            break;
    }
}

string strOutputFeatureType = "INPUT";
1 reference | Yuhui Wu, 15 hours ago | 1 change
private void cboFeatureType_SelectedIndexChanged(object sender, EventArgs e)
{
    string featureType = cboFeatureType.SelectedItem.ToString();
    switch (featureType)
    {
        case "By Input features":
            strOutputFeatureType = "INPUT";
            break;
        case "Line":
            strOutputFeatureType = "LINE";
            break;
        case "Point":
            strOutputFeatureType = "POINT";
            break;
        default:
            break;
    }
}
```

- IntersectOverlay, UnionOverlay and SymDiffOverlay

```
private IGeoProcessorResult SymDiffOverlay(Geoprocessor gp)
{
    SymDiff symDiff = new SymDiff();
    symDiff.in_features = GetFeatureLayer(strInputLayer);
    symDiff.update_features = GetFeatureLayer(strOverLayer);
    string outputFullPath = System.IO.Path.Combine
        (strOutputPath, strInputLayer + "_" + strOverLayer + "_" + "SymDiff.shp");
    symDiff.out_feature_class = outputFullPath;
    symDiff.join_attributes = strJoinAttributeType;
    symDiff.cluster_tolerance = tolerance;
    IGeoProcessorResult results = (IGeoProcessorResult)gp.Execute(symDiff, null);
    return results;
}
```

```
private IGeoProcessorResult IntersectOverlay(Geoprocessor gp)
{
    IGpValueTableObject vtobject = new GpValueTableObject()
        as IGpValueTableObject;
    vtobject.SetColumns(1);
    object row = null;
    row = GetFeatureLayer(strInputLayer);
    vtobject.AddRow(ref row);
    row = GetFeatureLayer(strOverLayer);
    vtobject.AddRow(ref row);
    IVariantArray pVarArray = new VarArrayClass();
    pVarArray.Add(vtobject);
    string outputFullPath = System.IO.Path.Combine
        (strOutputPath, strInputLayer + "_" + strOverLayer + "_" + "Intersect");
    pVarArray.Add(outputFullPath);
    pVarArray.Add(strJoinAttributeType);
    pVarArray.Add(tolerance);
    pVarArray.Add(strOutputFeatureType);
    IGeoProcessorResult results = gp.Execute(
        "intersect_analysis", pVarArray, null) as IGeoProcessorResult;
    return results;
}

private IGeoProcessorResult UnionOverlay(Geoprocessor gp)
{
    IGpValueTableObject vtobject = new GpValueTableObject()
        as IGpValueTableObject;
    vtobject.SetColumns(1);
    object row = "";
    row = GetFeatureLayer(strInputLayer);
    vtobject.AddRow(ref row);
    row = GetFeatureLayer(strOverLayer);
    vtobject.AddRow(ref row);
    IVariantArray pVarArray = new VarArrayClass();
    pVarArray.Add(vtobject);
    string outputFullPath = System.IO.Path.Combine
        (strOutputPath, strInputLayer + "_" + strOverLayer + "_" + "Union.shp");
    pVarArray.Add(outputFullPath);
    pVarArray.Add(strJoinAttributeType);
    pVarArray.Add(tolerance);
    IGeoProcessorResult results = gp.Execute
        ("Union_analysis", pVarArray, null) as IGeoProcessorResult;
    System.Runtime.InteropServices.Marshal.ReleaseComObject(pVarArray);
    System.Runtime.InteropServices.Marshal.ReleaseComObject(vtobject);
    return results;
}
```

- Click Event of “Overlay Analysis”:

```
private void btnOverlay_Click(object sender, EventArgs e)
 {
     if (strInputLayer == "" || strOverLayer == "") return;
     txtMessages.Text += "Overlay Analysis: " + strOveralyerType + "\r\n";
     txtMessages.Text += "Input Layer: " + strInputLayer + "\r\n";
     txtMessages.Text += "Overlay Layer: " + strOverLayer + "\r\n";
     txtMessages.Text += "\r\nOverlay Analysis Begin. Please wait...\r\n";
     txtMessages.Text += DateAndTime.Now.ToString() + "\r\n";
```

```
            txtMessages.Update();
            Geoprocessor gp = new Geoprocessor();
            gp.OverwriteOutput = true;
            gp.AddOutputsToMap = true;
            IGeoProcessorResult results = null;
            switch (strOveralyerType)
            {
                case "Intersect":
                    results = IntersectOverlay(gp);
                    break;
                case "Union":
                    results = UnionOverlay(gp);
                    break;
                case "Different":
                    results = SymDiffOverlay(gp);
                    break;
                default:
                    break;
            }
            txtMessages.Text += ReturnMessages(gp);
            txtMessages.Text += "\r\nOverlay Analysis Finsihed.\r\n";
            txtMessages.Text += DateAndTime.Now.ToString() + "\r\n";
            txtMessages.Text += "------------------------------------------------------
\r\n";
            ScrollToBottom(txtMessages);
            txtMessages.Update();
        }
```

3. Add A Base Command Class OverlayAnalysisCmd, and implement OnClick method:

```
public override void OnClick()
{
    // TODO: Add OverlayAnalysisCmd.OnClick implementation
    if (m_hookHelper == null) return;
    if (m_hookHelper.FocusMap.LayerCount > 0)
    {
        OverlayAnalysis overlayAnalysis =
            new OverlayAnalysis(m_hookHelper);
        overlayAnalysis.Show(m_hookHelper
            as System.Windows.Forms.IWin32Window);
    }
}
```

4. Back to MainForm, add a menu content and its click event:

```
private void overlayAnalysisToolStripMenuItem_Click(object sender, EventArgs e)
{
    ICommand command = new OverlayAnalysisCmd();
    command.OnCreate(m_mapControl.Object);
    command.OnClick();
}
```