ArcGIS Pro SDK for .NET: Editing and GeoDatase Integration

WOLFGANG KAISER
WKAISER@ESRI.COM

Session Overview

- Geometry API
- Editing API
- Sketch Tool / Construction Tool
- Edit Events
- Attribute Editing

Reminder

- Geometry manipulations and geodatabase operations can only be executed on the CIM thread.
- Must use QueuedTask.Run to get code execution to the CIM thread.

```
QueuedTask.Run(…)
```

Or

```
protected override void OnClick()
{
    MapPoint ohNo = MapPointBuilder.CreateMapPoint(-116.545, 33.83);
}
```

ConstructedOnWrongThreadException was unhandled

An unhandled exception of type 'ArcGIS.Core.ConstructedOnWrongThreadException' occurred in ArcGIS.Core.dll

ArcGIS.Core.Geometry API

- Similar to ArcGIS Runtime API.
- Contains key classes and enumerations defining geometry types such as MapPoint, MultiPoint, Envelope, Polyline, Polygon, and SpatialReference.
- Containers that house collections of the geometries as PointCollection, PartCollection and SegmentCollection.
- Support for straight lines, Bezier curves, circular arcs.

ArcGIS.Core.Geometry API (continued)

- Geometry objects are immutable – once created, their content cannot be altered.
  - Simpler to understand
  - No need for event handlers to deal with changing geometries
  - Avoid potential concurrency issues in the multithreaded environment of ArcGIS Pro.
- Use builder classes to construct and manipulate geometries objects. There is a builder for every geometry type.

GeometryBuilders

```
var mapPointBuilder = new MapPointBuilder(SpatialReferences.WGS84);
mapPointBuilder.X = -117.18;
mapPointBuilder.Y = 34.06;

MapPoint currentEsriHQ = mapPointBuilder.ToGeometry();

mapPointBuilder.X = -117.84;
mapPointBuilder.Y = 33.57;

MapPoint desiredEsriHQ = mapPointBuilder.ToGeometry();

// build the straight line from the points
var lineBuilder = new LineBuilder(currentEsriHQ, desiredEsriHQ);
LineSegment line = lineBuilder.ToSegment();

// build the polyline from the line segment
var polylineBuilder = new PolylineBuilder(line);
Polyline Polyline = polylineBuilder.ToGeometry();
```

GeometryBuilders (continued)

- Builders also have static convenience methods.

```
MapPoint pt1 = MapPointBuilder.CreateMapPoint(1.0, 1.0, SpatialReferences.WGS84);
MapPoint pt2 = MapPointBuilder.CreateMapPoint(1.0, 2.0, SpatialReferences.WGS84);
MapPoint pt3 = MapPointBuilder.CreateMapPoint(2.0, 2.0, SpatialReferences.WGS84);
MapPoint pt4 = MapPointBuilder.CreateMapPoint(2.0, 1.0, SpatialReferences.WGS84);

List<MapPoint> list = new List<MapPoint>();
list.Add(pt1);
list.Add(pt2);
list.Add(pt3);
list.Add(pt4);

Polygon polygon = PolygonBuilder.CreatePolygon(list);
```

GeometryEngine

- Utility class for performing geometric operations.
- Static methods perform manipulations such as buffer, cut, clip, offset, union, etc.

```
QueuedTask.Run(() =>
{
   var esriHQ = MapPointBuilder.CreateMapPoint(-117.18, 34.06,
          SpatialReferenceBuilder.CreateSpatialReference(4326));

   var aroundRedlandsCampus = GeometryEngine.Buffer(esriHQ, 0.035);
});
```

```
QueuedTask.Run(() =>
{
  MapPoint pt1 = MapPointBuilder.CreateMapPoint(1.0, 1.0);
  MapPoint pt2 = MapPointBuilder.CreateMapPoint(2.0, 2.5);

  Geometry geometry = GeometryEngine.Union(pt1, pt2);
  Multipoint multipoint = geometry as Multipoint;
});
```

ArcGIS.Desktop.Editing API

- Provides access to the edit templates and the sketch geometry.
- Create custom sketch tools and construction tools
  - Sketch tools modify the geometry of existing features
  - Construction tools create new features
- EditCompletedEvent captures creates, modifies, and deletes.
- Contains inspector class for streamlined attribute editing.

Edit Templates

```
// get the current template
var myTemplate = ArcGIS.Desktop.Editing.Templates.EditingTemplate.Current;


// get templates by name
ArcGIS.Desktop.Framework.Threading.Tasks.QueuedTask.Run(() =>
{
   var map = MapView.Active.Map;
   var mainTemplate = map.FindLayers("main").First().GetTemplate("Distribution");
   var mhTemplate = map.FindLayers("Manhole").First().GetTemplate("Active");

   // activate the default tool and make template current
   mhTemplate.ActivateDefaultToolAsync();
});
```

Edit Operations

- All data manipulations should be executed as part of an edit operation
- An edit operation can encapsulate multiple edits.

```
// Create an edit operation
var createOperation = new EditOperation();
createOperation.Name = "My first edit";

var template = ArcGIS.Desktop.Editing.Templates.EditingTemplate.Current;

// Queue feature creation
createOperation.Create(template, geometry);
createOperation.Create(template, anotherGeometry);

// Execute the operation
return createOperation.ExecuteAsync();
```

Save Edits

- Project.Current.SaveEditsAsync()
- Project.Current.DiscardEditsAsync()

```
if (Project.Current.HasEdits)
    await Project.Current.SaveEditsAsync();
```

Sketch Tool

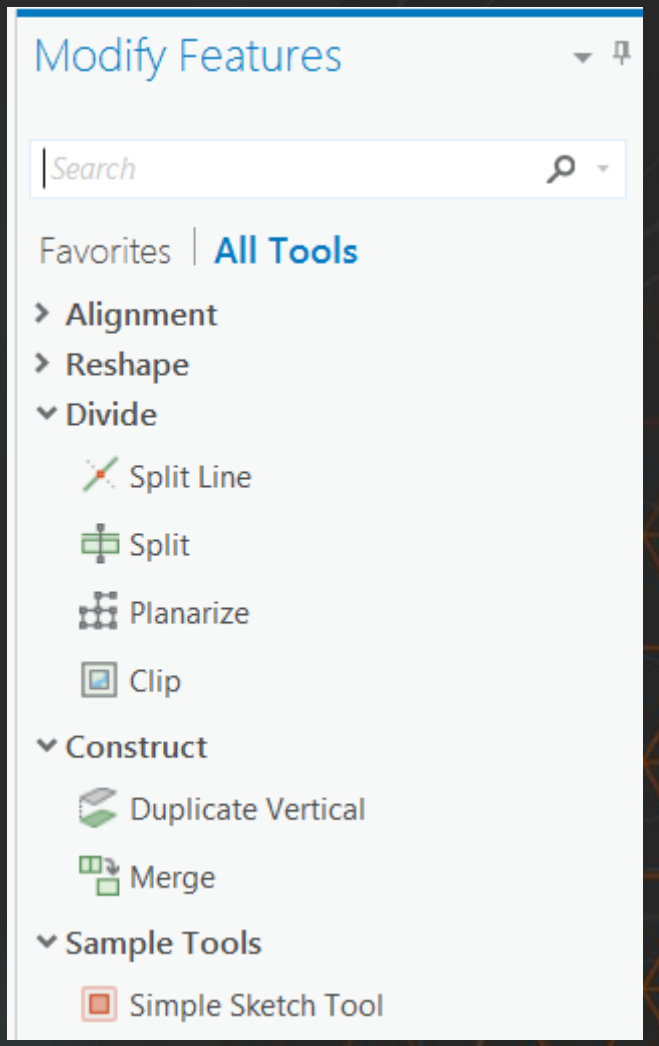
```
internal class SimpleSketchTool : MapTool
{
   public SimpleSketchTool()
   {
      IsSketchTool = true;
      SketchType = SketchGeometryType.Line;
      SketchOutputMode = SketchOutputMode.Map;
      UseSnapping = true;
   }

   /// Called when the sketch finishes. This is where we will create the
            sketch operation and then execute it.
   protected override Task<bool> OnSketchCompleteAsync(Geometry geometry)
   {
      if (geometry == null)
         return Task.FromResult(false);

      //Run on MCT
      return QueuedTask.Run(() =>
      {
         // TODO - perform the sketch operation
      }
   }
}
```

Sketch Tool (continued)

```
<controls>
   <tool id="SimpleSketchTool" caption="Simple Sketch Tool" loadOnClick="true"
      className="SimpleSketchTool" smallImage="Images\GenericButtonRed16.png"
      largeImage="Images\GenericButtonRed32.png">
      <tooltip heading="Sketch">Sketch Tool.<disabledText /></tooltip>
   </tool>
</controls>
```

- By default, the VS Template, adds the tool to a group on the ribbon
- To add the sketch tool to the Modify Features pane
  - Set categoryRefID attribute to "esri_editing_CommandList"
  - Set group attribute in content element

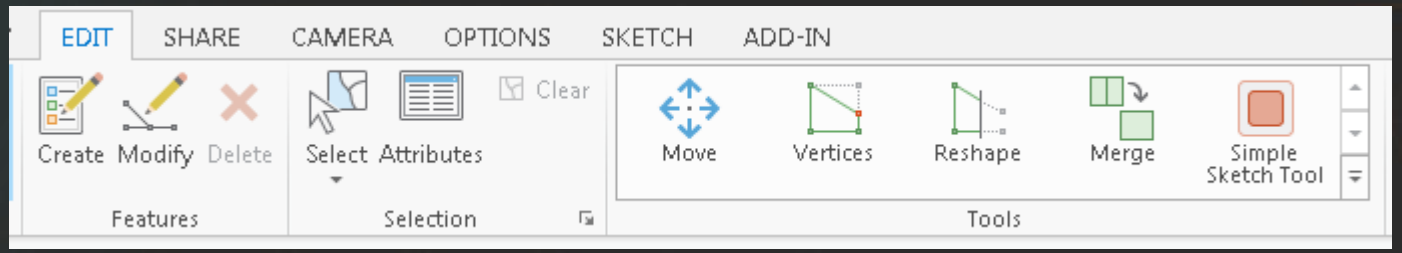
```
<controls>
   <tool id="SimpleSketchTool" caption="Simple Sketch Tool" loadOnClick="true"
      className="SimpleSketchTool" smallImage="Images\GenericButtonRed16.png"
      largeImage="Images\GenericButtonRed32.png"
      categoryRefID="esri_editing_CommandList">
      <tooltip heading="Sketch">Sketch Tool.<disabledText /></tooltip>
      <content L_group="Sample Tools" />
   </tool>
</controls>
```



Sketch Tool (continued)

- **To add the sketch tool to the Edit Tools gallery**
  - **Set values for gallery2d and gallery3d attributes in content element**

```
<controls>
   <tool id="SimpleSketchTool" caption="Simple Sketch Tool" loadOnClick="true"
      className="SimpleSketchTool" smallImage="Images\GenericButtonRed16.png"
      largeImage="Images\GenericButtonRed32.png">
      <tooltip heading="Sketch">Sketch Tool.<disabledText /></tooltip>
      <content gallery2d="true" gallery3d="false"/>
   </tool>
</controls>
```



Construction Tool

```
internal class SimpleConstructionTool : MapTool
{
   public SimpleConstructionTool ()
   {
      IsSketchTool = true;
      UseSnapping = true;
      SketchType = SketchGeometryType.Point;
   }

   /// Called when the sketch finishes. This is where we will create the sketch operation and then execute it.
   protected override Task<bool> OnSketchCompleteAsync(Geometry geometry)
   {
      if (CurrentTemplate == null || geometry == null)
         return Task.FromResult(false);

      // Create an edit operation
      var createOperation = new EditOperation();
      createOperation.Name = string.Format("Create {0}", CurrentTemplate.Layer.Name);
      createOperation.SelectNewFeatures = true;

      createOperation.Create(CurrentTemplate, geometry);
      return createOperation.ExecuteAsync();
   }
}
```

Construction Tool (continued)

```
<controls>
   <tool id="SimpleConstructionTool" categoryRefID="esri_editing_construction_point"
         caption="SimpleConstructionTool"
         className="SimpleConstructionTool" loadOnClick="true"
         smallImage="Images\GenericButtonRed16.png" largeImage="Images\GenericButtonRed32.png">
      <tooltip heading="ArcGIS Pro SDK">Default construction tool.<disabledText /></tooltip>
   </tool>
<controls>
```

- Change construction tool type by modifying both the DAML and the SketchType
- Supported categories are
  - esri_editing_construction_point
  - esri_editing_construction_polyline
  - esri_editing_construction_polygon
  - esri_editing_construction_multipoint

EditCompletedEvent

```
public void SetupEvents()
{
   var editComplete = EditCompletedEvent.Subscribe(onEditComplete);
}

protected Task onEditComplete(EditCompletedEventArgs args)
{
   IReadOnlyDictionary<MapMember, IReadOnlyCollection<long>> creates = args.Creates;
   IReadOnlyDictionary<MapMember, IReadOnlyCollection<long>> modifies = args.Modifies;
   IReadOnlyDictionary<MapMember, IReadOnlyCollection<long>> deletes = args.Deletes;

   foreach (var item in creates)
   {
      MapMember layer = item.Key;
      IReadOnlyCollection<long> oids = item.Value;

      // do something
   }

   return Task.FromResult(0);
}
```

Attributes

- Access and update attributes with the Inspector class.

```
QueuedTask.Run(() =>
{
   //Get the geometry of the selected feature.
   var selectedFeatures = MapView.Active.Map.GetSelection();
   var insp = new Inspector();
   insp.Load(selectedFeatures.Keys.First(), selectedFeatures.Values.First());
   var selGeom = insp.Shape;

   //var extPolyline = do some geometry operation.

   //Set the new geometry back on the feature
   insp.Shape = extPolyline;

   //Update an attribute value
   insp["RouteNumber"] = 42;

   //Create and execute the edit operation
   var op = new EditOperation();
   op.Name = "Extend";
   op.Modify(insp);
   op.Execute();
});
```

Attributes

- Why using Inspector versus updating attributes using rowcursors?
- Field Inspector does automatically bind your updates to the attribute grid/pane that shows up on the Editor dock pane