Measuring distances and area with ArcGIS Runtime SDK for .NET

If you are a .NET developer looking into building your own measure tool or is simply curious to know about the editing experience in the new API - [ArcGIS Runtime SDK for .NET](https://developers.arcgis.com/net/), you have come to the right place.

This blog post aims to introduce you to the following concepts:

1. Sketching with the Editor.
2. Editor vs. Map Navigation.
3. Customizing Editor Symbology.
4. Tracking Editor’s progress.

To best explain these concepts, we will be referring to two [universal apps](http://msdn.microsoft.com/en-us/library/windows/apps/dn609832.aspx) – Simple and Advanced measure tool. Each of these sample applications target both [Windows Store](http://msdn.microsoft.com/en-us/windows/apps/br211386.aspx) and [Windows Phone](http://dev.windowsphone.com/en-us/getstarted) platforms. Although not included in the samples, it is worth mentioning that the same API can also be used in an equivalent [Windows Desktop/WPF](http://msdn.microsoft.com/en-us/windows/desktop/bg161546.aspx) application.

We will be using the simple measure tool to discuss the first two sections, I & II, and the advanced measure tool to discuss the latter two sections, III & IV, with the advanced measure tool building on top of the simple measure tool.

**I – Sketching with the Editor.**

Before we could begin to measure distances and area on the map, we must first create the geometry. The *Editor* enables you to sketch geometries on the topmost layer of the map.

Each *MapView* control is paired with an Editor instance. It is through this Editor instance that you can call *RequestShapeAsync*, which is an awaitable task that returns geometry based on the specified *DrawShape* parameter.

A DrawShape.Point, for example, will return a MapPoint. DrawShape.Polyline, LineSegement, or Freehand will return a Polyline. DrawShape.Polygon, Triangle, Rectangle, Circle, Ellipse, and Arrow will return a Polygon. DrawShape.Envelope will return an Envelope. Among these DrawShapes, only DrawShape.Polyline and Polygon enable editing before the geometry is returned, which means you can insert, update and delete vertices, and also undo/redo an edit before completing the shape.

The following code puts the map in sketch mode and returns a polyline geometry upon completion of the asynchronous task.

var geometry = await MyMapView.Editor.RequestShapeAsync(DrawShape.Polyline);

**II – Editor vs. Map Navigation.**

While digitizing the geometry for measure, you may need to interact with the map by panning to a new location or maybe zooming into a region of interest.

Since the Editor responds to the same platform events used for navigation, it is important for you to know the following properties and commands. *IsActive* is a read-only property that indicates when Editor is in the middle of sketch or edit. *IsSuspended*, is the property that indicates when Editor had been suppressed to respond to any platform events. *Cancel* is the command that deactivates the Editor without returning geometry. And, *Complete* is the command that allows the Editor to return geometry before it is deactivated.

The following code enables the Suspend button based on IsActive property and handles the Complete button click event through Complete command.

<AppBarButton Icon="Pause"

Label="Suspend"

IsEnabled="{Binding IsActive, Source={StaticResource MyEditor}}" />

<AppBarButton Icon="Stop"

Label="Complete"

Command="{Binding Complete, Source={StaticResource MyEditor}}" />

**Customizing Editor Symbology.**

The following image shows the default symbology used by the edit tools of the Editor.

As you may find convenient, you are allowed to override these symbols. If like the Advanced measure tool sample, you need to display incremental distances during measure, you may want the vertices to be numbered to indicate their position in the collection, de-emphasize the mid-vertex and also disregard their selection state as seen in the image below.



To override the symbol used for the geometry shape, you can pass a new symbol to the second and optional parameter of RequestShapeAsync. To define a new symbology for any of the edit tools, you can use EditorConfiguration. To dynamically alter any of the edit tool symbol, you can sub-class the Editor and override its OnGenerateSymbol method.

You need not use both the EditorConfiguration and sub-class the Editor for they serve the same function, which is to modify the default edit tool symbol. Note, however, that EditorConfiguration will not accept CompositeSymbol as a symbol for vertex because EditorConfiguration expects a MarkerSymbol type. But should you opt to use both methods in replacing the symbols, OnGenerateSymbol will take precedence.

To override the symbol for geometry shape requested:

var geometry = await MyMapView.Editor.RequestShapeAsync(DrawShape.Polyline, symbol);

To change any of the edit tools with a symbol matching their geometry type, use EditorConfiguration.

<Page.Resources>

<symbology:SimpleMarkerSymbol x:Key="MidVertexSymbol" … />

<esri:EditorConfiguration x:Key="MyEditorConfiguration"

MidVertexSymbol="{StaticResource MidVertexSymbol}"/>

<local:MeasureEditor x:Key="MyEditor" EditorConfiguration="{StaticResource EditorConfiguration}"/>

</Page.Resources>

To dynamically alter any of the edit tools symbol or to use CompositeSymbol, sub-class the Editor.

public class MeasureEditor : Editor

{

protected override Symbol OnGenerateSymbol(GenerateSymbolInfo generateSymbolInfo)

{

if(generateSymbolInfo.GenerateSymbolType == GenerateSymbolType.Vertex || generateSymbolInfo.GenerateSymbolType == GenerateSymbolType.SelectedVertex)

{

var index = generateSymbolInfo.VertexPosition.CoordinateIndex + 1;

return new CompositeSymbol() { … };

}

return base.OnGenerateSymbol(generateSymbolInfo);

}

**III – Tracking Editor’s progress.**

If your measure tool need to calculate lengths between intermediate segments as you digitize the geometry, you can use the third and optional parameter of RequestShapeAsync to track the sketch progress. This parameter, which is of type *IProgress<GeometryEditStatus>*, is responsible for capturing every add, move or delete vertex.

To tap into progress reporting, we first declare an instance of Progress<GeometryEditStatus>, which accepts an Action delegate that takes the same parameter type before passing as a parameter to RequestShapeAsync.

var progress = new Progress<GeometryEditStatus>(OnStatusUpdated);

var geometry = await MyMapView.Editor.RequestShapeAsync(DrawShape.Polyline, symbol, progress);

Below is an example of this delegate, where *GeometryEditStatus* is used to capture the current state of geometry through its *NewGeometry* property.

private void OnStatusUpdated(GeometryEditStatus status)

{

var polyline = status.NewGeometry as Polyline;

}

GeometryEditStatus also contains other information such as *OldGeometry* and *GeometryEditAction*, which tells us whether *Add/Delete/MoveVertex* occurred. For the case of Add/MoveVertex, we can also get the *NewVertex* and in the case of Move/DeleteVertex, we can also get the *OldVertex.*

In conclusion, the Editor exposes properties, commands, configuration and override method for customization, and progress reporting – all of which can be used to enable you to sketch and perform calculations.

I hope this blog post is enough to get you started and excited with the Editor. To see another example of Measure Tool used in a bigger application, please visit the [ArcGIS Portal Viewer for .NET](https://github.com/Esri/arcgis-portalviewer-dotnet). As always, we look forward to hearing from our users. Please share with us your feedback and how you are using the Editor in your applications.