Intermediate Editing

Charlie Macleod

ESRI EUROPEAN DEVELOPER SUMMIT 2019

Session Overview

- Review of Edit Operations + Inspector
  - Basic workflows and usage
- Edit tools
  - Modify + Construction
    - UI customizations

Edit Operations Review - Workflow

- Coarse grained API for editing in Pro
  - Instantiate a new EditOperation
  - Specify the edits to be performed
    - Creates, Updates, Deletes (in any combination)
  - Execute the operation
  - Check status (as needed)

Edit Operations Review - Example

```
var editOp = new EditOperation() {
     Name = "Simple edit2",
     ErrorMessage = "'Simple edit2' failed",
     SelectModifiedFeatures = true
};

editOp.Modify(polyLayer1, 1, buffer);
editOp.Reshape(polyLayer2, 1, reshape_poly);
editOp.Modify(pointsLayer1, 28, MapPointBuilder.CreateMapPoint(6057000, 2112600));

//Execute the operations
editOp.Execute();
if (editOp.IsSucceeded) { ...
```

Inspector Review – Recap

- Convenient utility class for ***MapMember*** feature attribute access and editing
  - Primary use:
    - Get and Set feature attributes (to include SHAPE)

Inspector Review – Usage Continued

- **Get and Set existing feature attributes:**
  - **Instantiate an inspector:**

```
using ArcGIS.Desktop.Editing.Attributes;

var inspector = new Inspector();
```

  - **Load the feature attributes:**

```
inspector.Load(pointsLayer1, 28); //or LoadAsync
```

  - **Make the change(s) and Apply:**

```
inspector["FACILITYID"] = “11055.00"; //Use dictionary-style setters
inspector["NAME"] = "BLDG 52";
inspector["DESCRIPT"] = inspector["NAME"];
```

```
inspector.Apply(); //or ApplyAsync
```

Inspector Review – Usage Continued

- Integrate with EditOperation to combine with other edits:
  - Use the relevant overload that takes an Inspector
    - Typically EditOperation.Modify
  - Use EditOperation.Execute()

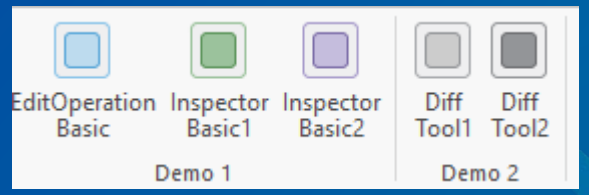
```
var inspector = new Inspector();
inspector.Load(pointsLayer1, 28);
inspector["FACILITYID"] = “11055.00";
inspector["NAME"] = "BLDG 52";

var editOp = new EditOperation() {
  Name = "Modify Facility Points",
  CancelMessage = "Modify Facility Points canceled"
};

editOp.Modify(inspector);
...
editOp.Execute();
```

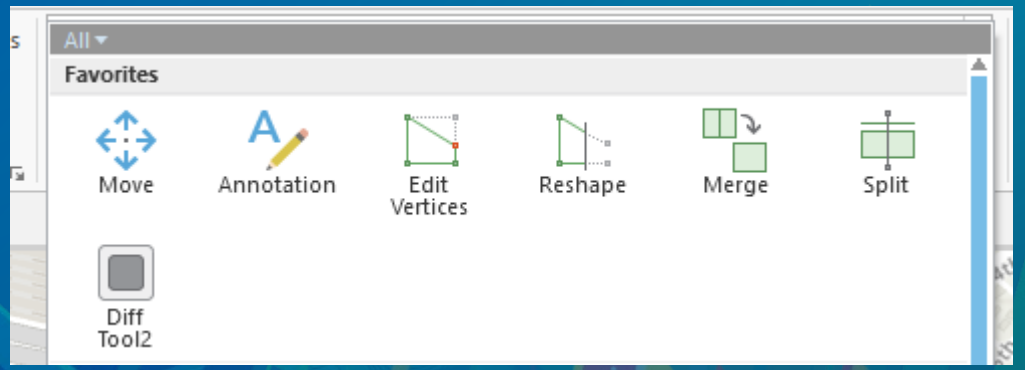
Editing Tools

- Modify
- Construction

Editing Tools - Modify

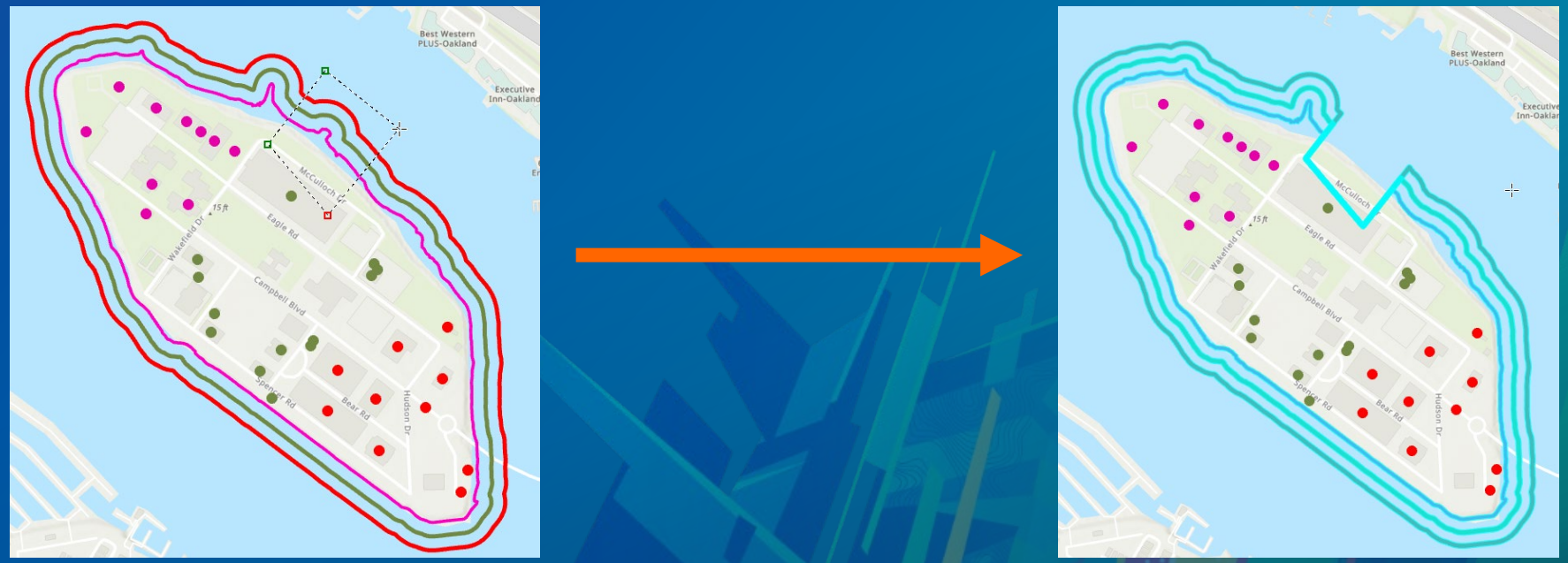
- Map Tool (aka “Sketch Tool”) used for editing features
  - By Convention, hosted on the ribbon
  - Uses the sketch to edit features
    - Integrate in Edit Operation, Inspector
  - Can host on the Editor 2D and 3D Galleries and Modify Features Dockpane
    - Via Config.daml





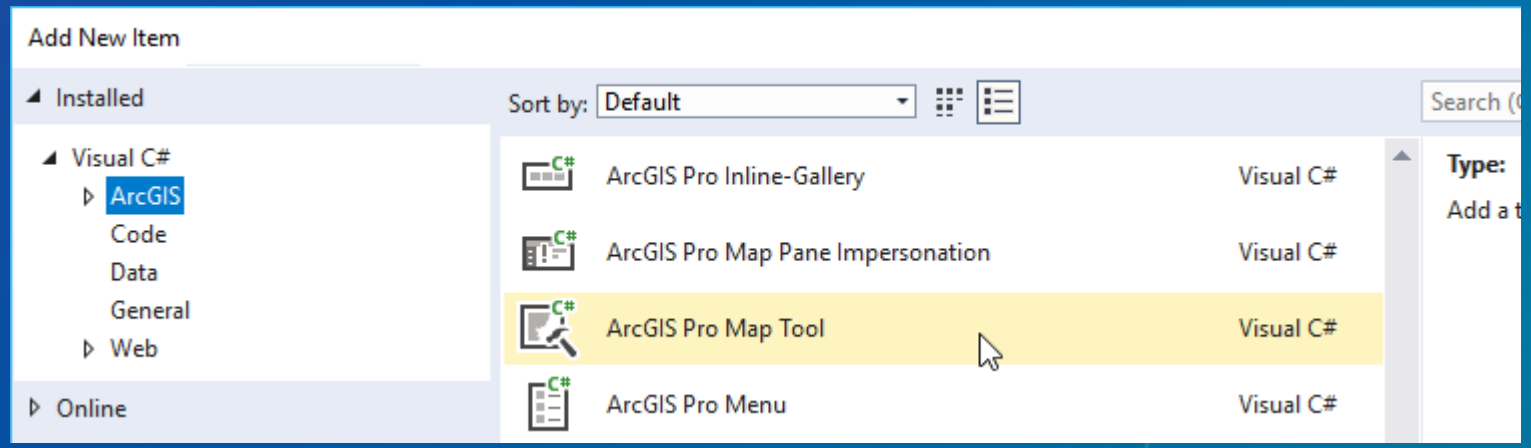
Editing Tool - Implementation

- We will be making a custom “Difference” tool
  - Edit polygon features performing a difference with the sketch
    - The “Intersection” is discarded
    - A Difference Tool is sometimes called a “Cookie Cutter”



Editing Tool - Implementation

- Run the ArcGIS Pro Map Tool template



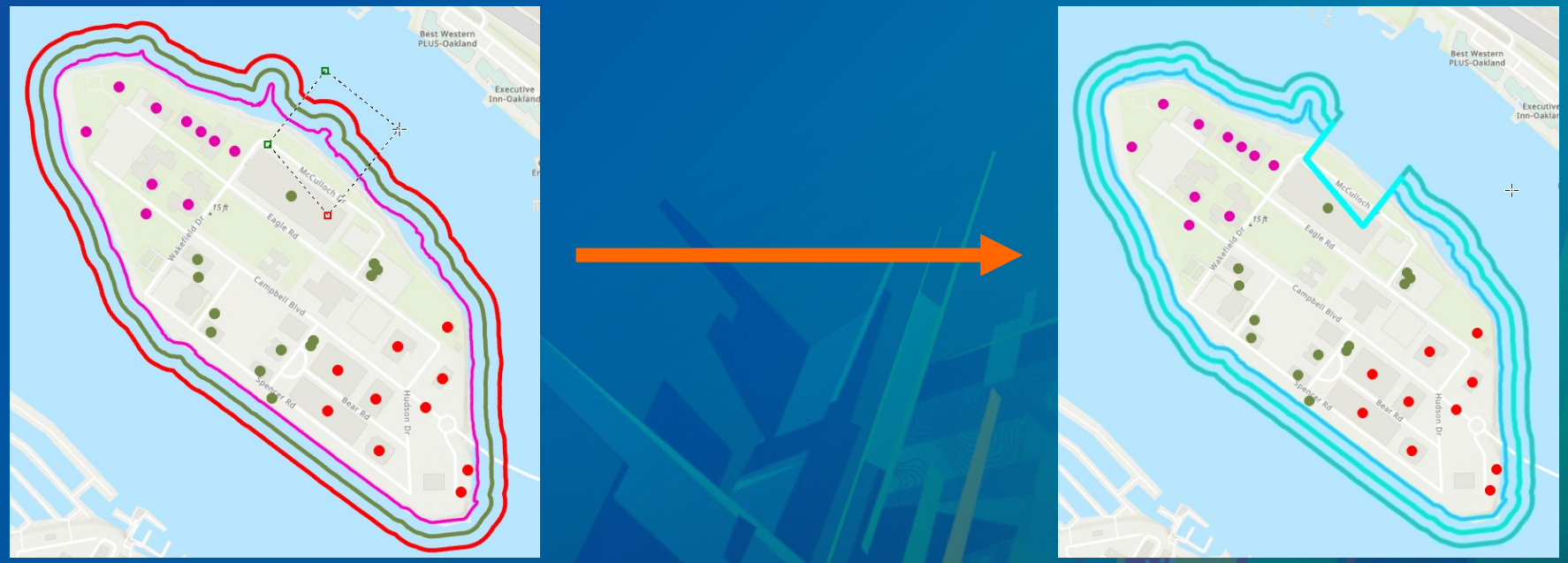
Editing Tool (Implementation continued)

- By default stubs out a map tool with `SketchGeometryType.Rectangle`
  - Change as needed to line, circle, polygon, lasso, etc.
- `OnSketchCompleteAsync(Geometry)` callback for the sketch geometry
  - Default logic calls base class (which is effectively a no-op)
  - Customize OnSketchCompleteAsync as needed with “your” editing logic

```
internal class DifferenceTool : MapTool {
   public DifferenceTool() {
      SketchType = SketchGeometryType.Rectangle;

   protected override Task<bool> OnSketchCompleteAsync(Geometry geometry) {
      // TODO – add editing logic
      return base.OnSketchCompleteAsync(geometry);
```

Editing Tool (Implementation continued)

- In our particular case – to implement our "Difference" tool:
  - Change SketchType to `SketchGeometryType.Polygon`
  - In `OnSketchCompleteAsync` we:
    - Select polygon features that intersect our sketch
    - Modify each feature shape with the "difference" between "it" and the sketch polygon



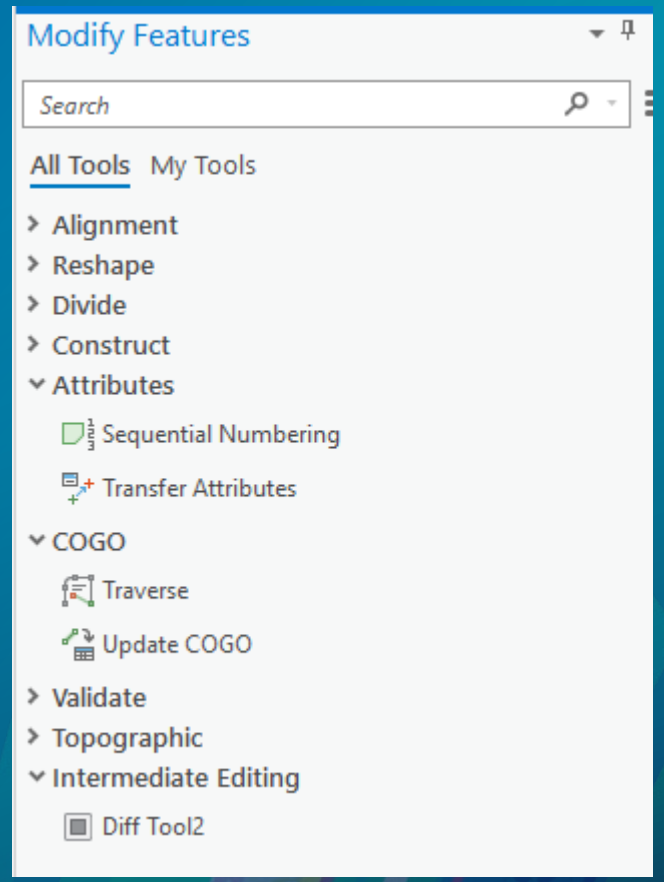
Editing Tool (Implementation continued)

```
protected override Task<bool> OnSketchCompleteAsync(Geometry sketchGeometry) {
   var poly_layers = ...;//select polygon features

   QueuedTask.Run(() => {
      var editOp = new EditOperation() { ... };

      foreach(var kvp in poly_layers) {//"poly_layers" is a Dictionary<BasicFeatureLayer, List<long>>
         var layer = kvp.Key;  var oids = kvp.Value;

         foreach(var oid in oids) {
           //get the feature shape
           var insp = new Inspector();
           insp.Load(layer, oid);
           //do the difference
           var geom = GeometryEngine.Instance.Difference((Geometry)insp["SHAPE"], sketchGeometry);
           insp["SHAPE"] = geom;
           //do the update
           editOp.Modify(insp);
         }
      }
      editOp.Execute();
```

Editing Tool – UI Integration

- Edit Config.daml to add our editing tool to the Modify Features pane:
  - Set CategoryRefID attribute to “esri_editing_CommandList”
  - Set “L_group” attribute in content element

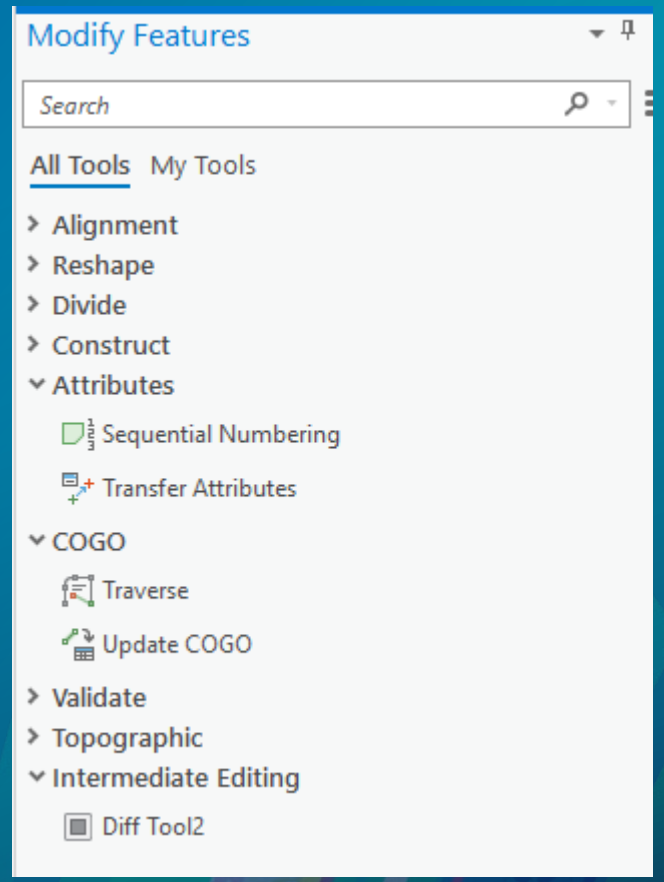
```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption=“Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
      <content L_group="Intermediate Editing"/>
   </tool>
</controls>
```



Editing Tool (UI Integration continued)

- Edit Config.daml to add our editing tool to the Modify Features pane:
  - Set CategoryRefID attribute to “esri_editing_CommandList”
  - Set “L_group” attribute in content element

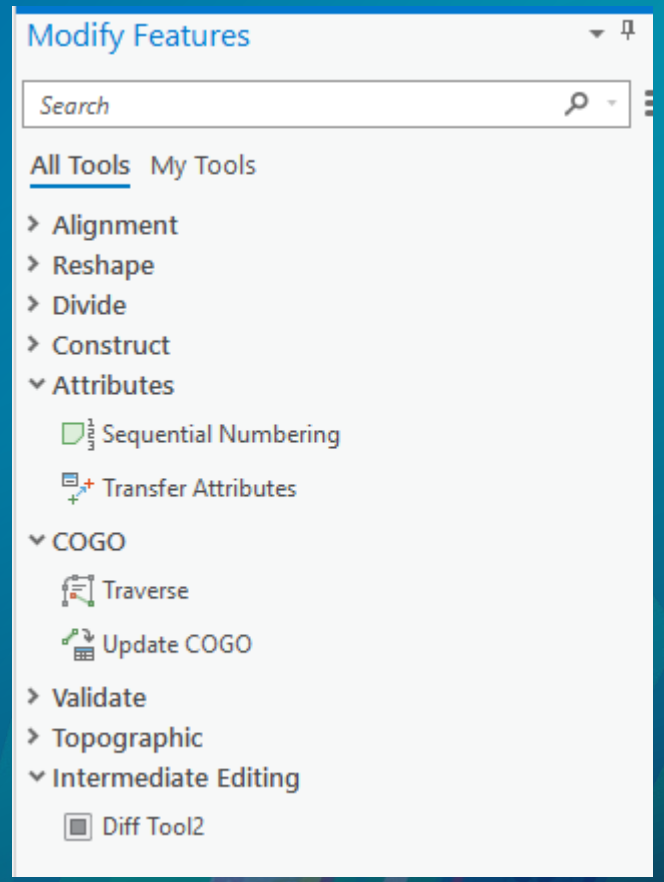
```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption="Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
      <content L_group="Intermediate Editing"/>
   </tool>
</controls>
```



Editing Tool (UI Integration continued)

- Edit Config.daml to add our editing tool to the Modify Features pane:
  - Set CategoryRefID attribute to “esri_editing_CommandList”
  - Set “L_group” attribute in content element

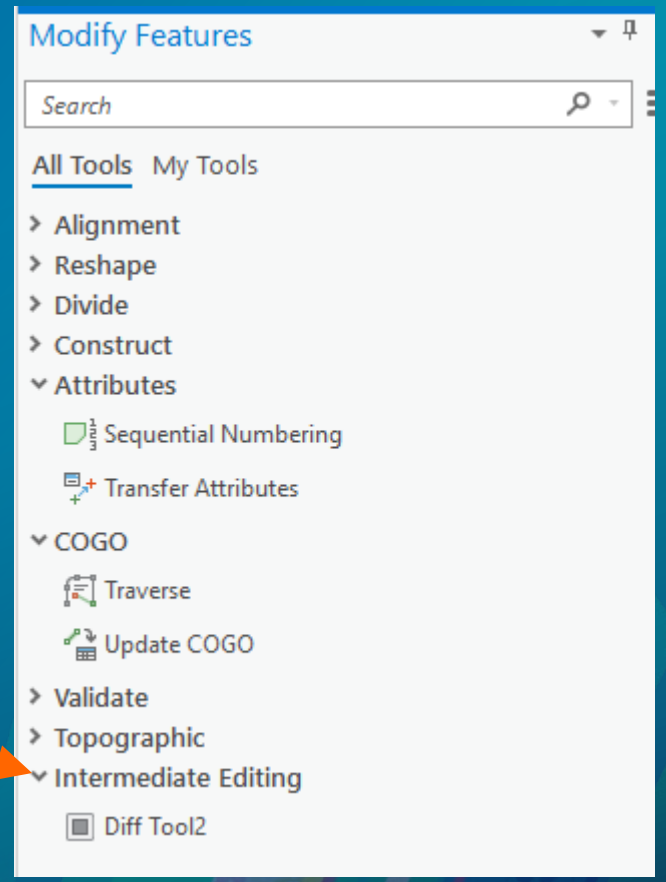
```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption="Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
      <content L_group="Intermediate Editing"/>
   </tool>
</controls>
```



Editing Tool (UI Integration continued)

- Edit Config.daml to add our editing tool to the Modify Features pane:
  - Set CategoryRefID attribute to “esri_editing_CommandList”
  - Set “L_group” attribute in content element

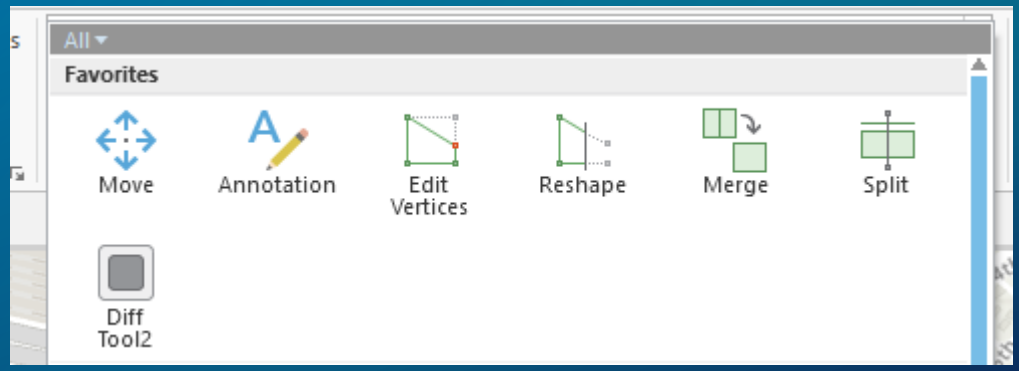
```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption="Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
      <content L_group="Intermediate Editing"/>
   </tool>
</controls>
```



Editing Tool (UI Integration continued)

- To add the our editing tool to the Edit Tools gallery…
  - Set values for gallery2d and gallery3d attributes in content element

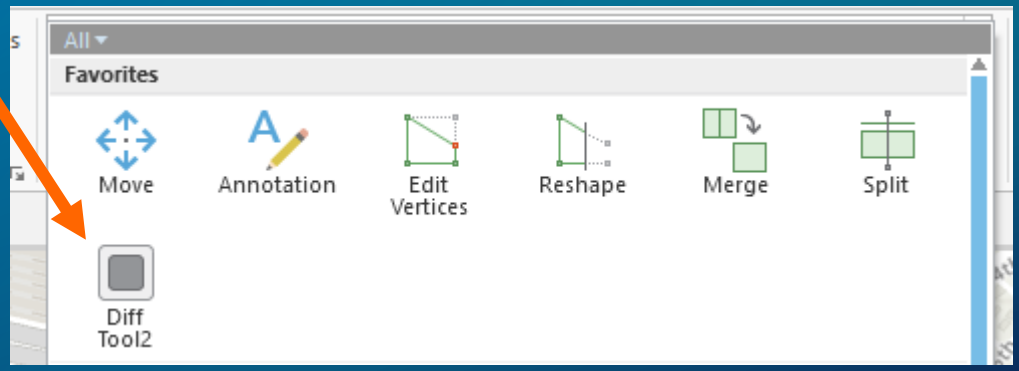
```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption=“Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
     <content L_group="Intermediate Editing“ gallery2d="true" gallery3d="false"/>
   </tool>
</controls>
```



Editing Tool (UI Integration continued)

- To add the our editing tool to the Edit Tools gallery…
  - Set values for gallery2d and gallery3d attributes in content element

```
<controls>
   <tool id="IntermediateEditing_Modify_DifferenceTool2" caption="Diff Tool2" ...
         categoryRefID="esri_editing_CommandList">
      ...
     <content L_group="Intermediate Editing" gallery2d="true" gallery3d="false"/>
   </tool>
</controls>
```

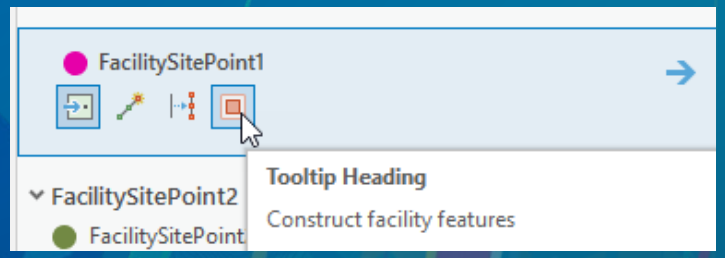


Intermediate Editing

Demo – Editing Tool

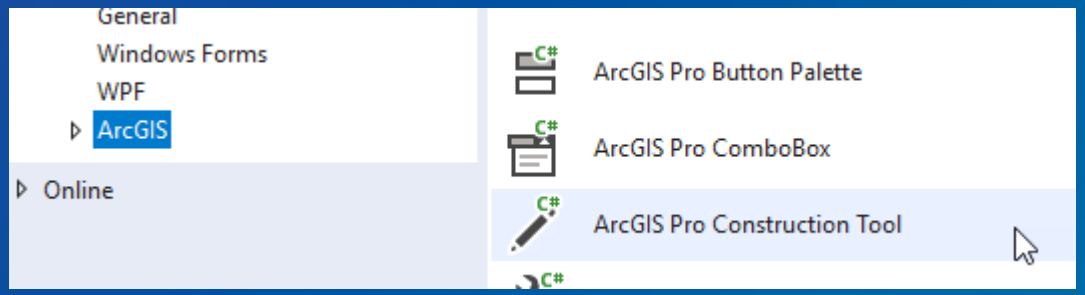
Editing Tool - Construction

- Construction Tools
- Use for creating features
  - Derived from Map Tool same as our Editing Tool for Modify
  - By Convention, hosted on the Create Features dockpane tool palette(s)
  - Uses the sketch to create features
  - Can enhance with “extra” UI for configurable options (eg buffer distance)
  - Associated with a “CurrentTemplate” for assigning default attributes



Construction Tool - Implementation

- Implement using the Pro SDK “Construction Tool” Item Template:



Construction Tool (Implementation continued)

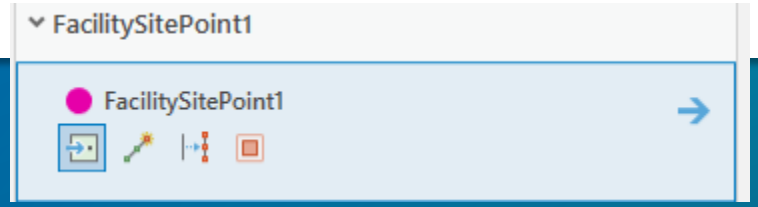
- By default stubs out a Creation tool for Point features
  - Change as needed for line, polygon, etc.
- `OnSketchCompleteAsync(Geometry geometry)` callback for the sketch geometry
  - Default creation logic to create a new feature
  - Has access to current template for assigning default attributes

```
internal class ConstructionFacilitiesTool : MapTool {
   public ConstructionFacilitiesTool() {
      ...
      SketchType = SketchGeometryType.Point;
   }
 protected override Task<bool> OnSketchCompleteAsync(Geometry geometry) {
      ...
      // 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 - Palette Placement

- In the Config.daml, tool is registered in the relevant construction tool category via *categoryRefID*
  - Adds your tool to the correct Create Pane construction tool palettes
  - Category should match your Sketch geometry type (and that of the destination feature class).

```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
            caption="ConstructionFacilitiesTool" ...>
   </tool>
<controls>
```

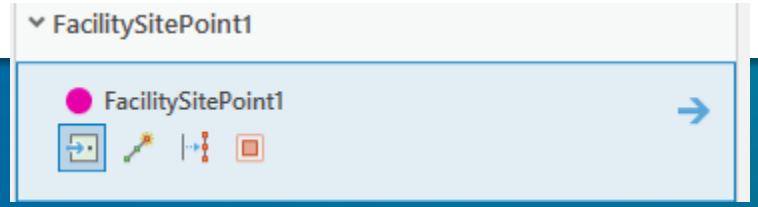


- Supported categories are
  - esri_editing_construction_point, esri_editing_construction_polyline
  - esri_editing_construction_polygon, esri_editing_construction_multipoint
  - esri_editing_construction_annotation, esri_editing_construction_dimension

Construction Tool - Palette Placement

- In the Config.daml, tool is registered in the relevant construction tool category via *categoryRefID*
  - Adds your tool to the correct Create Pane construction tool palettes
  - Category should match your Sketch geometry type (and that of the destination feature class).

```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
         caption="ConstructionFacilitiesTool" ...>
   </tool>
<controls>
```

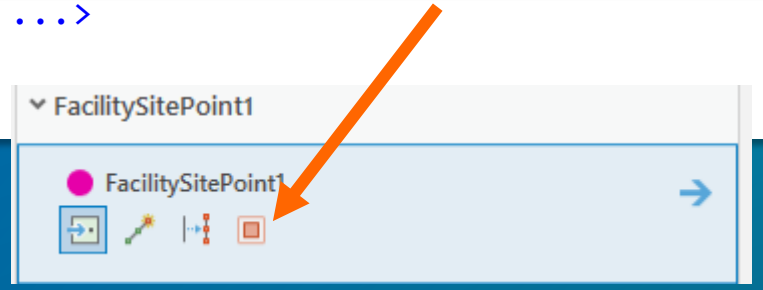


- Supported categories are
  - esri_editing_construction_point, esri_editing_construction_polyline
  - esri_editing_construction_polygon, esri_editing_construction_multipoint
  - esri_editing_construction_annotation, esri_editing_construction_dimension

Construction Tool - Palette Placement

- In the Config.daml, tool is registered in the relevant construction tool category via *categoryRefID*
  - Adds your tool to the correct Create Pane construction tool palettes
  - Category should match your Sketch geometry type (and that of the destination feature class).

```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
         caption="ConstructionFacilitiesTool" ...>
   </tool>
<controls>
```

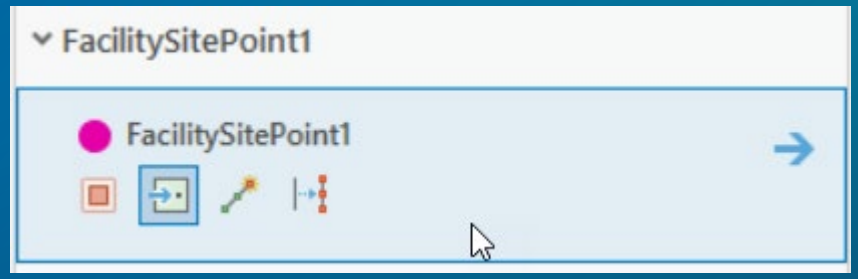


- Supported categories are
  - esri_editing_construction_point, esri_editing_construction_polyline
  - esri_editing_construction_polygon, esri_editing_construction_multipoint
  - esri_editing_construction_annotation, esri_editing_construction_dimension

Construction Tool - Palette Placement

- Control placement on the palette with a `<content … />` child element in the Config.daml
  - Add a `placeWith=daml_id`, `insert=before | after` attribute
    - Default placement is “after” if no insert attribute is specified.

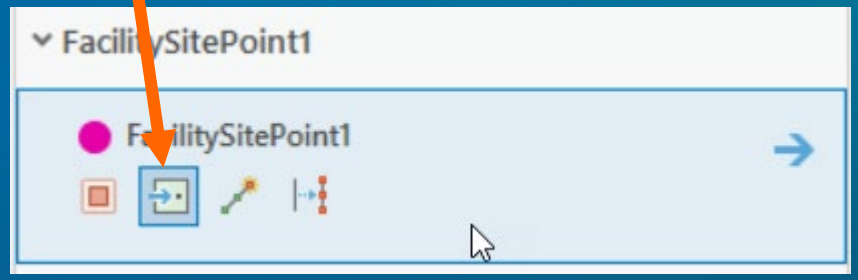
```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
         caption="ConstructionFacilitiesTool" ...>
     <content insert="before" placeWith="esri_editing_SketchPointTool"/>
   </tool>
<controls>
```



Construction Tool - Palette Placement

- Control placement on the palette with a `<content … />` child element in the Config.daml
  - Add a `placeWith=daml_id`, `insert=before | after` attribute
    - Default placement is “after” if no insert attribute is specified.

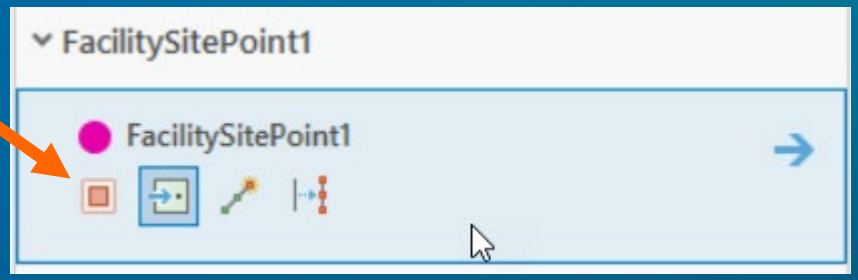
```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
         caption="ConstructionFacilitiesTool" ...>
     <content insert="before" placeWith="esri_editing_SketchPointTool"/>
   </tool>
<controls>
```



Construction Tool - Palette Placement

- Control placement on the palette with a `<content … />` child element in the Config.daml
  - Add a `placeWith=daml_id`, `insert=before | after` attribute
    - Default placement is “after” if no insert attribute is specified.

```
<controls>
   <tool id="ConstructionFacilitiesTool" categoryRefID="esri_editing_construction_point"
         caption="ConstructionFacilitiesTool" ...>
     <content insert="before" placeWith="esri_editing_SketchPointTool"/>
   </tool>
<controls>
```



Intermediate Editing

Demo – Construction Tool (1)

Construction Tool – CurrentTemplate, Defaults and “Overrides”

- A construction tool has access to the currently activated template via `this.CurrentTemplate`
  - Primary use for “defaults” is when creating features
    - Default values are stored *within* the template
    - Applied “implicitly” via `EditOperation.Create(EditingTemplate,…)`

```
var createOperation = new EditOperation();
...
createOperation.Create(CurrentTemplate, geometry);
```

Construction Tool – Defaults

- Access Defaults via the template's "Definition"
  - Definition is retrieved using `CurrentTemplate.GetDefinition()`
- Stored as a .NET Dictionary property called "`DefaultValues`"
  - Not all fields may have defaults stored – only those whose default is *non-null*
    - (i.e. if a default value is "missing", it is *assumed to be null*)
  - `DefaultValues` property can be null (no defaults stored)
  - Use "standard" .NET to get and set the defaults
    - [Note: Changes are applied using `CurrentTemplate.SetDefinition(templateDef)`]

```
//Template definition
public abstract class CIMFeatureTemplate : CIMBasicFeatureTemplate {

   // Gets or sets the default values – inherited from CIMBasicFeatureTemplate
   public Dictionary<string, object> DefaultValues { get; set;}
```

Construction Tool – Defaults

```
//Get the template definition
var templateDef = this.CurrentTemplate.GetDefinition() as CIMFeatureTemplate;

//Get Defaults (note – DefaultValues can be NULL)
string ownerVal = string.Empty;
if (templateDef.DefaultValues?.ContainsKey(“owner") ?? false) //use lower-case when
   ownerVal = (string)templateDef.DefaultValues[“owner"];   //when getting default values

// Set defaults on the template definition
if (templateDef.DefaultValues == null)
   templateDef.DefaultValues = new Dictionary<string, object>();
templateDef.DefaultValues[“OWNER”] = “US Coast Guard”;
templateDef.DefaultValues[“FACILITY”] = “BLDG 54”;

//Apply the changes back
this.CurrentTemplate.SetDefinition(templateDef);
```

Construction Tool – Overrides

- Default values can be overridden via “Overrides”
  - Access or change overrides via the Inspector property on the template
    - (not the template definition where the DefaultValues reside)
  - Overrides are initialized to the same values as the defaults each time the template is activated
    - Changes to the overrides are temporary – not persisted.
      - Lost when the tool is deactivated
  - Note: Call `Inspector.Cancel()` to manually reset overrides back to their original defaults.

```
public abstract class EditingTemplate {
   // Gets the Inspector that contains the Attributes associated with this EditingTemplate
   public abstract Inspector Inspector { get; }
```

Construction Tool – Overrides

- We use the template Inspector to assign overrides

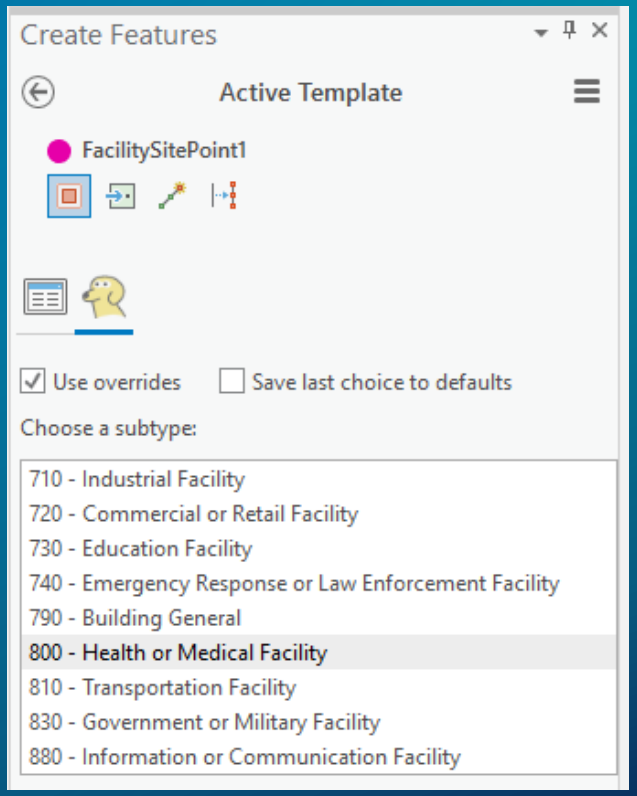
```
// Set overrides on the template Inspector “manually”
 CurrentTemplate.Inspector[“SUBTYPEFIELD”] = 790;
 CurrentTemplate.Inspector[“FEATURECODE”] = “Building General”;

 //…or Transfer from an existing feature
 CurrentTemplate.Inspector.Load(CurrentTemplate.Layer, 11);//Use Load function

 //Applied via Create with Template
 createOperation.Create(CurrentTemplate, geometry);

 //”Manually” reset
 CurrentTemplate.Inspector.Cancel();
```

Intermediate Editing

Demo – Construction Tool (2)

Construction Tool – Tool Option UI

- Custom UI for providing configurable “settings” for use with your construction tool edits
  - Eg default buffer distance, proximity distance, use of overrides, etc.
  - Hosted on the Active Template pane when its related construction tool(s) is/are activated*
  - Implement using an Embeddable Control (Pro SDK item template)



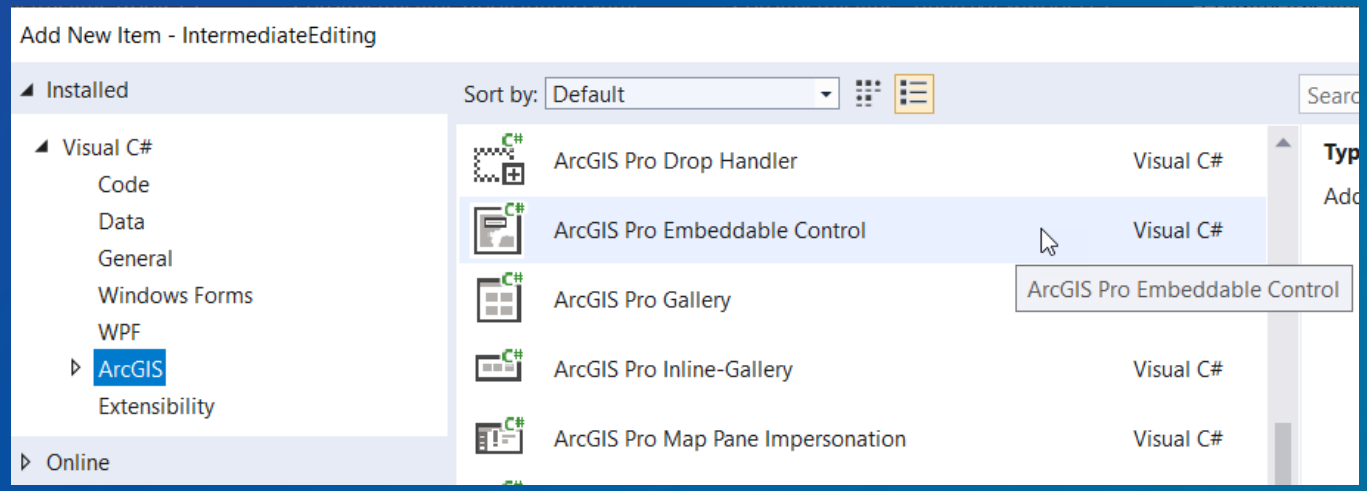
  - *And/or on Template Properties Dialog to persist custom settings in the template

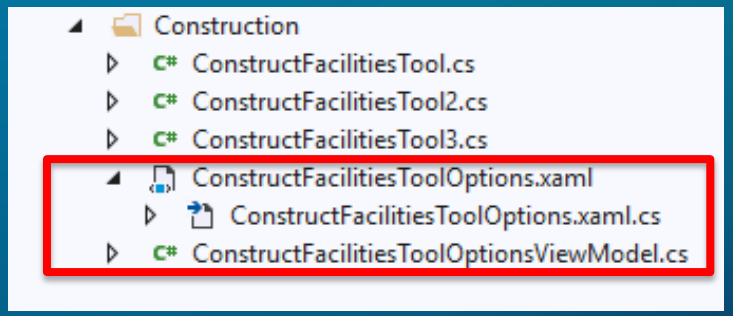
Construction Tool – Tool Option UI continued

- Procedure
  - Add an Embeddable Control (run the Pro SDK item template)
  - Register Embeddable control in the **`esri_editing_tool_options`** category in the Config.daml
  - Implement the UI (standard WPF)
  - Implement `IEditingCreateToolControl` (on the embeddable control)

Construction Tool – Tool Option UI continued

- Procedure
  - Add the Embeddable Control (via Pro SDK item template) to your VS project





Construction Tool – Tool Option UI – Config.daml

- Register Embeddable control in the **`esri_editing_tool_options`** category in the Config.daml
  - Find the embeddable control declaration
  - Change `refID` from `esri_embeddableControls` to `esri_editing_tool_options`

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
   <content insert="before" placeWith="esri_editing_SketchPointTool“ />
</tool>

<categories>
    <updateCategory refID="esri_embeddableControls">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Tool Option UI – Config.daml

- Register Embeddable control in the **`esri_editing_tool_options`** category in the Config.daml
  - Find the embeddable control declaration
  - Change `refID` from `esri_embeddableControls` to `esri_editing_tool_options`

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
   <content insert="before" placeWith="esri_editing_SketchPointTool“ />
</tool>

<categories>
    <updateCategory refID="esri_embeddableControls">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Tool Option UI – Config.daml

- Register Embeddable control in the **`esri_editing_tool_options`** category in the Config.daml
  - Find the embeddable control declaration
  - Change `refID` from `esri_embeddableControls` to `esri_editing_tool_options`

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
   <content insert="before" placeWith="esri_editing_SketchPointTool“ />
</tool>

<categories>
    <updateCategory refID="esri_editing_tool_options">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Tool Option UI – Config.daml

- Add a **content** element to the tool daml (if one is not already there)

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
   <content insert="before" placeWith="esri_editing_SketchPointTool“ />
</tool>

<categories>
    <updateCategory refID="esri_editing_tool_options">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Tool Option UI – Config.daml

- Add a **content** element to the tool daml (if one is not already there)

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
   <content insert="before" placeWith="esri_editing_SketchPointTool“ />
</tool>

<categories>
    <updateCategory refID="esri_editing_tool_options">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Tool Option UI – Config.daml

- Add a toolOptionsID specifying the id of the embeddable control in the `esri_editing_tool_options` category

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
 <content ... toolOptionsID="IntermediateEditing_Construction_ConstructFacilitiesOptions" />
</tool>

<categories>
    <updateCategory refID="esri_editing_tool_options">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

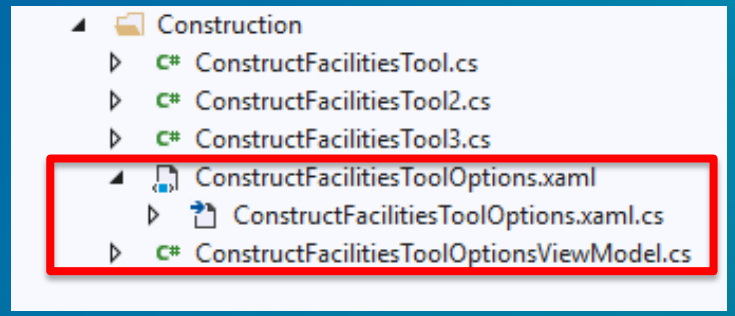
Construction Tool – Tool Option UI – Config.daml

- Add a toolOptionsID specifying the id of the embeddable control in the `esri_editing_tool_options` category

```
<tool id="IntermediateEditing_Construction_ConstructFacilitiesTool3"
       categoryRefID="esri_editing_construction_point" ...>
 <content ... toolOptionsID="IntermediateEditing_Construction_ConstructFacilitiesOptions" />
</tool>

<categories>
    <updateCategory refID="esri_editing_tool_options">
      <insertComponent id="IntermediateEditing_Construction_ConstructFacilitiesToolOptions"
 className="IntermediateEditing.Construction.ConstructFacilitiesViewModel">
        <content className="IntermediateEditing.Construction.ConstructFacilitiesView" />
      </insertComponent>
    </updateCategory>
 </categories>
```

Construction Tool – Embeddable Control

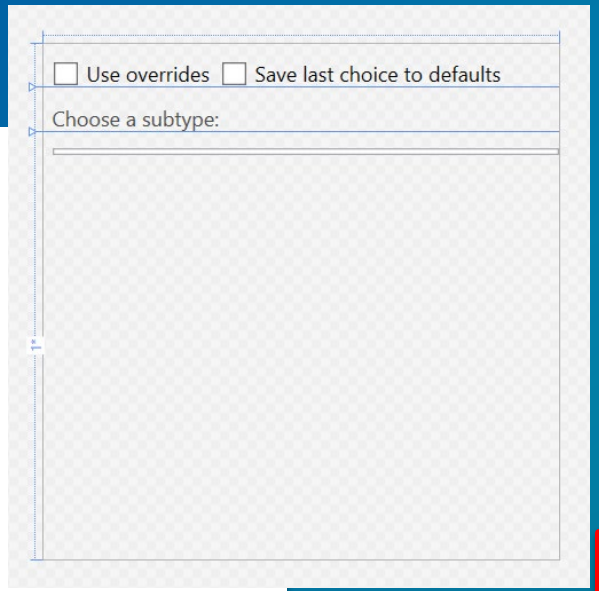
- Implement UI
  - Standard WPF/XAML to design View UI - .xaml
  - Code-behind goes in View Model - .cs

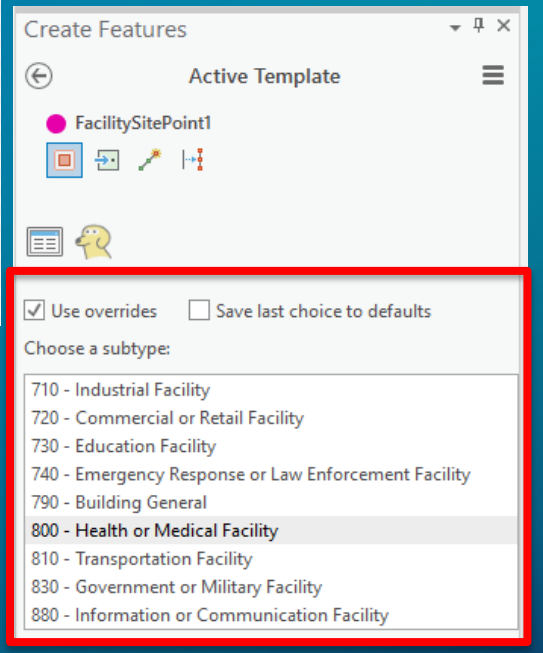


Construction Tool – Embeddable Control

- Implement UI

```
internal class ConstructFacilitiesToolOptionsViewModel:
              EmbeddableControl,
              IEditingCreateToolControl {

   public bool SaveLastSubtypeChoiceToDefaults {
      get => Module1.Current.SaveLastSubtypeChoiceToDefaults;
      set =>
       Module1.Current.SaveLastSubtypeChoiceToDefaults = value;
   }
   public bool UseSubtypeChoiceOverride {
      get => Module1.Current.UseSubtypeChoiceOverride;
      set =>
       Module1.Current.UseSubtypeChoiceOverride = value;
   }
  public List<SubtypeChoice> SubtypeChoices => Module1.Current.SubtypeChoices;
```



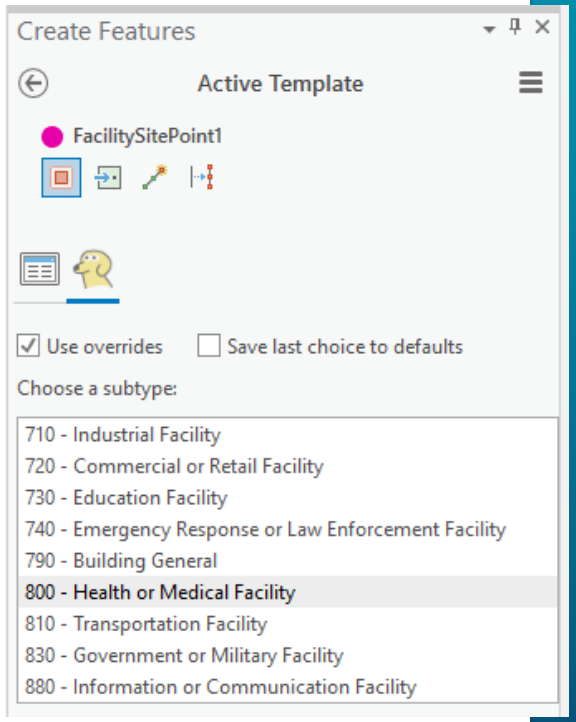


Construction Tool – Embeddable Control

- Implement `IEditingCreateToolControl` on the EmbeddableControl
  - Handles behavior of the Tool Option UI when the Tool is activated*
  - *Also handles behavior of Tool Option UI if it is used on the Template Properties Dialog

Construction Tool – EmbeddableControl - IEditingCreateToolControl

```
internal class ConstructFacilitiesToolOptionsViewModel: EmbeddableControl,
              IEditingCreateToolControl {

   public ImageSource ActiveTemplateSelectorIcon =>
      System.Windows.Application.Current.Resources["BexDog32"] as ImageSource;

   //Auto open on activate
   public bool AutoOpenActiveTemplatePane(string toolID) => true;

   public bool InitializeForActiveTemplate(ToolOptions options) => true;

   //These are for the Template Properties Dialog- leave at these defaults
   public bool IsValid => true;
   public bool IsDirty => false;
   public bool InitializeForTemplateProperties(ToolOptions options) => false;
```



Intermediate Editing

Demo – Construction Tool (3)

Intermediate Editing - Summary

- Recap of Edit Operation and Inspector
  - Edit Operations provide a coarse grained API for editing
  - Inspector is geared toward updates applied to single features

- Editing Tools
  - Modify
  - Leverage the sketch to edit existing features
  - Apply edits using a mix of edit operation and/or Inspector

  - Construction
  - Specialized “flavor” of Editing Tool
    - Focused on feature construction, hosted on Create Features pane
  - Has built-in access to the Layer Template(s)
  - Enhanced with custom UI and Tool Options

Intermediate Editing

Questions?

ArcGIS Pro SDK for .NET – Technical Sessions

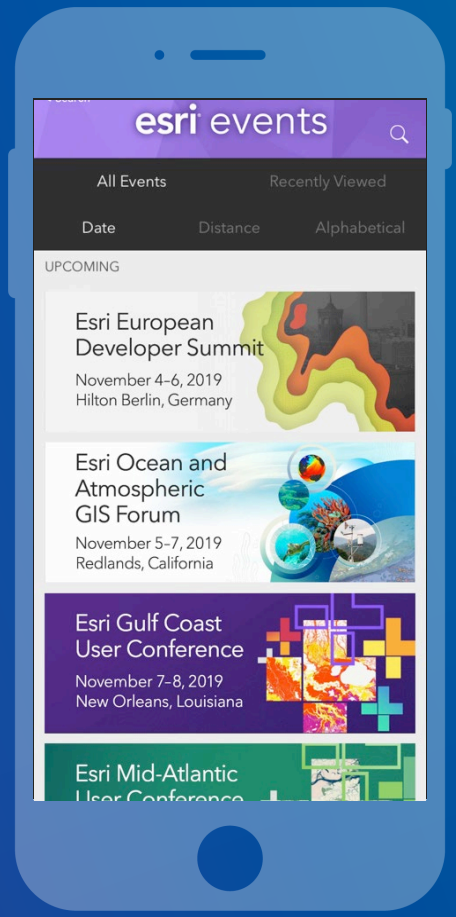
Date	Time	Session	Location
~~Mon, Nov 04~~	~~1:00 pm – 1:45 pm~~	~~Learning Customization and Extensibility~~	~~Salon Durieux~~
Tue, Nov 05	~~9:00 am – 9:45 am~~	~~Understanding the CIM, a Guide for Developers~~	~~Salon Humboldt~~
	~~2:00 pm – 2:45 pm~~	~~Intermediate Editing~~	~~Salon Durieux~~
	4:00 pm – 4:45 pm	Advanced Customization Patterns	Salon Durieux
Thu, Mar 07	2:00 pm – 2:45 pm	Learning Customization and Extensibility	Salon Humboldt

Intermediate Editing

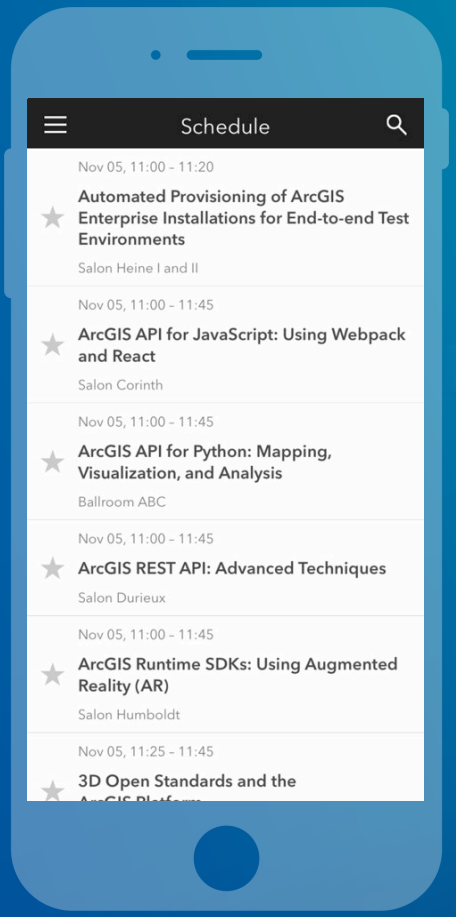
- Questions?
  - https://github.com/esri/arcgis-pro-sdk/wiki/tech-sessions#2019-berlin

Please Take Our Survey on the App

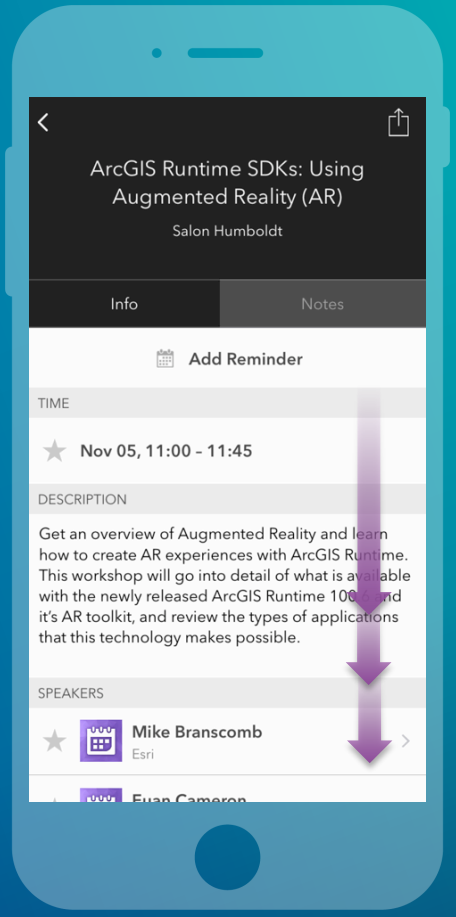
Download the Esri Events app and find your event



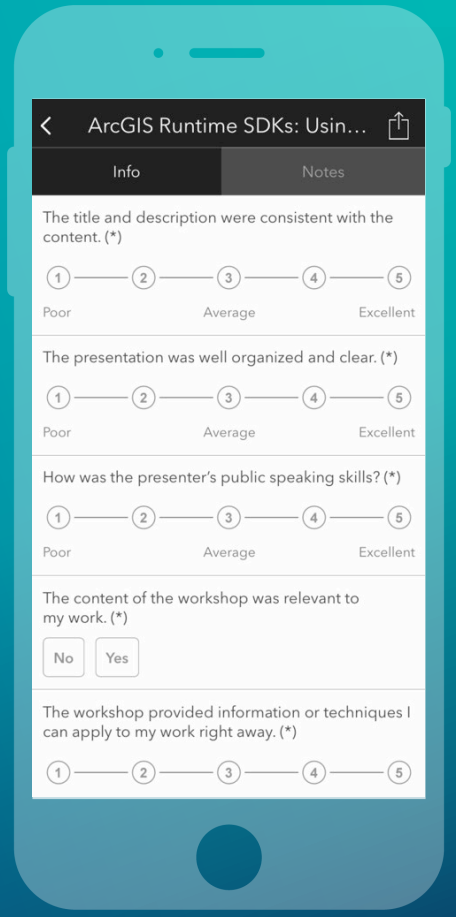
Select the session you attended



Scroll down to find the feedback section



Complete answers and select "Submit"



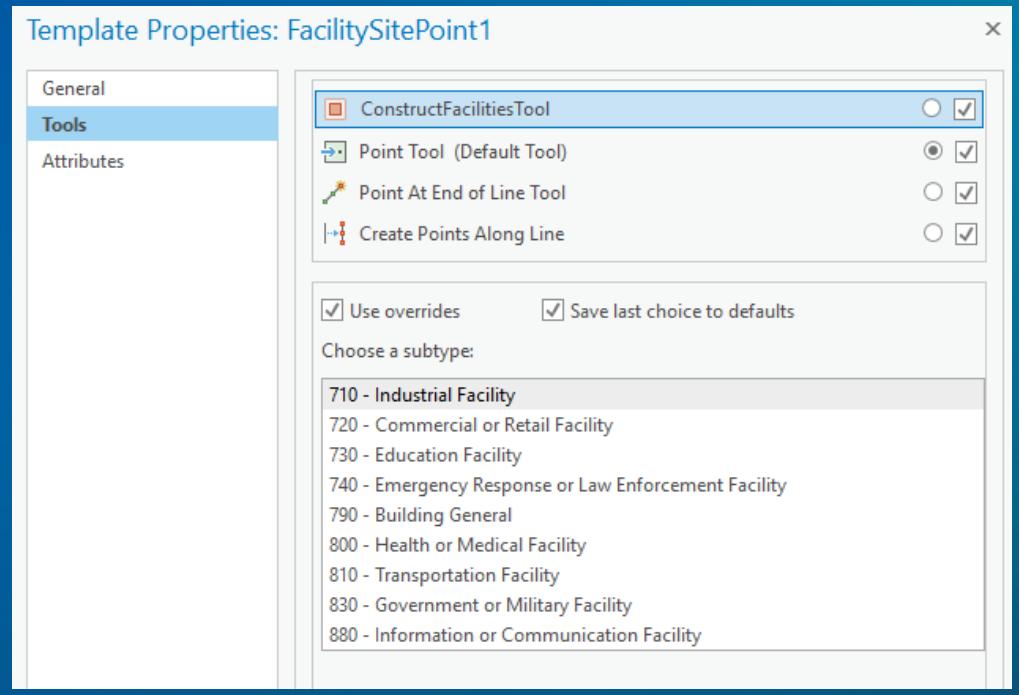

esri
THE
SCIENCE
OF
WHERE

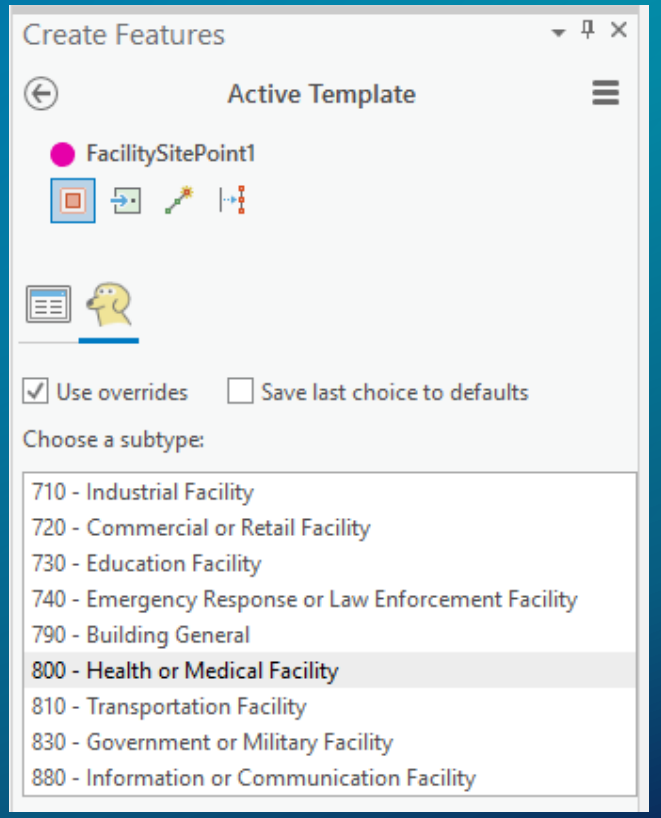
Edit Operations Review - Recap

- EditOperation is the primary pattern for Editing in the Pro API
  - Performs 3 key functions:
    - Provides a coarse grained API for editing.
    - Consolidate multiple edits into a single operation
    - Invalidates underlying caches for MapMembers edited by the operation
- Edit multiple datasets from the same datastore or different datastores
  - EditOperation ensures the required underlying edit sessions are established

Construction Tool – Tool Option UI

- Custom UI for additional tool options
  - Hosted on the Active Template pane and Template Properties
    - Active Template to use settings in our edits
    - Template Properties to persist settings in the template
  - Implement using an Embeddable Control (Pro SDK item template)





Construction Tool – ToolOptions UI - `IEditingCreateToolControl`

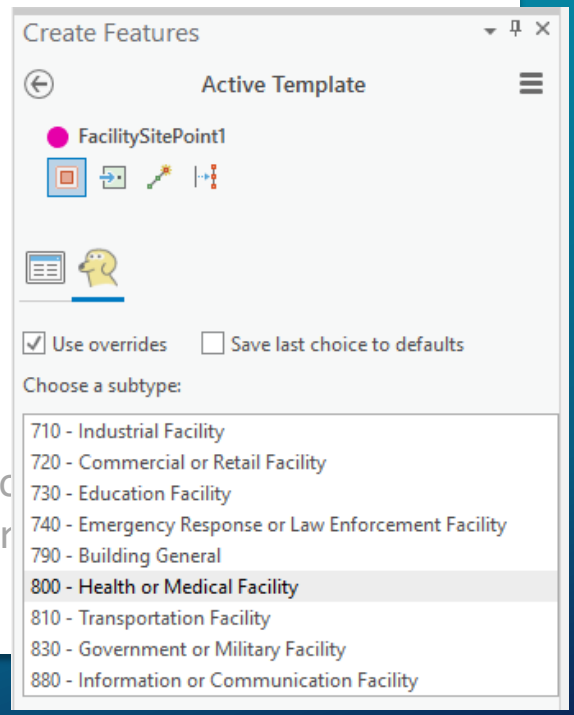
- Add implementation for `IEditingCreateToolControl` to your Embeddable Control
  - The interface is used to initialize your UI with your own “tool options”
    - “Tool Options” are provided as a .NET dictionary of key/value pairs
    - Tool options are persisted to the Template via the Template Properties dialog
      - Defaults applied on Tool activation

```
public sealed class ToolOptions : IDictionary<string, object>, ... {
   public object this[string key] { get; set; }
   public ICollection<object> Values { get; }
   public ICollection<string> Keys { get; }
```

Construction Tool – `IEditingCreateToolControl`

- Implementing **`IEditingCreateToolControl`** – For the Active Template

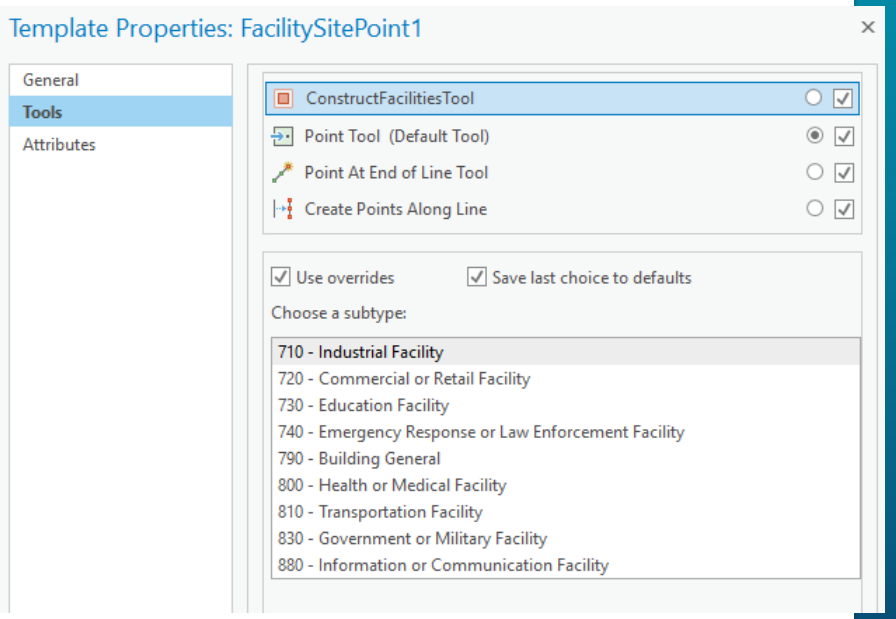
```
internal class ConstructFacilitiesToolOptionsViewModel: EmbeddableControl, IEditingCreateToolControl {
   //1. These are for the Active Template pane
   public ImageSource ActiveTemplateSelectorIcon =>
      System.Windows.Application.Current.Resources["BexDog32"] as ImageSource;

   public bool AutoOpenActiveTemplatePane(string toolID) //Auto open on activate
   public bool InitializeForActiveTemplate(ToolOptions options)//Consume tool
                                                              //options
   //2. These are for the Template Properties Dialog
   public bool IsValid => true; //False disables OK button
   public bool IsDirty => _dirty; //True indicates ToolOptions have been changed

   public bool InitializeForTemplateProperties(ToolOptions options) //Consume too
                                                                    //Template pr
```



Construction Tool – IEditingCreateToolControl

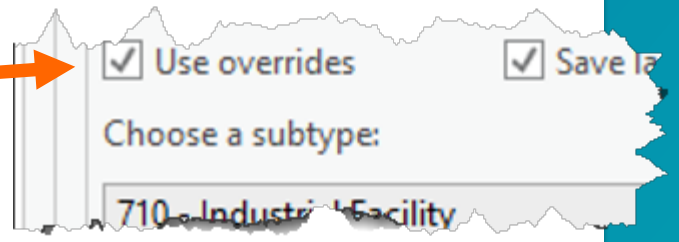
- Implementing **IEditingCreateToolControl** – for Template Properties Dialog

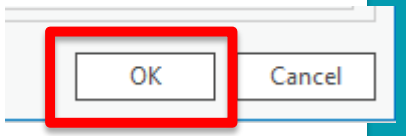
```
internal class ConstructFacilitiesToolOptionsViewModel: EmbeddableControl, IEditingCreateToolControl {
   //1. These are for the Active Template pane
   public ImageSource ActiveTemplateSelectorIcon =>
      System.Windows.Application.Current.Resources["BexDog32"] as

   public bool AutoOpenActiveTemplatePane(string toolID)
   public bool InitializeForActiveTemplate(ToolOptions options)


   //2. These are for the Template Properties Dialog
   public bool IsValid => true; //False disables OK button
   public bool IsDirty => _dirty; //True indicates ToolOptions
                                  //have been changed

   public bool InitializeForTemplateProperties(ToolOptions options) //Consume tool options for
                                                                    //Template properties
```



Construction Tool – Persisting ToolOptions *into* the Template

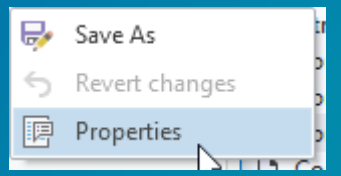
```
internal class DuplicatorOptionsViewModel: EmbeddableControl, IEditingCreateToolControl {

   private ToolOptions _toolOptions;
   public bool InitializeForTemplateProperties(ToolOptions options) {
      SetCheckboxes(options);
      _toolOptions = options; //"Capture" tool options


   public bool UseSubtypeChoiceOverride {
       get { return Module1.Current.UseSubtypeChoiceOverride; }
       set {
            Module1.Current.UseSubtypeChoiceOverride = value;
            if (_toolOptions != null) {
                 _dirty = true;//options have changed
                 _toolOptions["UseSubtypeChoiceOverride"] = value;
             }
           }

   private bool _dirty = false;
   public bool IsDirty => _dirty;//True indicates ToolOptions have been changed
```

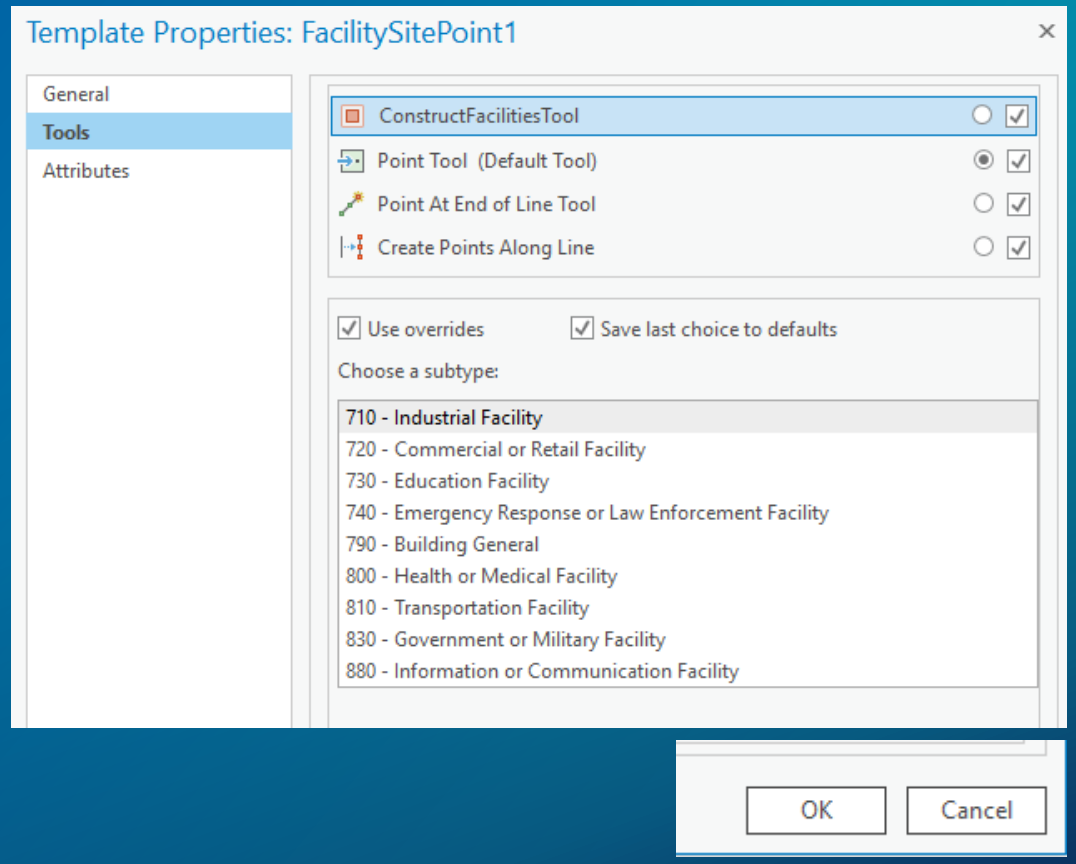




Construction Tool – Persisting ToolOptions *into* the Template

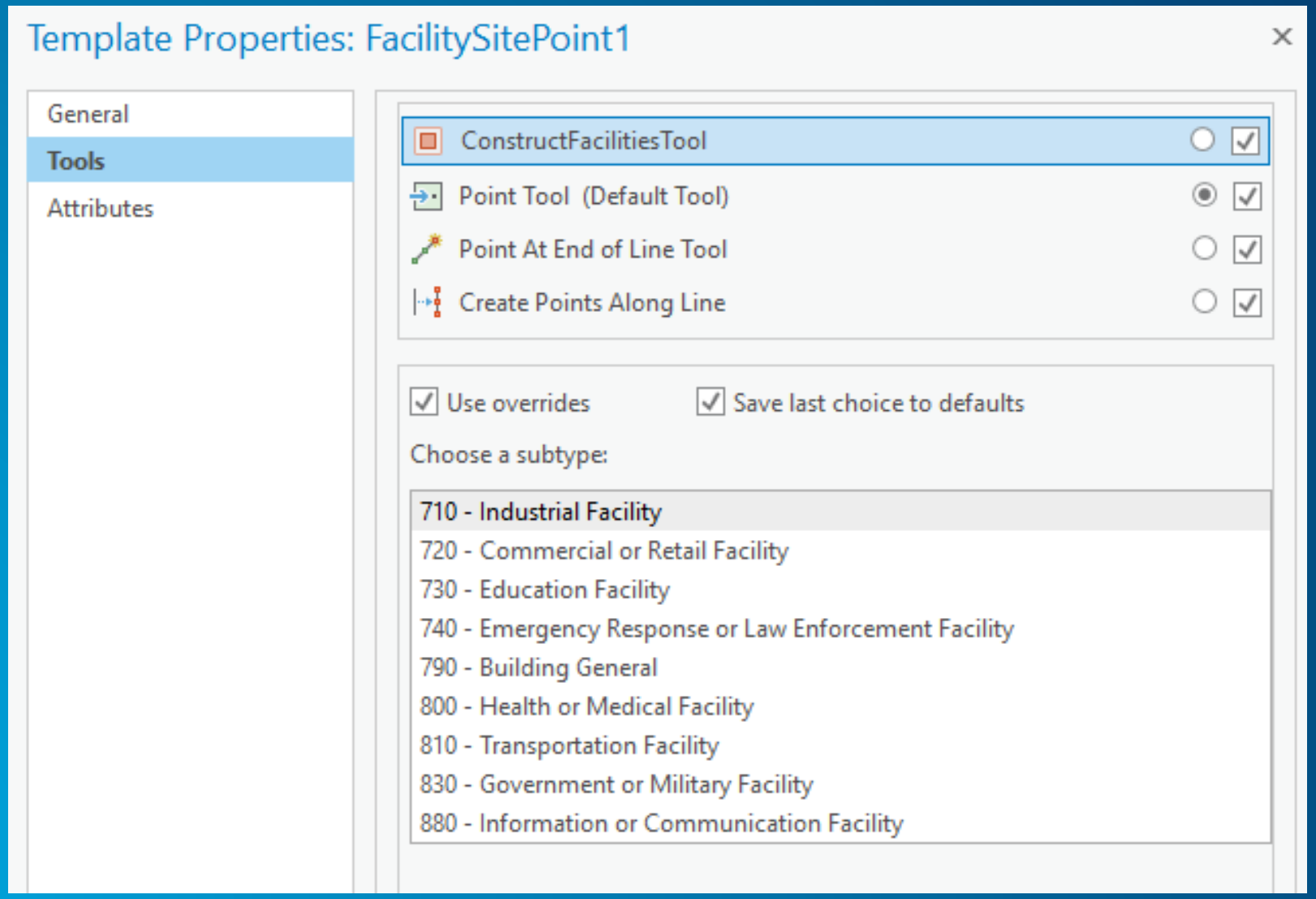
- Capture the reference to ToolOptions
  - Passed to you via `InitializeForTemplateProperties(…)`
  - E.g. store in an instance field variable
- Apply any changes to ToolOptions
- Flag the View Model as dirty
  - (recall IEditingCreateToolControl.IsDirty property…)
- On OK changes are persisted by the dialog





Construction Tool – IEditingCreateToolControl

```
internal class ConstructFacilitiesToolOptionsViewModel: EmbeddableControl, IEditingCreateToolControl {
   ....
   public bool InitializeForActiveTemplate(ToolOptions options) {//palette activation
       LoadToolOptions(options); //consume tool options
       return true; //return false to hide the UI
   }

   private ToolOptions _toolOptions;
   public bool UseSubtypeChoiceOverride { ...

   public bool InitializeForTemplateProperties(ToolOptions options) {//property dialog activation
      LoadToolOptions(options);
      _toolOptions = options; //Hold “onto” ToolOptions to persist changes!
      return true;
   }

   private void LoadToolOptions(ToolOptions options) {
     if (options.ContainsKey("UseSubtypeChoiceOverride"))
        this.UseSubtypeChoiceOverride = (bool)options["UseSubtypeChoiceOverride"];
   }
```


Template Properties: FacilitySitePoint1
General
Tools
Attributes
ConstructFacilitiesTool
Point Tool (Default Tool)
Point At End of Line Tool
Create Points Along Line
Use overrides
Save last choice to defaults
Choose a subtype:
710 - Industrial Facility
720 - Commercial or Retail Facility
730 - Education Facility
740 - Emergency Response or Law Enforcement Facility
790 - Building General
800 - Health or Medical Facility
810 - Transportation Facility
830 - Government or Military Facility
880 - Information or Communication Facility