AUTODESK UNIVERSITY

SD501372

Using Custom Grip to Enhance User Interaction with AutoCAD Entities

Norman Yuan
NormCadSoft

Learning Objectives

- Create custom grip with GripOverrule
- Create custom grip with context menu
- Use data with custom grip

Description

AutoCAD use grips showed with selected entity not only to provide visual hints of the entity, such start/end/center points, or mid/tangent points or vertices, but also more importantly provide shortcuts to various operations against the entity, such as stretch/move/rotation/scale. Other AutoCAD verticals (C3D/ARCH/MEP…) use custom grips with AEC entities extensively. Fortunately, with .NET API, we can add custom grip to AutoCAD entity for our own specialized operation against selected entity. This instruction shows a few practical examples of using custom grips.

Speaker

Norman Yuan. He started using AutoCAD in his work as civil engineer in early 1990s and soon realized how much a few lines of AutoLISP code could make using AutoCAD so much easier. That was when his programming life began: AutoLisp/ADE -> AutoCAD VBA -> generic enterprise system development, which covers all kinds of programming aspects. He has worked in engineering companies of all sizes of, from a score of employees to over tens of thousands. He is currently working as independent IT/CAD technology consultant, based in Edmonton, AB, Canada. He has 25+ years of experience of AutoCAD programming

Why Custom Grip

All AutoCAD users are familiar to “grip”, which appears when an AutoCAD entity is selected in AutoCAD editor. Grips not only shows geometrical significant information of the entity, such as end point, center, mid-point…, but also provides shortcut to some entity editing actions, such as dragging to stretch or move, or showing context menu for extra action options.

Besides standard grips that were available from earlier versions of AutoCAD, other types of grips were introduced into AutoCAD, such as the various grip types used by dynamic block, which are used for user to change block’s dynamic properties. Many AutoCAD verticals (C3D, Architecture, MEP…) use their custom grips extensively to provide better user experience when users interact with entities in AutoCAD editor.

GripOverrule exposed in AutoCAD .NET API makes it possible to us, as AutoCAD customization programmer, to create custom entity grips to allow CAD uses to interact with the entities easier, more efficient and more accurate.

Creating custom grips would involve 2 tasks:

- Create the grips visually, so they appear when the entities are selected
- Associate actions to the grip, so when use can interact with them (usually dragging, or clicking/right-clicking) to trigger certain changes/updates to the entities.

Create Custom Grip with GripData Class and GripOverrule Class

There are 2 classes in the AutoCAD .NET API for us to manipulate grips in AutoCAD: *GripData* and *GripOverrule*.

GripData class is used to define a custom grip: is visual appearance and its behavior. *CripOverrule* class is used to overrule how grips is associated with a selected entity. With this class, we can not only add custom grips to an entity, but also hide built-in grips, if necessary.

Example 1: a Do-Nothing Grip

Typically, we start from create a class derived from *GridData* class as our custom grip:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.GraphicsInterface;
```

```
using Autodesk.AutoCAD.Runtime;
using CadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace GripOverruleSample.Polygon
{
    public class PolygonGrip : GripData
    {
        public  PolygonGrip()
        {
            ForcedPickOn = false;
            GizmosEnabled = false;
            DrawAtDragImageGripPoint = false;
            IsPerViewport = false;
            ModeKeywordsDisabled = true;
            RubberBandLineDisabled = true;
            TriggerGrip = true;
            HotGripInvokesRightClick = true;
            HotGripInvokesRightClick = false;
        }

        public ObjectId EntityId { get; set; } = ObjectId.Null;

        public override bool ViewportDraw(
            ViewportDraw worldDraw,
            ObjectId entityId,
            DrawType type,
            Point3d? imageGripPoint,
            int gripSizeInPixels)
        {
            var unit = worldDraw.Viewport.GetNumPixelsInUnitSquare(GripPoint);
            var gripHeight = 3.0 * gripSizeInPixels / unit.X;
            var points = new Point3dCollection();
            var x = GripPoint.X;
            var y = GripPoint.Y;
            var offset = gripHeight / 2.0;

            points.Add(new Point3d(x - offset, y, 0.0));
            points.Add(new Point3d(x, y - offset, 0.0));
            points.Add(new Point3d(x + offset, y, 0.0));
            points.Add(new Point3d(x, y + offset, 0.0));

            Point3d center = new Point3d(x, y, 0.0);
            var radius = offset;

            worldDraw.SubEntityTraits.FillType = FillType.FillAlways;
            worldDraw.SubEntityTraits.Color = 30;
            worldDraw.Geometry.Circle(center, radius, Vector3d.ZAxis);

            return true;
        }
```

```
        public override ReturnValue OnHotGrip(ObjectId entityId, Context contex
tFlags)
        {
            return ReturnValue.Ok;
        }
    }
}
```

We override base class's *ViewportDraw()* method to draw the custom grip. Then we override the *OnHotGrip()* method to decide what action or actions to associated with the custom grip. If needed, we can also override *OnHover(), OnGripStatusChanged(), onRightClick().* In the constructor, we can turn on or off certain behaviours of the grip by setting the properties exposed by the base class *GripData*.

Once the custom GripData class is created, we go ahead to create a custom GripOverrule class, which determines when/where the custom grip show appear with a selected entity. Besides the usual Overrule class's setup (adding target RXClass to the overrule, setting overrule filter…), it is the overridden method GetGripPoints() to actually create custom grip or grips, which occurs when an entity is selected.

Depending one what kind of action is associated to the custom grip in the custom GripData class, overriding GripOverrule's *MoveGripPointsAt()* might required. The code below only creates a simple custom grip with a closed polyline, the grip has no action associated, and also can toggle showing built-in grips or not.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;

namespace GripOverruleSample.Polygon
{
    public class PolygonGripOverrule : GripOverrule
    {
        private bool _enabled = false;
        private bool _originalOverruling = false;
        private static PolygonGripOverrule _instance = null;

        public static PolygonGripOverrule Instance
        {
            get
            {
                if (_instance == null )
                {
                    _instance = new PolygonGripOverrule();
```

```
                }
                return _instance;
            }
        }

        public bool HideOriginals { get; set; } = true;

        public void EnableOverrule(bool enable)
        {
            if (enable)
            {
                if (_enabled) return;
                _originalOverruling = Overrule.Overruling;
                AddOverrule(RXClass.GetClass(typeof(Polyline)), this, false);
                SetCustomFilter();

                Overrule.Overruling = true;
                _enabled = true;
            }
            else
            {
                if (!_enabled) return;
                RemoveOverrule(RXClass.GetClass(typeof(Polyline)), this);
                Overrule.Overruling = _originalOverruling;
                _enabled = false;
            }
        }

        public override bool IsApplicable(RXObject overruledSubject)
        {
            var poly = overruledSubject as Polyline;
            if (poly!=null)
            {
                return poly.Closed;
            }
            return false;
        }

        public override void GetGripPoints(
            Entity entity,
            GripDataCollection grips,
            double curViewUnitSize,
            int gripSize,
            Vector3d curViewDir,
            GetGripPointsFlags bitFlags)
        {
            var poly = entity as Polyline;
            if (poly != null && poly.Closed)
            {
                using (var tran = entity.Database.TransactionManager.StartTrans
action())
                {
```

```
                    var pt = poly.StartPoint;
                    var grip = new PolygonGrip()
                    {
                        GripPoint = pt,
                        EntityId = entity.ObjectId
                    };
                    grips.Add(grip);

                    tran.Commit();
                }

                if (!HideOriginals)
                {
                    base.GetGripPoints(
                        entity, grips, curViewUnitSize, gripSize, curViewDir, b
itFlags);
                }
                return;
            }

            base.GetGripPoints(
                entity, grips, curViewUnitSize, gripSize, curViewDir, bitFlags)
;
        }
    }
}
```

While the code shown above is really not much useful, it shows where the grip is defined (in derived GripData class) and when it is actually drawn in AutoCAD editor. In the meantime, we have chance to decide whether we want to the selected entity only shows custom grips or show both built-in grips and the custom ones.

Example 2: Incremental Drag Grip

As aforementioned, typically, the overridden method *OnHotGrip()* of the custom *GridData* class associate an action to the user interaction with the custom grip, such as user clicks it.

In the custom grip class *IncrementDragGrip*, which is derived from class GripData, a jig that allows user to drag one end of a Line entity to change its length with in incremental interval. The code looks like:

```
public override ReturnValue OnHotGrip(ObjectId entityId, Context contextFlags)
        {
            var dwg = CadApp.DocumentManager.MdiActiveDocument;
            using (dwg.LockDocument())
            {
                var jig = new IncrementDrag.LineIncrementJig(dwg, entityId, Gri
pPoint);
```

```
                jig.Drag(DragIncrement);
            }
            CadApp.UpdateScreen();
            return ReturnValue.GetNewGripPoints;
        }
```

As the code shows, the custom grip does not care what the action is, it only plays as an action trigger: when user clicks the custom grip, a jig does its work, as if user starts a custom jig command. That is, we use a custom grip as command alternative for user to do something with the selected entity.

Note: this custom grip could be a good candidate for the grip overrule to hide the built-in grips, if we only allow user to change the line's length in given increment.

Example 3: Attribute Move Grip

The code of previous custom grip shows how to use the custom grip to trigger a quite complicated action (a jig that modifies Line entity's length). In this case, the custom grip needs to know what an external code component does.

To make a custom grip more general in terms of actions being associated, we can "inject" an action to the custom grip when it is created. This way, the same code of a custom *GridData* class can be used in different custom GripOverrle classes with different action injected, thus behaves differently when user interacts with the custom grip.

This is the custom *GripData* class *AttGrip*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.GraphicsInterface;
using Autodesk.AutoCAD.Runtime;
using CadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace GripOverruleSample.AttMove
{
    public class AttGrip : GripData
    {
        public ObjectId AttributeId { set; get; } = ObjectId.Null;
```

```
        public Action<ObjectId> ClickAction { set; get; } = null;

        public AttGrip()
        {
            ForcedPickOn = true;
            GizmosEnabled = false;
            DrawAtDragImageGripPoint = false;
            IsPerViewport = false;
            ModeKeywordsDisabled = true;
            RubberBandLineDisabled = true;
            TriggerGrip = true;
            HotGripInvokesRightClick = false;
            HotGripInvokesRightClick = false;
        }

        public override bool ViewportDraw(
            ViewportDraw worldDraw,
            ObjectId entityId,
            DrawType type,
            Point3d? imageGripPoint,
            int gripSizeInPixels)
        {
            var unit = worldDraw.Viewport.GetNumPixelsInUnitSquare(GripPoint);
            var gripHeight = 2.0 * gripSizeInPixels / unit.X;
            var points = new Point3dCollection();
            var x = GripPoint.X;
            var y = GripPoint.Y;
            var offset = gripHeight / 2.0;
            points.Add(new Point3d(x - offset, y, 0.0));
            points.Add(new Point3d(x, y - offset, 0.0));
            points.Add(new Point3d(x + offset, y, 0.0));
            points.Add(new Point3d(x, y + offset, 0.0));

            worldDraw.SubEntityTraits.FillType = FillType.FillAlways;
            worldDraw.SubEntityTraits.Color = 3;
            worldDraw.Geometry.Polygon(points);

            return true;
        }

        public override ReturnValue OnHotGrip(ObjectId entityId, Context contex
tFlags)
        {
            if (ClickAction==null)
            {
                return ReturnValue.Ok;
            }

            var dwg = CadApp.DocumentManager.MdiActiveDocument;
            using (dwg.LockDocument())
            {
                ClickAction(AttributeId);
```

```
            }

            return ReturnValue.GetNewGripPoints;
        }

        public override IEnumerable<IMenuItem> OnRightClick(
            GripDataCollection hotGrips, ObjectIdCollection entities)
        {
            return null;
        }
    }
}
```

Pay attention to the highlighted lines. It shows that when overridden OnHotGrip() method is called, the *ClickAction()* method is, as long as it is not null, executed. The custom grip does not know, or care, what is to happen.

Now, it is the custom GripOverrule's responsibility to inject certain action into the custom grip when it is created. Here is a portion of the code from the class *AttMoveGripOverrule*:

```
public override void GetGripPoints(
    Entity entity,
    GripDataCollection grips,
    double curViewUnitSize,
    int gripSize,
    Vector3d curViewDir,
    GetGripPointsFlags bitFlags)
{
    var blk = entity as BlockReference;
    if (blk != null && blk.AttributeCollection.Count > 0)
    {
        using (var tran = blk.Database.TransactionManager.StartTransaction())
        {
            foreach (ObjectId id in blk.AttributeCollection)
            {
                var att = (AttributeReference)tran.GetObject(
                    id, OpenMode.ForRead);
                if (att.ExtensionDictionary.IsNull) continue;

                var extDict = (DBDictionary)tran.GetObject(att.ExtensionDiction
ary, OpenMode.ForRead);
                if (extDict.Contains(MyOverruleTypes.AttMoveGripOverrule.ToStri
ng()))
                {
                    var attGrip = new AttGrip()
                    {
                        GripPoint = att.Position,
                        AttributeId = id,
                        ClickAction = GenericHelper.MoveAttribute
                    };
```

```
                    grips.Add(attGrip);
                }
            }

            tran.Commit();
        }
    }

    base.GetGripPoints(
        entity, grips, curViewUnitSize, gripSize, curViewDir, bitFlags);
}
```

The highlighted code shows how a static method *MoveAttribute()* from a utility class *GenericHelper* is injected into the custom grip *AttGrip* when it is created. The code looks like:

```
        public static void MoveAttribute(ObjectId attId)
        {
            var ed = CadApp.DocumentManager.MdiActiveDocument.Editor;

            using (var tran = attId.Database.TransactionManager.StartTransactio
n())
            {
                var att = (AttributeReference)tran.GetObject(attId, OpenMode.Fo
rRead);
                att.Highlight();
                Point3d basePt = att.Position;
                var opt = new PromptPointOptions("\nPick new position for selec
ted attribute");
                opt.UseBasePoint = true;
                opt.BasePoint = basePt;
                opt.UseDashedLine = true;

                var res = ed.GetPoint(opt);
                att.Unhighlight();

                if (res.Status == PromptStatus.OK)
                {
                    att.UpgradeOpen();
                    att.TransformBy(Matrix3d.Displacement(basePt.GetVectorTo(re
s.Value)));
                }

                tran.Commit();
            }
        }
```

Obviously, we can use the same custom AttGrip class to perform other action against the Attribute entity, such as change its text height, its color…as long as the injected method takes an *ObjectId* (*AttributeReference.ObjectId*) parameter.

Create Custom Grip with Context Menu

If there are multiple actions, or an action can be performed differently with different options, it is obvious we can add multiple custom grips to the entities via our custom GripOverrule. However, it may not be desirable to have an entity to show many grips crowded together. All CAD users know that for some type of entity, when the mouse cursor hovers a built-in grip, a context menu may display; or when right-clicking a built-in grip, a context menu shows to provide a series of available operations against the selected entity.

There are different ways to let custom grip show context menu. One of them is to override *GridData* class's *OnRightClick()* method. Inside this method, we need to build our own context menu items that implement *Autodesk.AutoCAD.DatabaseServices.IMenuItem* interface. The other is to derive a custom class from *Autodesk.AutoCAD.DatabaseServices.MultiModeGripPE* class. With the former, it is completely up to us to find a suitable context menu components and implement the required *IMuneItem* interface. The real challenge with this approach is to show the context menu properly. With the latter approach, the base class *MultiModeGripPE* has already handled the context menu issue, we only need to focus on what data to show with each menu item and what action to be associated with each menu item. Here we focus on using custom *MultiModeGripPE* class with GripOverrule.

The key points of making a custom grip show context menu via using *MultiModeGripPE* are:

1. The custom *GripData* class:

- In the custom *GripData* class, add a member field of *GripModeCollection* class. This field is also exposed as read-only property, which can be accessed in custom *MultiModeGripPE* class;
- Add a public method to create multiple objects of *GripMode* class, each of them will shows in the context menu as menu item;
- Add member field of *GripMode.ModeIdentifier* enum type to indicate the current GripMode and it exposes as a property, which is accessed in custom *MultiModeGripPE* class;
- A public method that is called from the custom GripOverrule when one of the context menu item, which does the actual work against the selected entity. In this method, different operations perform according to the current GripMode. This method is optional, depending on what action type is associated to a GripMode. The commonly used action type is *GripMode.ActionType.*Immediate and *GripMode.ActionType.Command*. The public method is required if one of the action type is *GridMode.ActionType.Immediate*.

Following is a custom grip used with increase or reduce an Attribute's text height (thus 2 options in the context menu). This custom grip also presents another grip mode to toggle attribute's Invisible property:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.GraphicsInterface;
using Autodesk.AutoCAD.Runtime;
using CadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace GripOverruleSample.MultiAction
{
    public class AttMultiActionGrip : GripData
    {
        public const string USERDATA_KEY= "AttMultiActionGripData";
        public static ObjectId _commandTargetId = ObjectId.Null;
        private readonly GripModeCollection _gripModes;
        private const short GRIP_COLOR = 41;
        private GripMode.ModeIdentifier _currentModeId = GripMode.ModeIdentifie
r.CustomStart;

        public AttMultiActionGrip()
        {
            ForcedPickOn = true;
            GizmosEnabled = false;
            DrawAtDragImageGripPoint = false;
            IsPerViewport = false;
            ModeKeywordsDisabled = true;
            RubberBandLineDisabled = true;
            TriggerGrip = true;
            HotGripInvokesRightClick = true;
            HotGripInvokesRightClick = false;

            _gripModes = new GripModeCollection();
        }

        public virtual GripMode.ModeIdentifier CurrentModeId
        {
            get => _currentModeId;
            set => _currentModeId = value;
        }

        public static ObjectId CommandTargetId => _commandTargetId;

        public GripModeCollection GripModes => _gripModes;
        public ObjectId AttributeId { set; get; } = ObjectId.Null;
        public Action<ObjectId, bool> AttHeightAction { set; get; } = null;

        public override bool ViewportDraw(
            ViewportDraw worldDraw,
            ObjectId entityId,
            DrawType type,
```

```
            Point3d? imageGripPoint,
            int gripSizeInPixels)
        {
            var unit = worldDraw.Viewport.GetNumPixelsInUnitSquare(GripPoint);
            var gripHeight = 3.0 * gripSizeInPixels / unit.X;
            var points = new Point3dCollection();
            var x = GripPoint.X;
            var y = GripPoint.Y;
            var offset = gripHeight / 2.0;

            points.Add(new Point3d(x - offset, y, 0.0));
            points.Add(new Point3d(x, y - offset, 0.0));
            points.Add(new Point3d(x + offset, y, 0.0));
            points.Add(new Point3d(x, y + offset, 0.0));

            Point3d center = new Point3d(x, y, 0.0);
            var radius = offset;

            worldDraw.SubEntityTraits.FillType = FillType.FillAlways;
            worldDraw.SubEntityTraits.Color = GRIP_COLOR;
            worldDraw.Geometry.Circle(center, radius, Vector3d.ZAxis);

            return true;
        }

        public override ReturnValue OnHotGrip(ObjectId entityId, Context contex
tFlags)
        {
            return ReturnValue.Ok;
        }

        public bool GetGripModes(ref GripModeCollection modes, ref uint curMode
)
        {
            var gripMode = new GripMode()
            {
                ModeId = 0,
                DisplayString = "Increase Height",
                Action = GripMode.ActionType.Immediate,
                ToolTip = "Increase attribute height"
            };
            modes.Add(gripMode);

            gripMode = new GripMode()
            {
                ModeId = 1,
                DisplayString = "Reduce Height",
                Action = GripMode.ActionType.Immediate,
                ToolTip = "Reduce attribute height"
            };
            modes.Add(gripMode);
```

```
            gripMode = new GripMode()
            {
                ModeId = 2,
                DisplayString = "Toggle Visibility",
                Action = GripMode.ActionType.Command,
                CommandString = "ToggleAttributeVisible ",
                ToolTip = "Show/hide attribute"
            };
            modes.Add(gripMode);
            _commandTargetId = AttributeId;

            return true;
        }

        public void ChangeAttribute(ObjectId attId)
        {
            if (AttHeightAction == null) return;

            switch((int)CurrentModeId)
            {
                case 0:
                    AttHeightAction(AttributeId, true);
                    break;
                case 1:
                    AttHeightAction(AttributeId, false);
                    break;
            }
        }
    }
}
```

2. The custom *MultiModeGripPE* class:

- Derive a custom *MultiModeGripPE* class. Since it is only used inside the custom GripOverrule, it can be a private class inside the custom GripOverrule class;
- Override the base MultiModeGripPE class' virtual methods/properties.

The sample code as below:

```
        private class OverruledBlock : MultiModesGripPE
        {
            public override GripMode CurrentMode(Entity entity, GripData gripDa
ta)
            {
                var grip = gripData as AttMultiActionGrip;
                if (grip == null) return null;
                var index = (int)grip.CurrentModeId - (int)GripMode.ModeIdentif
ier.CustomStart;
                return grip.GripModes[index];
            }
```

```
            public override uint CurrentModeId(Entity entity, GripData gripData
)
            {
                var grip = gripData as AttMultiActionGrip;
                if (grip != null) return (uint)grip.CurrentModeId;
                return 0;
            }

            public override bool GetGripModes(
                Entity entity, GripData gripData, GripModeCollection modes, ref
 uint curMode)
            {
                if (!(gripData is AttMultiActionGrip)) return false;
                return ((AttMultiActionGrip)gripData).GetGripModes(ref modes, r
ef curMode);
            }

            public override GripType GetGripType(Entity entity, GripData gripDa
ta)
            {
                return (gripData is AttMultiActionGrip) ? GripType.Secondary :
GripType.Primary;
            }

            public override bool SetCurrentMode(Entity entity, GripData gripDat
a, uint curMode)
            {
                if (!(gripData is AttMultiActionGrip)) return false;
                ((AttMultiActionGrip)gripData).CurrentModeId = (GripMode.ModeId
entifier)curMode;
                return true;
            }

            public override void Reset(Entity entity)
            {
            }
        }
```

3. The custom *GripOverrule* class

- In the custom GripOverrule class' constructor, create an instance of the custom *MultiModeGripPE* class;
- Call *Overrule.GetClass(typeof([overruled entity]).AddX();*
- In the custom GripOverrule class' overridden *MoveGripPointsAt()* method, loop through the *GridDataCollection* to find our custom grip. If the custom grip has a public method exposed, as aforementioned, call it, which will perform the action against the selected entity.

The sample code looks like:

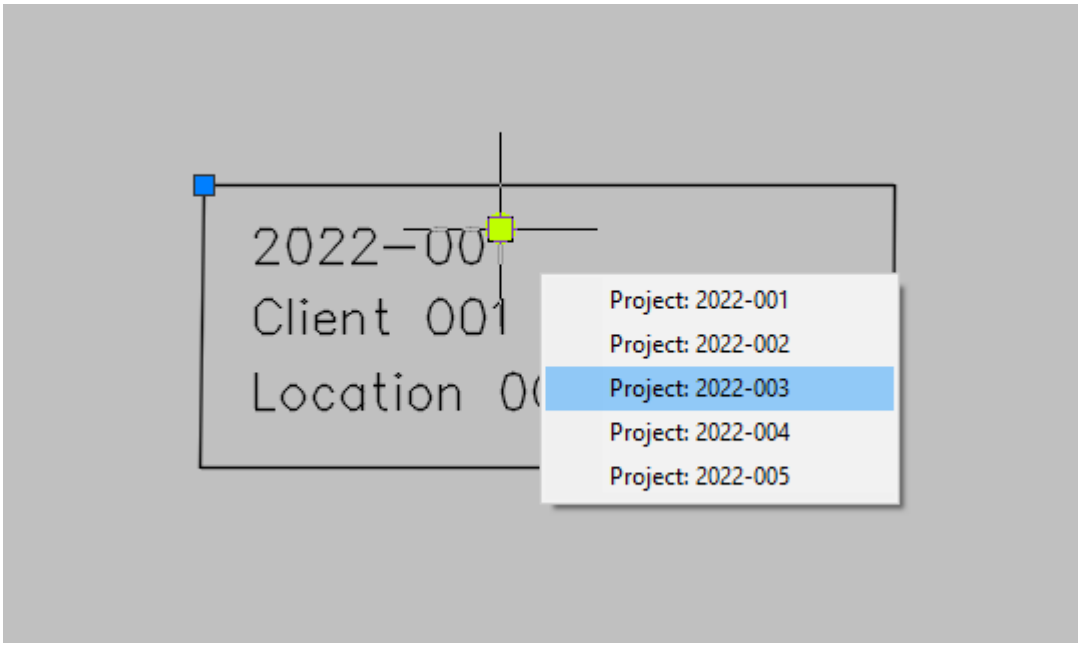
```
    // the custom GripOverrule's contructor
        public AttMultiActionGripOverrule() : base(…)
        {
            … …
            var overruled = new OverruledBlock();
            Overrule.GetClass(typeof(BlockReference)).AddX(
                GetClass(typeof(OverruledBlock)), overruled);
        }

    // the overridden method that triggers action defined in custom grip
        public override void MoveGripPointsAt(
            Entity entity, GripDataCollection grips, Vector3d offset, MoveGripP
ointsFlags bitFlags)
        {
            var ed = CadApp.DocumentManager.MdiActiveDocument.Editor;
            foreach (var grip in grips)
            {
                var attGrip = grip as AttMultiActionGrip;
                if (attGrip != null)
                {
                    attGrip.ChangeAttribute(attGrip.AttributeId);
                }
                else
                {
                    base.MoveGripPointsAt(entity, grips, offset, bitFlags);
                }
            }

        }
```

Use Data with Custom Grip

We now know custom grips basically provide CAD user a visualized command trigger/shortcut to start certain operation against selected entity. Because of its programmability, it is possible to make it data-driven.

Imagine this scenario: a drawing has multiple block references of a block, which has a few attributes which are to be updated with a set of data from a list of data records from external data source. The usual way to update the attributes with external data is to identify a target block reference (user selects it, for example), and run a command to get the attribute updated. During the command, user would be presented a list of the data record and has to select one record.

However, it is possible to use this set of data records to drive a custom grip in the target block reference to give user an easy and straightforward attribute update. For example, if the data list is not too long, we can use the data list to dynamically create the custom grip's context menu. So, user simply click a context menu item of the custom grip to get the block's attribute updated (as shown in picture below).



Here is the custom grip class *BlockDataGrip*:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.GraphicsInterface;
using CadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace GripOverruleSample.BlockData
{
    public class BlockDataGrip : GripData
    {
        private readonly GripModeCollection _gripModes;
        private const short GRIP_COLOR = 60;
        private GripMode.ModeIdentifier _currentModeId = GripMode.ModeIdentifie
r.CustomStart;

        public BlockDataGrip()
        {
            ForcedPickOn = true;
            GizmosEnabled = false;
            DrawAtDragImageGripPoint = false;
            IsPerViewport = false;
            ModeKeywordsDisabled = true;
            RubberBandLineDisabled = true;
```

```
            TriggerGrip = true;
            HotGripInvokesRightClick = true;
            HotGripInvokesRightClick = false;

            _gripModes = new GripModeCollection();
        }

        public ObjectId BlockId { set; get; } = ObjectId.Null;
        public List<(string project, string location, string client)> ProjectDa
ta { set; get; } = null;
        public GripModeCollection GripModes => _gripModes;
        public virtual GripMode.ModeIdentifier CurrentModeId
        {
            get => _currentModeId;
            set => _currentModeId = value;
        }

        public override bool ViewportDraw(
            ViewportDraw worldDraw,
            ObjectId entityId,
            DrawType type,
            Point3d? imageGripPoint,
            int gripSizeInPixels)
        {
            var unit = worldDraw.Viewport.GetNumPixelsInUnitSquare(GripPoint);
            var gripHeight = 3.0 * gripSizeInPixels / unit.X;
            var points = new Point3dCollection();
            var x = GripPoint.X;
            var y = GripPoint.Y;
            var offset = gripHeight / 2.0;

            points.Add(new Point3d(x - offset, y, 0.0));
            points.Add(new Point3d(x, y - offset, 0.0));
            points.Add(new Point3d(x + offset, y, 0.0));
            points.Add(new Point3d(x, y + offset, 0.0));

            Point3d center = new Point3d(x, y, 0.0);
            var radius = offset;

            worldDraw.SubEntityTraits.FillType = FillType.FillAlways;
            worldDraw.SubEntityTraits.Color = GRIP_COLOR;
            worldDraw.Geometry.Circle(center, radius, Vector3d.ZAxis);

            return true;
        }

        public override ReturnValue OnHotGrip(ObjectId entityId, Context contex
tFlags)
        {
            return base.OnHotGrip(entityId, contextFlags);
        }
```

```
        public bool GetGripModes(ref GripModeCollection modes, ref uint curMode
)
        {
            if (ProjectData!=null)
            {
                uint modeId = 0;
                foreach (var project in ProjectData)
                {
                    var gripMode = new GripMode()
                    {
                        ModeId = modeId,
                        DisplayString = $"Project: {project.project}",
                        Action = GripMode.ActionType.Immediate,
                        ToolTip = $"Project name: {project.project}"
                    };
                    modes.Add(gripMode);
                    modeId++;
                }
            }
            return true;
        }

        public void UpdateAttributes()
        {
            var projData = ProjectData[(int)CurrentModeId];
            using (var tran = BlockId.Database.TransactionManager.StartTransact
ion())
            {
                var blk = (BlockReference)tran.GetObject(BlockId, OpenMode.ForR
ead);
                foreach (ObjectId id in blk.AttributeCollection)
                {
                    var att = (AttributeReference)tran.GetObject(id, OpenMode.F
orWrite);
                    if (att.Tag.ToUpper() == "PROJECT") att.TextString = projDa
ta.project;
                    if (att.Tag.ToUpper() == "LOCATION") att.TextString = projD
ata.location;
                    if (att.Tag.ToUpper() == "CLIENT") att.TextString = projDat
a.client;
                }
                tran.Commit();
            }
        }
    }
}
```

Notice these points:

- The class has a public property “ProjectData”, which will be filled when the custom grip is created in the custom grip overrule class *BlockDataGripOverrule*;

- The method *GetGripMode(),* which is called in the custom *MultiModeGripPE* class inside the custom *BlockDataGripOverrule*, uses the supplied ProjectData to create corresponding GripMode (e.g. the context menu item)
- The method *UpdateAttributes()* is called when a context menu item is clicked, and the data record in the ProjectData is then identified by the CurrentModeId (e.g. which context menu item is clicked) and used to update the attributes in the block.

Since the custom grip is created inside the custom grip overrule, in order to pass external data to the custom grip, the custom grip overrule must has access to the external data.

Here is the custom grip overrule class *BlockDataGripOverrule:*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;

namespace GripOverruleSample.BlockData
{
    public class BlockDataGripOverrule : MyGripOverruleBase
    {
        private static BlockDataGripOverrule _instance = null;
        public BlockDataGripOverrule() :
            base(
                MyOverruleTypes.AttMoveGripOverrule,
                new[] { RXClass.GetClass(typeof(BlockReference)) },
                BlockDataGripOverrule.IsTargetBlock)
        {
            var overruled = new OverruledBlock();
            Overrule.GetClass(typeof(BlockReference)).AddX(GetClass(typeof(Over
ruledBlock)), overruled);
        }

        public static BlockDataGripOverrule Instance
        {
            get
            {
                if (_instance == null)
                {
                    _instance = new BlockDataGripOverrule();
                }
                return _instance;
            }
        }
```

```
        public List<(string project, string location, string client)> ProjectDa
ta
            { set; get; } = null;

        private class OverruledBlock : MultiModesGripPE
        {
            public override GripMode CurrentMode(Entity entity, GripData gripDa
ta)
            {
                var grip = gripData as BlockDataGrip;
                if (grip == null) return null;
                var index = (int)grip.CurrentModeId - (int)GripMode.ModeIdentif
ier.CustomStart;
                return grip.GripModes[index];
            }

            public override uint CurrentModeId(Entity entity, GripData gripData
)
            {
                var grip = gripData as BlockDataGrip;
                if (grip != null) return (uint)grip.CurrentModeId;
                return 0;
            }

            public override bool GetGripModes(
                Entity entity, GripData gripData, GripModeCollection modes, ref
 uint curMode)
            {
                if (!(gripData is BlockDataGrip)) return false;
                return ((BlockDataGrip)gripData).GetGripModes(ref modes, ref cu
rMode);
            }

            public override GripType GetGripType(Entity entity, GripData gripDa
ta)
            {
                return (gripData is BlockDataGrip) ? GripType.Secondary : GripT
ype.Primary;
            }

            public override bool SetCurrentMode(Entity entity, GripData gripDat
a, uint curMode)
            {
                if (!(gripData is BlockDataGrip)) return false;
                ((BlockDataGrip)gripData).CurrentModeId = (GripMode.ModeIdentif
ier)curMode;
                return true;
            }

            public override void Reset(Entity entity)
            {
            }
```

```
        }

        private static bool IsTargetBlock(Entity ent)
        {
            var blk = ent as BlockReference;
            if (blk != null && blk.AttributeCollection.Count > 0)
            {
                using (var tran = blk.Database.TransactionManager.StartTransact
ion())
                {
                    foreach (ObjectId id in blk.AttributeCollection)
                    {
                        var att = (AttributeReference)tran.GetObject(id, OpenMo
de.ForRead);
                        if (att.Tag.ToUpper() == "GRIPUPDATE" &&
                            att.Invisible &&
                            att.TextString.ToUpper()=="YES")
                        {
                            return true;
                        }
                    }
                    tran.Commit();
                }
            }
            return false;
        }

        public override void GetGripPoints(
            Entity entity,
            GripDataCollection grips,
            double curViewUnitSize,
            int gripSize,
            Vector3d curViewDir,
            GetGripPointsFlags bitFlags)
        {
            var blk = entity as BlockReference;
            if (blk != null && blk.AttributeCollection.Count > 0)
            {
                using (var tran = blk.Database.TransactionManager.StartTransact
ion())
                {
                    foreach (ObjectId id in blk.AttributeCollection)
                    {
                        var att = (AttributeReference)tran.GetObject(
                            id, OpenMode.ForRead);

                        if (att.Tag.ToUpper() != "PROJECT") continue;

                        Func<AttributeReference, Point3d?> GetAttributeCenter =
 (attRef) =>
                        {
                            if (!string.IsNullOrEmpty(attRef.TextString))
```

```
                            {
                                var ext = attRef.GeometricExtents;
                                return ext.MaxPoint;
                            }
                            else
                            {
                                return null;
                            }
                        };

                        var pt = GetAttributeCenter(att);
                        var attGrip = new BlockDataGrip()
                        {
                            GripPoint = pt.HasValue ? pt.Value : att.Position,
                            ProjectData = ProjectData,
                            BlockId = blk.ObjectId
                        };

                        grips.Add(attGrip);
                    }

                    tran.Commit();
                }
            }

            base.GetGripPoints(
                entity, grips, curViewUnitSize, gripSize, curViewDir, bitFlags)
;
        }

        public override void MoveGripPointsAt(
            Entity entity,
            GripDataCollection grips,
            Vector3d offset,
            MoveGripPointsFlags bitFlags)
        {
            var ed = Application.DocumentManager.MdiActiveDocument.Editor;
            foreach (var grip in grips)
            {
                var attGrip = grip as BlockDataGrip;
                if (attGrip != null)
                {
                    attGrip.UpdateAttributes();
                }
                else
                {
                    base.MoveGripPointsAt(entity, grips, offset, bitFlags);
                }
            }

        }
    }
```

```
}
```

This class has a public property “ProjectData”, which should be filled when this custom grip overrule is created. Then in the overridden method *GetGripPoints()* the ProjectData is passed to a newly created *BlockDataGrip* instance

Because we need to supply the external data to the custom grip overrule, the code of a CommandMethod, where the grip overrule is enabled, looks like:

```
[CommandMethod("BlkDataGripOr")]
        public static void EnableBlockDataGrip()
        {
            var dwg = CadApp.DocumentManager.MdiActiveDocument;
            var ed = dwg.Editor;

            if (!_blkDataEnabled)
            {
                // supply data to the grip overrule
                var projectData = GetProjectData();
                BlockData.BlockDataGripOverrule.Instance.ProjectData = projectD
ata;

                BlockData.BlockDataGripOverrule.Instance.EnableOverrule(true);
                _blkDataEnabled = true;
                ed.WriteMessage("\nAttributeMoveGripOeverrule is enabled.\n");
            }
            else
            {
                BlockData.BlockDataGripOverrule.Instance.EnableOverrule(false);
                _blkDataEnabled = false;
                ed.WriteMessage("\nAttributeMoveGripOeverrule is disabled.\n");
            }
        }

        private static List<(string project, string location, string client)> G
etProjectData()
        {
            // project data usually is obtained from external data source,
            // such as database, data files... Here we use the hard-coded
            // for simplicity of the code
            var data = new List<(string, string, string)>();
            for (int i=1; i<=5; i++)
            {
                var p = $"2022-{i.ToString().PadLeft(3, '0')}";
                var l = $"Location {i.ToString().PadLeft(3, '0')}";
                var c = $"Client {i.ToString().PadLeft(3, '0')}";
                data.Add((p, l, c));
            }
            return data;
        }
```

Now with BlockDataGripOverrule enabled, once the custom grip shows up with an applied block reference, hover the custom grip with mouse cursor will bring up a context menu with the data set record as menu item, as seen in the picture a few pages previous.

Sampe AutoCAD .NET API Project

A sample AutoCAD .NET API Plugin project in C# is available, which works with all later versions of AutoCAD (AutoCAD 2015 or later).