# Chapter 6: Working with Feature Service, Part II

## Updating, Deleting, and Creating Features



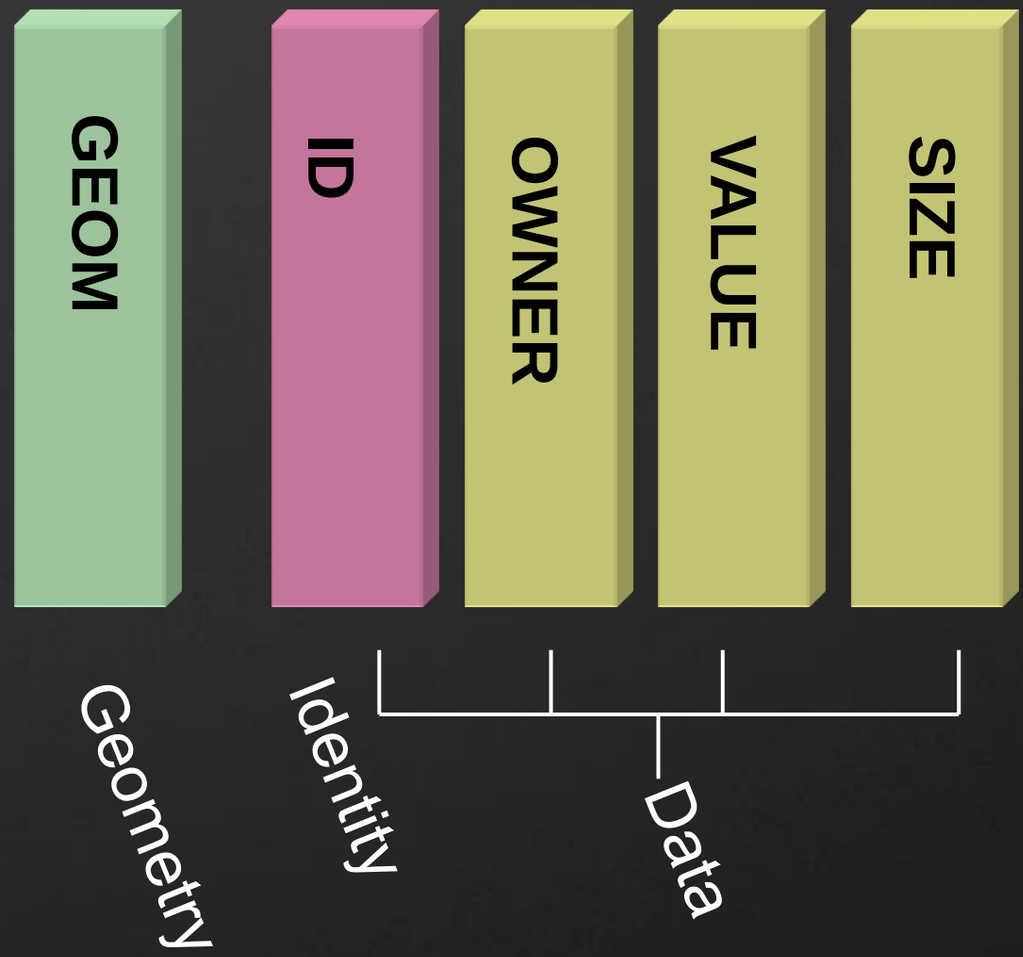
Autodesk

# Chapter Overview

- Structure of feature class
- Update feature class
  - Insert new features
  - Update features
  - Delete features
- Create feature class and feature layer
  - feature source
  - feature layer
- Create new geometry with mouse
  - Viewer API
  - Server API

# Feature Class Structure

- Database-table-like structure
- Feature class contains properties corresponding to table columns.
- **Property types**:
  - Geometry
  - Data
  - Raster
  - Object
- Identity properties are used to uniquely identify a feature in a feature class.



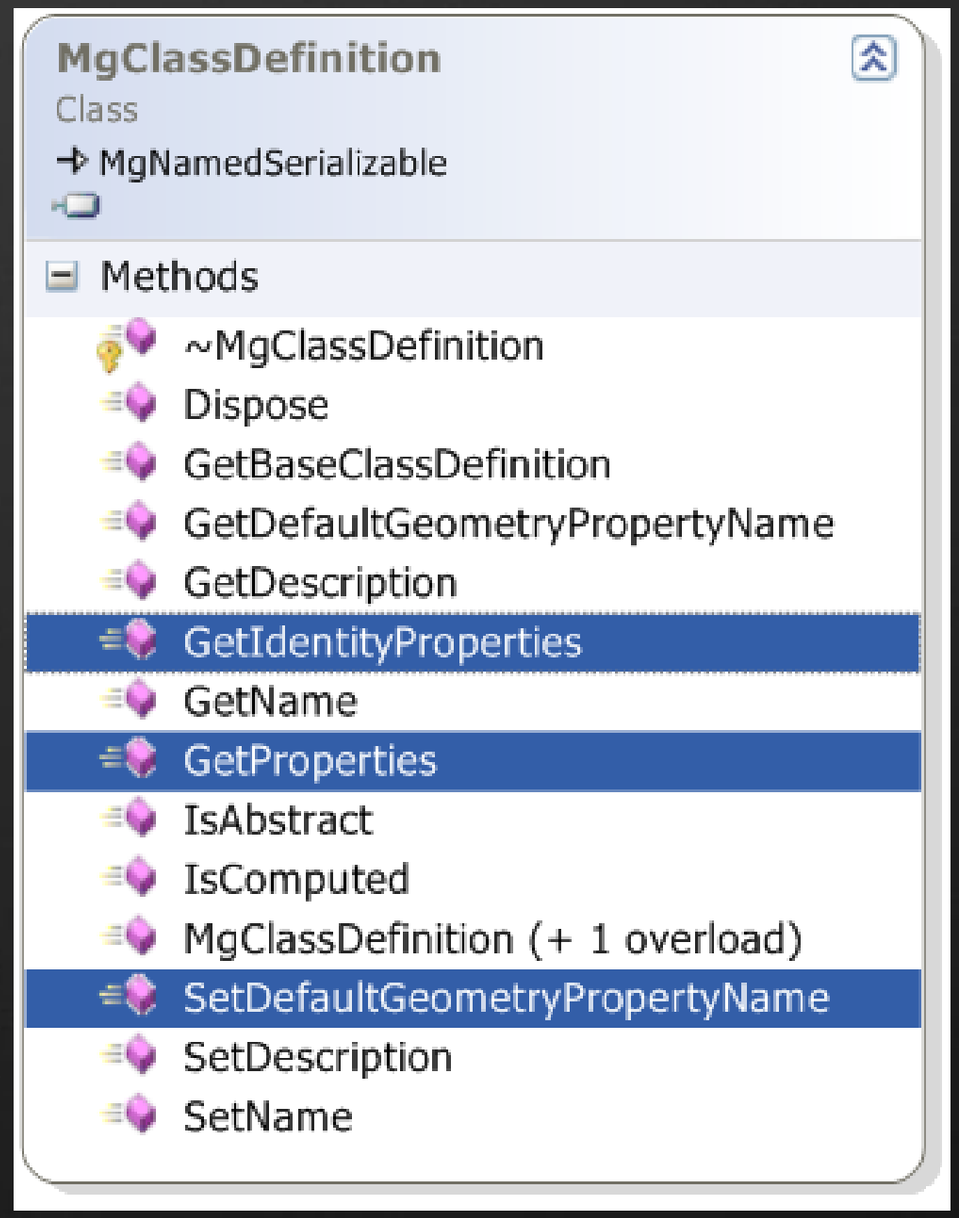
# MgClassDefinition Class

MgClassDefinition

- Represent a feature class
- Sets class properties
- Used to describe or create a new feature class

- **Important methods**
  - GetProperties()
  - GetIdentityProperties()

- The return type of these two methods is **MgPropertyDefinitionCollection**






Autodesk

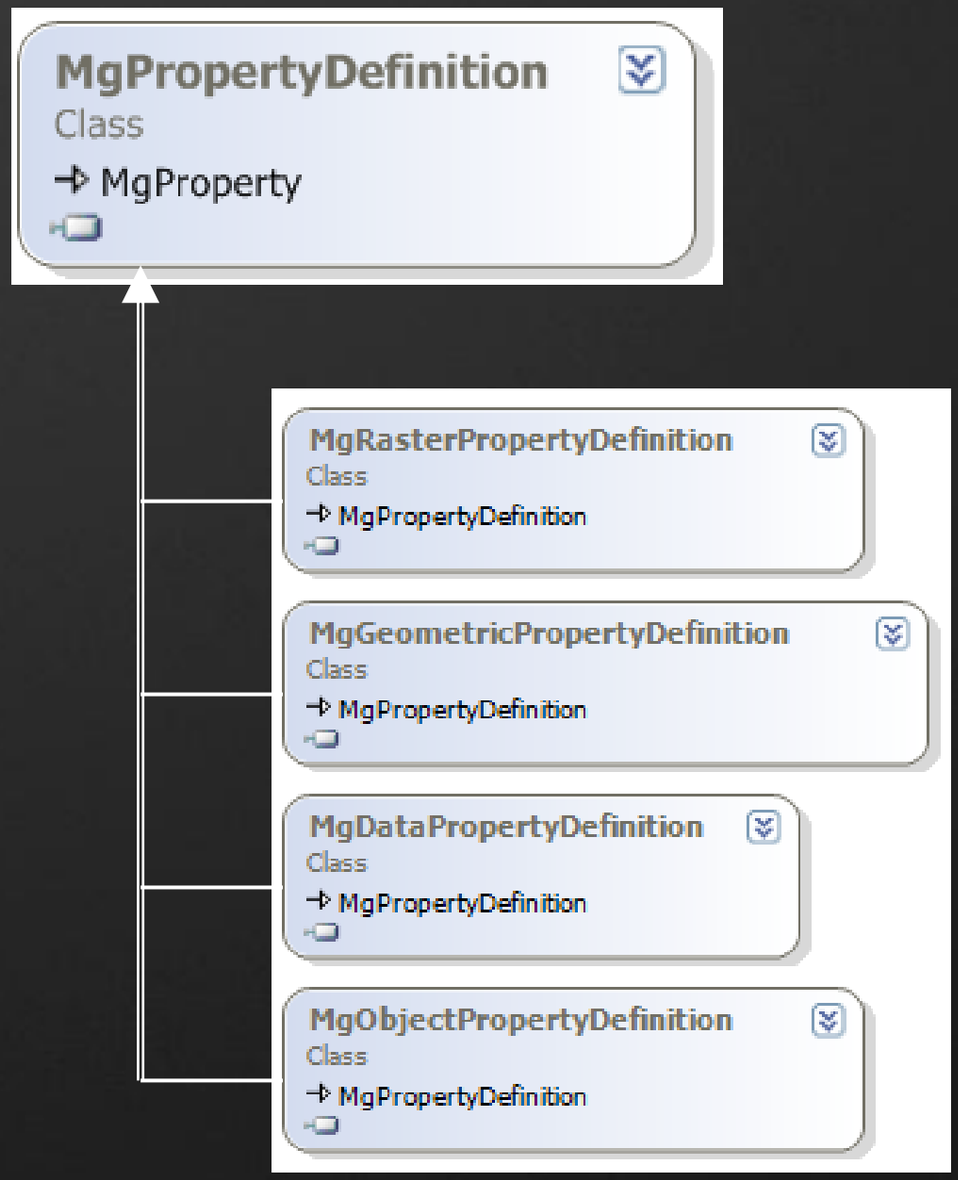
# MgPropertyDefinition Class

MgPropertyDefinitionCollection

- A collection of property definitions.

MgPropertyDefinition

- The details of a feature class property.
- Has 4 subclasses or types
  - MgDataPropertyDefinition
  - MgGeometric…
  - MgRaster…
  - MgObject…






Autodesk

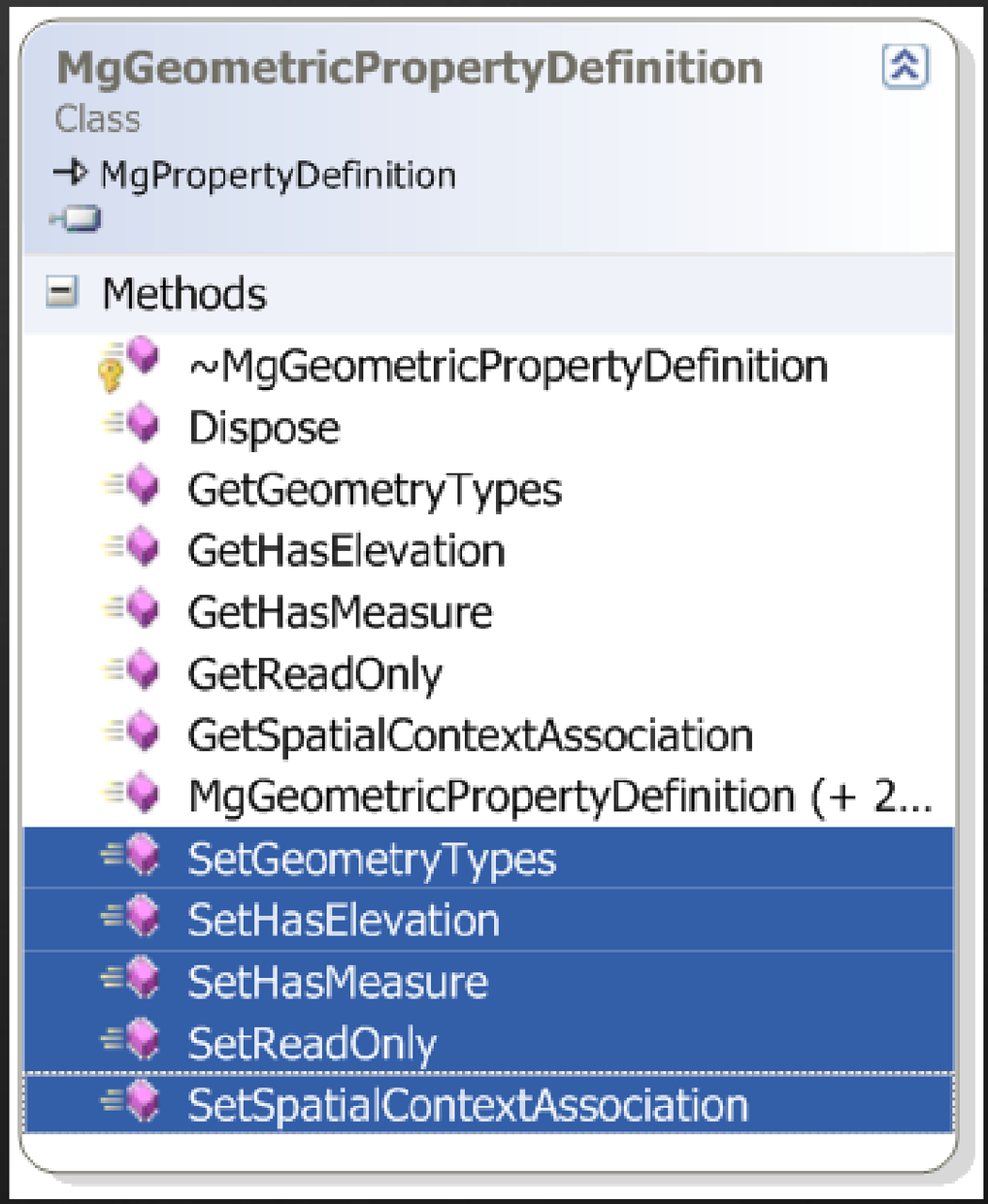
# MgGeometricPropertyDefinition Class

MgGeometricPropertyDefinition

- Defines a geometric property
- Important methods
  - SetHasElevation()
  - SetHasMeasure()
  - SetReadOnly()
  - SetSpatialContextAssociation
  - SetGeometryTypes()

Types of geometric properties

- MgFeatureGeometricType
  - Point = 1
  - Curve = 2
  - Surface = 4
  - Solid = 8






Autodesk

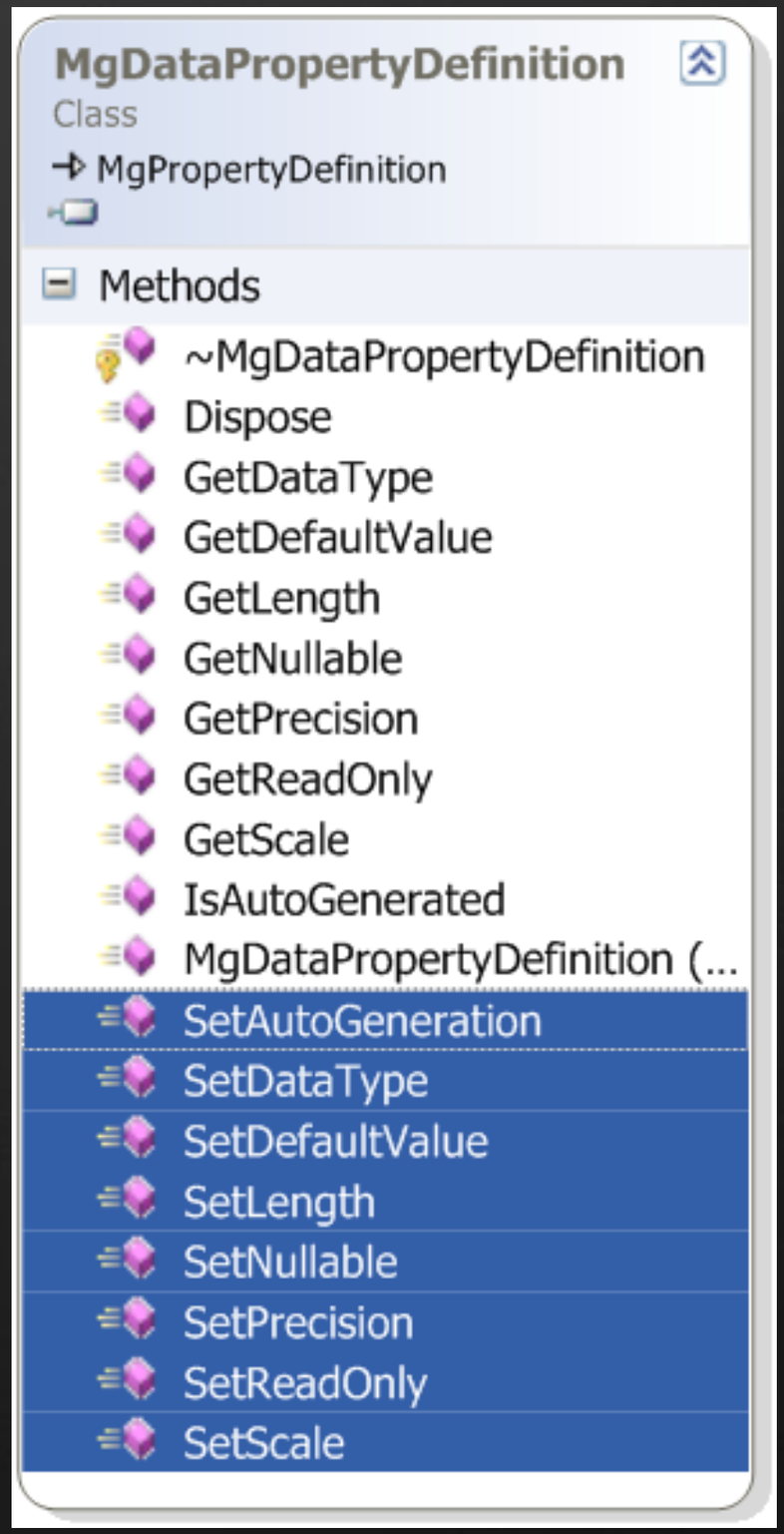
# MgDataPropertyDefinition Class

MgDataPropertyDefinition

- Defines a data property
- Important methods
  - SetAutoGeneration()
  - SetDefaultValue()
  - SetLength()
  - SetNullable()
  - SetPrecision()
  - SetReadOnly()
  - SetScale()

Types of data properties

- MgPropertyType
  - Double = 5
  - Int32 = 7
  - String = 9
  - …






Autodesk

# Create a New Feature Class

Step 1

- Create a new MgClassDefinition object
- Get the property definition collection from the class

```
string ll84Wkt = @"GEOGCS[\"LL84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS
84\",6378137,298.25722293287],TOWGS84[0,0,0,0,0,0,0]],PRIMEM[\"Greenwich
\",0],UNIT[\"Degrees\",1]]";

MgClassDefinition parcelClass = new MgClassDefinition();

parcelClass.SetName("tempParcel");

MgPropertyDefinitionCollection props =
    parcelClass.GetProperties();
```

# Create a New Feature Class

Step 2

- Populate the property collection with proper values

```
MgDataPropertyDefinition id = new MgDataPropertyDefinition("ID");
id.SetDataType(MgPropertyType.Int32);
id.SetReadOnly(true);
id.SetNullable(false);
id.SetAutoGeneration(true);
props.Add(id);

MgPropertyDefinitionCollection idProps = parcelClass.GetIdentityProperties();
idProps.Add(id);

MgGeometricPropertyDefinition geom = new MgGeometricPropertyDefinition("GEOM");
geom.SetGeometryTypes(MgFeatureGeometricType.Surface);
geom.SetHasElevation(false);
geom.SetHasMeasure(false);
geom.SetSpatialContextAssociation("LL84");
props.Add(geom);
parcelClass.SetDefaultGeometryPropertyName("GEOM");

MgDataPropertyDefinition acre = new MgDataPropertyDefinition("ACRE");
acre.SetDataType(MgPropertyType.String);
acre.SetLength(256);
props.Add(acre);
```

# Create a New Feature Class

Step 3

- Create and set up schema
- Create the physical storage format

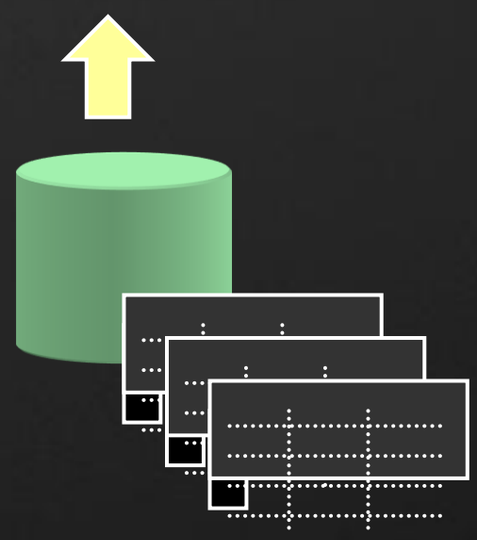
```
MgFeatureSchema schema = new MgFeatureSchema();
schema.SetName("SchemaParcels");
schema.GetClasses().Add(parcelClass);

MgCreateSdfParams sdfParams = new
    MgCreateSdfParams("LL84", ll84Wkt, schema);
featureService.CreateFeatureSource(resId, sdfParams);
```

# Map Layer

Map Layer

- Visual representation of feature class.
- Based on feature class.
- Composition and style are defined in LayerDefinition file.

- Like feature sources, map layers can be created on permanent library or temporary session repository.



- Can be dynamically created on-the-fly.

# Create a Map Layer Programmatically

- Create or parse the LayerDefinition XML file.
- Create a new map layer resource.
- Set the map layer's properties.

```
Byte[] layerDef = new byte[length - 1];
MgByteSource byteSource = new MgByteSource(layerDef,
    layerDef.Length);
byteSource.SetMimeType(MgMimeType.Xml);

MgResourceIdentifier parcelLayerId = new
MgResourceIdentifier("Session:" + sessionId +
"//parcelLayer.LayerDefinition");
resService.SetResource(parcelLayerId,
byteSource.GetReader(), null);

MgLayer parcelLayer = new MgLayer(
    tempParcelLayerId, resService);
parcelLayer.SetName("NewParcels");
parcelLayer.SetLegendLabel("New Parcels");
parcelLayer.SetDisplayInLegend(true);
parcelLayer.SetSelectable(false);
```

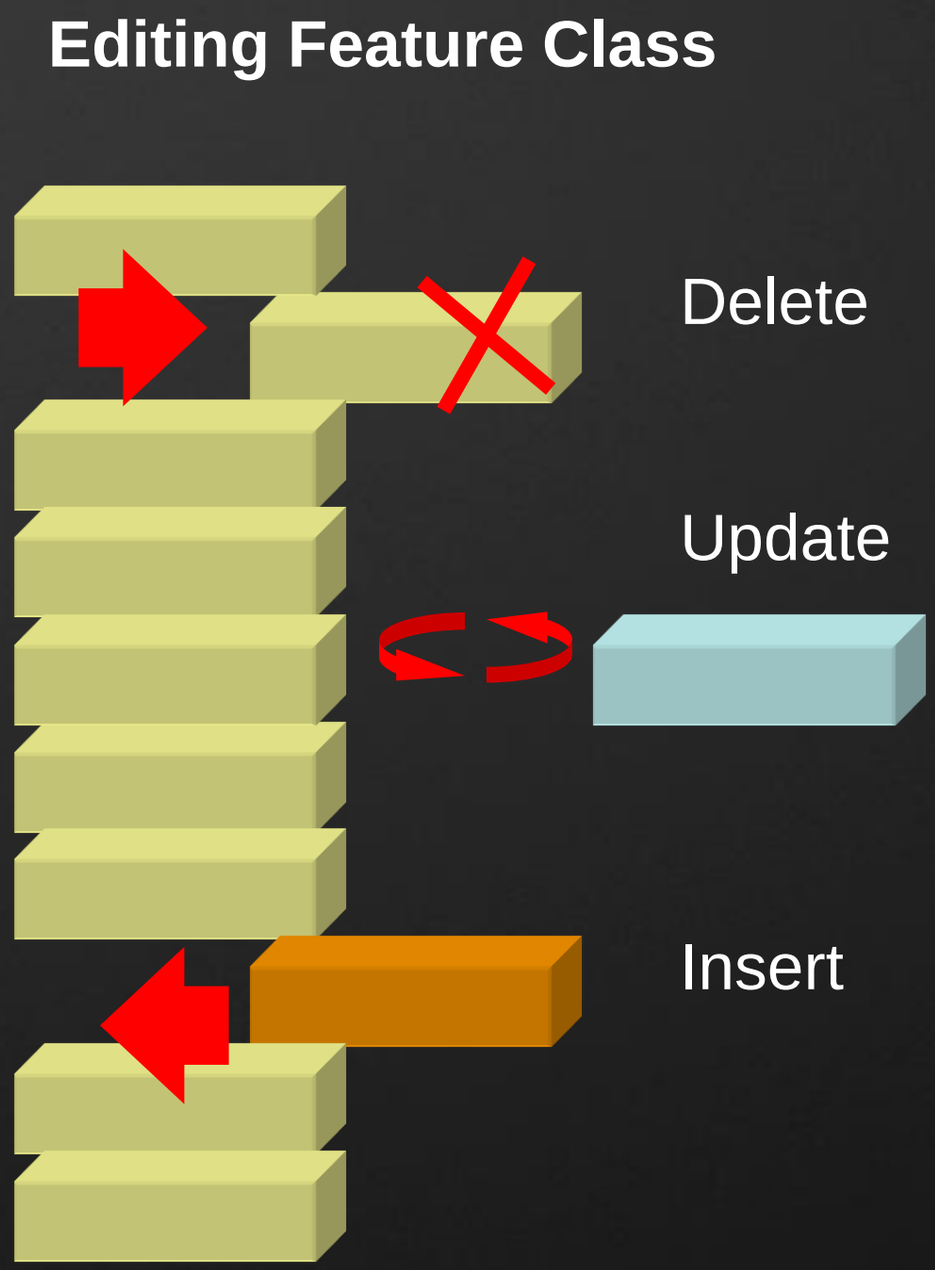
# Add Layer to Map Display

- Get the current map display, MgMap.
- Add the layer to map display.
- Refresh map display and save it.

```
MgMap map = new MgMap();
map.Open(resService, "Sheboygan");

MgLayer parcelLayer = getLayerByName(map, "NewParcels");
map.GetLayers().Insert(0, parcelLayer);

parcelLayer.SetVisible(true);
parcelLayer.ForceRefresh();
map.Save(resService);
```

# Editing Feature Class

- Feature class can be edited.
- **Three operations**
  - Delete
  - Update
  - Insert
- Feature editing is carried out on FDO data source






Autodesk

# Delete Features - MgDeleteFeatures

- Deleting features is performed by MgDeletFeatures.
- Pass in the query string and the name of the feature class on which the deletion will be carried out.

```
MgDeleteFeatures deleteFeatures = new MgDeleteFeatures
    ("Parcels", "ID=2354");
```

```
MgDeleteFeatures deleteFeatures = new MgDeleteFeatures
    ("Parcels", "OWNER LIKE 'JOHN%'");
```

```
MgDeleteFeatures deleteFeatures = new MgDeleteFeatures
    ("Parcels", "GEOM INTERSECTS GEOMFROMTEXT
    ('POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))')");
```

# Update Features - MgUpdateFeatures

- Updating features is performed by MgUpdateFeatures.
- Steps
  1. Create an MgPropertyCollection object, which holds the new property values for the features to be updated.
  2. Passing in the query string, MgPropertyCollection object, and name of the feature class on which the updating will be performed.

```
//Variable "properties" is the MgPropertyCollection object.
MgUpdateFeatures updateFeatures = new MgUpdateFeatures
    ("Parcels", properties, "ID=2354");
```

```
MgUpdateFeatures updateFeatures = new MgUpdateFeatures
    ("Parcels", properties, "OWNER LIKE 'JOHN%'");
```

```
MgUpdateFeatures updateFeatures = new MgUpdateFeatures
    ("Parcels", properties, "GEOM INTERSECTS GEOMFROMTEXT
    ('POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))')");
```

# MgPropertyCollection Class

MgPropertyCollection Class

- Holds the property values used to update a feature class.
- Must contains the values for the non-nullable properties.
- For example, to update a feature class with schema like the one on the right MgPropertyCollection object must contains these properties.
  - ID
  - GEOM
  - OWNER

# MgPropertyCollection Class

Example on using MgPropertyCollection class

- Assume all the properties are nullable and we need to update the feature with these values

```
MgPropertyCollection props = new MgPropertyCollection();
MgAgfReaderWriter agfWriter = new MgAgfReaderWriter();

props.Add(new MgGeometryProperty("GEOM", agfWriter.Write(geom)));
props.Add(new MgStringProperty("LOTDIM", "540X400"));
props.Add(new MgInt32Property("SQFT", "6600"));
props.Add(new MgStringProperty("ZONE", "RES" ));
```

# Insert Features, MgInsertFeatures

- Inserting features is performed by MgInsertFeatures class.
- Steps
  1. Create an MgPropertyCollection object, which hold the values of a new feature.
  2. Passing in the MgPropertyColllection object and the name of the feature class, which is to be inserted.

```
//Variable "properties" is the MgPropertyCollection object
MgInsertFeatures insertFeatures =
    new MgInsertFeatures ("Parcels", properties);
```

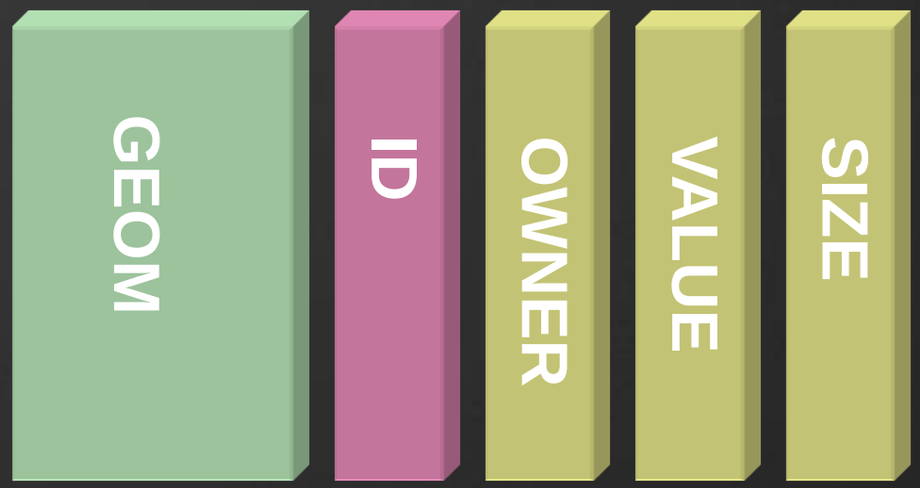
# Commit Edits to Feature Class

- Edits of deleting, updating, and inserting need to be committed to the feature class.
- It’s done by MgFeatureCommandCollection class.
- Feature commands are executed in the order they are added to the MgFeatureCommandCollection object.
- The execution can be a transaction for the supported FDO data providers.

```
MgFeatureCommandCollection commands = new
    MgFeatureCommandCollection();
commands.Add(deleteFeatures);
commands.Add(updateFeatures);
commands.Add(insertFeatures);
//true means the execution is a transaction
featureService.UpdateFeatures(sourceId, commands, true);
```

# Create Geometries

- Features contain geometries, which the geographic presentation of the real world entities.
- Geometries can be created dynamically during the execution of MapGuide applications.
- With the help of view API, geometries can be drawn with mouse in browser.






Autodesk

# MgGeometryFactory Class

MgGeometryFactory Class

- Constructs geometry objects such points, line strings (polyline), and polygons.
- Doesn’t examine geometric validity. You as the developer need to verify the legitimacy of the geometry.

```
MgGeometryFactory geoFactory = new MgGeometryFactory();
MgCoordinateCollection coordCol = new
    MgCoordinateCollection();

for (int i = 0; i < num; i++)
{
    MgCoordinate coord =
    geoFactory.CreateCoordinateXY(x[i], y[i]);
   coordCol.Add(coord);
 }

MgLinearRing outRing =
    geoFactory.CreateLinearRing(coordCol);
MgPolygon polygon =
    geoFactory.CreatePolygon(outRing, null);
```

# Create Polygon with View API

- Users can draw a geometry like polygon in browser with viewer API.
- The drawing can be submitted to MapGuide applications to construct an MgPolygon object.

```
<script type="text/javascript">
   function DigitizePolygon() {
      parent.parent.mapFrame.DigitizePolygon
         (OnPolygonDigitized);
   }
   function OnPolygonDigitized(poly) {
      FormatLineResultsForParser(poly);
   }
   function FormatLineResultsForParser(line) {
      str = line.Count + "~";
      for(var i = 0; i < line.Count; i++) {
        pt = line.Point(i);
            str += pt.X + "!" + pt.Y + "_";
      }
      document.getElementById("coordinates").value = str;
   }
</script>
```

# Questions

## Questions ?



Autodesk

# Exercise

- Create layer /temp layer
- Solution 6