AutoCAD® Map 3D 2013 Platform API Training

Autodesk

Feature Service

Autodesk

Contents

- Feature Service Overview
- FDO Registry and Providers
- Feature Schema and Feature Class
- Creating and Managing Features
- Query and Filtering

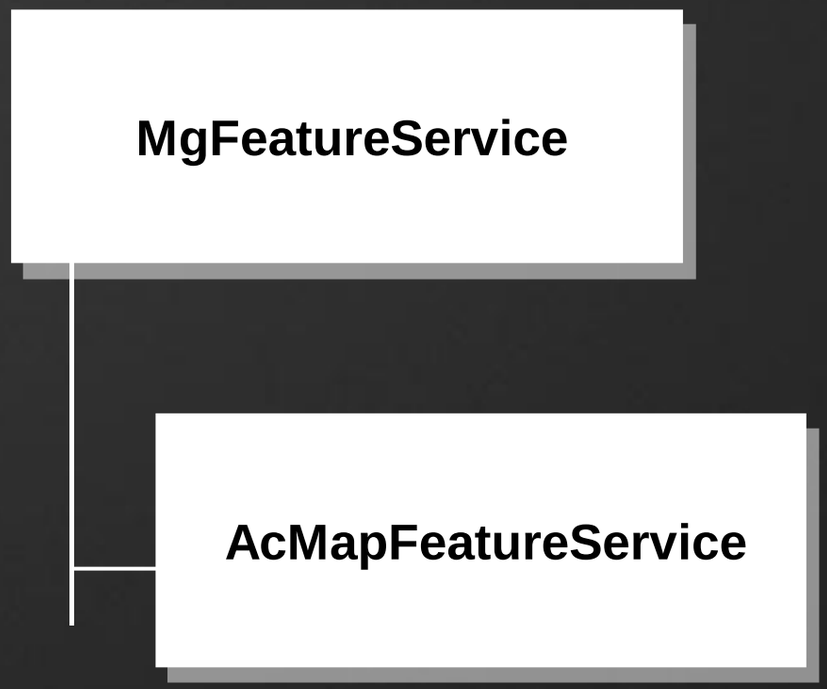
Autodesk

Feature Service Overview

- Feature service provides APIs to store and retrieve features
- Independent of the data storage technology by creating an abstraction layer.
- FDO (Feature Data Object) is used to construct this abstraction layer
- Key concepts
  - Feature
  - Feature source
  - Feature schema
  - Feature class

Creating the Feature Service

Feature service is created from **AcMapServiceFactory** object using the *GetService* method




```
// Get the feature service
AcMapFeatureService featureService =
AcMapServiceFactory GetService(ServiceType.FeatureService) as
AcMapFeatureService
```

FDO Registry

- Physically under this directory
  C:\Program Files\ Autodesk \AutoCAD Map 3D 2013\bin\FDO
  - Programmatically get the registry and list the providers with the GetFeatureProviders method

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <FeatureProviderRegistry>
 - <FeatureProvider>
    <Name>OSGeo.WMS.3.3</Name>
    <DisplayName>OSGeo FDO Provider for WMS</DisplayName>
    <Description>Read access to OGC WMS-based data store.</Description>
    <IsManaged>False</IsManaged>
    <Version>3.3.0.0</Version>
    <FeatureDataObjectsVersion>3.3.0.0</FeatureDataObjectsVersion>
    <LibraryPath>WMSProvider.dll</LibraryPath>
  </FeatureProvider>
 - <FeatureProvider>
    <Name>OSGeo.WFS.3.3</Name>
    <DisplayName>OSGeo FDO Provider for WFS</DisplayName>
    <Description>Read access to OGC WFS-based data store.</Description>
    <IsManaged>False</IsManaged>
    <Version>3.3.0.0</Version>
    <FeatureDataObjectsVersion>3.3.0.0</FeatureDataObjectsVersion>
    <LibraryPath>WFSProvider.dll</LibraryPath>
  </FeatureProvider>
- <FeatureProvider>
    <Name>OSGeo.SHP.3.3</Name>
    <DisplayName>OSGeo FDO Provider for SHP</DisplayName>
    <Description>Read/write access to spatial and attribute data in an ESRI SHP file.</Description>
    <IsManaged>False</IsManaged>
    <Version>3.3.0.0</Version>
    <FeatureDataObjectsVersion>3.3.0.0</FeatureDataObjectsVersion>
    <LibraryPath>ShpProvider.dll</LibraryPath>
  </FeatureProvider>
```

```
// Get the registered feature providers
MgByteReader reader =  featureService.GetFeatureProviders();
String providers = reader.ToString();
```


Autodesk

FDO Provider Capabilities

- Different FDO providers have different capabilities.
- For example
  - SHP and SDF don’t support topology.
  - Oracle and SQL Server support database transaction.
- Get a provider’s capability programmatically using the GetCapabilities method

FDO Provider Capabilities

FdoProviderCapabilities	
E Provider	(Provider)
E Connection	(Connection)
E Schema	(Schema)
E Command	(Command)
E Filter	(Filter)
E Expression	(Expression)
E Raster	(Raster)
E Topology	(Topology)
E Geometry	(Geometry)

```
// Get the capabilities of a provider
 MgByteReader reader = featureService.GetCapabilities(fullProviderName);
```

Feature Schema (MgFeatureSchema)

- Defines the structure of data in a feature source
- Contains one or more feature classes

```
// Get the XML representation of a feature schema
MgResourceIdentifier id = new MgResourceIdentifier("Library://Data/Raster/Redding.FeatureSource");
String schema = featureService.DescribeSchemaAsXml(id, “ReddingSchema");
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <fdo:DataStore xmlns:xs="http://www.w3.org/2001/X
  xmlns:gml="http://www.opengis.net/gml" xmlns:fdo=
  - <xs:schema xmlns:xs="http://www.w3.org/2001/XM
    xmlns:gml="http://www.opengis.net/gml" xmlns:ra
    - <xs:element name="Redding" type="rasters:Redding
      - <xs:key name="ReddingKey">
          <xs:selector xpath=".//Redding" />
          <xs:field xpath="Id" />
        </xs:key>
      </xs:element>
    - <xs:complexType name="ReddingType" abstract="fal
      - <xs:complexContent>
        - <xs:extension base="fdo:ClassType">
          - <xs:sequence>
            - <xs:element name="Id">
              - <xs:simpleType>
                - <xs:restriction base="xs:string">
                    <xs:maxLength value="256" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="Image" type="fdo:Raster
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
  </xs:schema>
```

Feature Schema

Getting existing schemas

```
// Get the names of all schemas in a feature source
MgResourceIdentifier id = new MgResourceIdentifier("Library://Data/Raster/Redding.FeatureSource");
MgStringCollection schemaNames = featureService.GetSchemas(id);
MgFeatureSchema schema = null;

// Get a particular schema
foreach( String schemaName in schemaNames)
{
   if(schemaName.ToLower() == “reddingschema")
   {
     schema = featureService.DescribeSchema(id, schemaName).GetItem(0);
     break;
   }
}
```

Creating a new schema

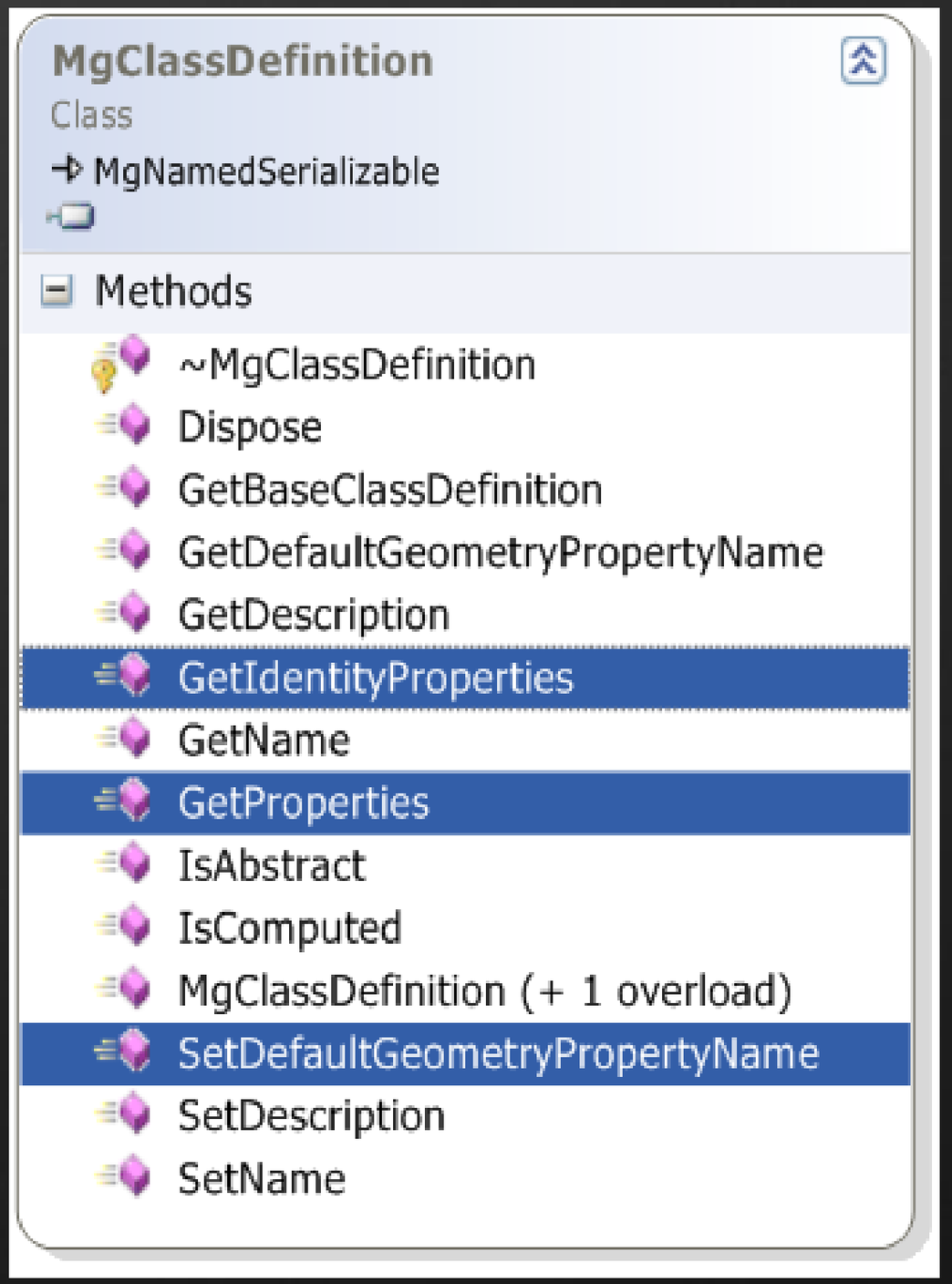
```
// Use the FDO API
FeatureSchema schema = new FeatureSchema(“SchemaName”,
“SchemaDescription")
```

Feature Class (MgClassDefinition)

- 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

- Sets class properties
- Used to describe or create a new feature class
- **Important methods**
  - SetDefaultGeometryPropertyName()
  - GetProperties()
    - Returns MgPropertyDefinitionCollection
  - GetIdentityProperties()
    - Returns MgPropertyDefinitionCollection

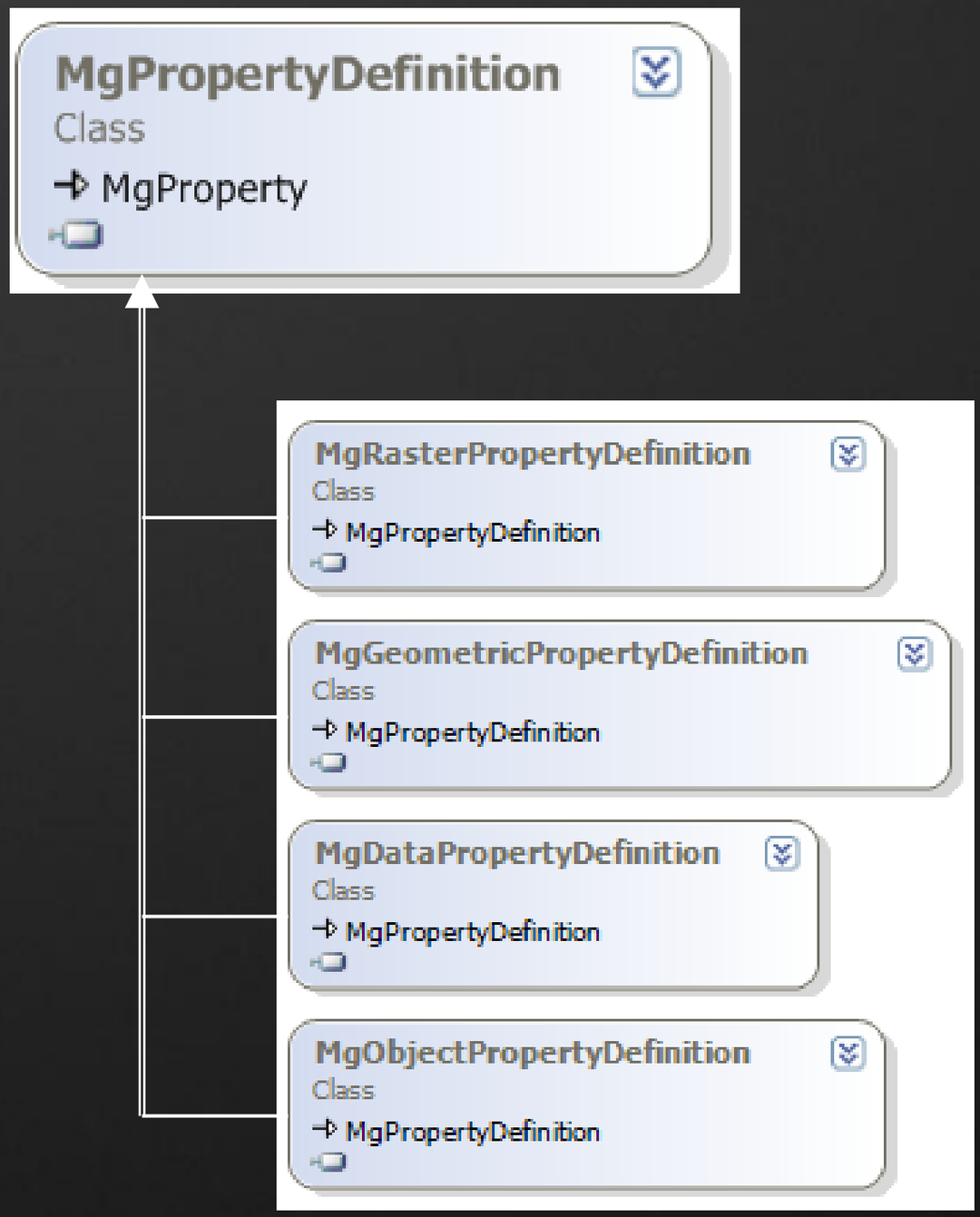


Autodesk

MgPropertyDefinition

MgPropertyDefinition

- The details of a feature class property.
- Has 4 subclasses or types
  - MgDataPropertyDefinition
  - MgGeometricPropertyDefinition
  - MgRasterPropertyDefinition
  - MgObjectPropertyDefinition



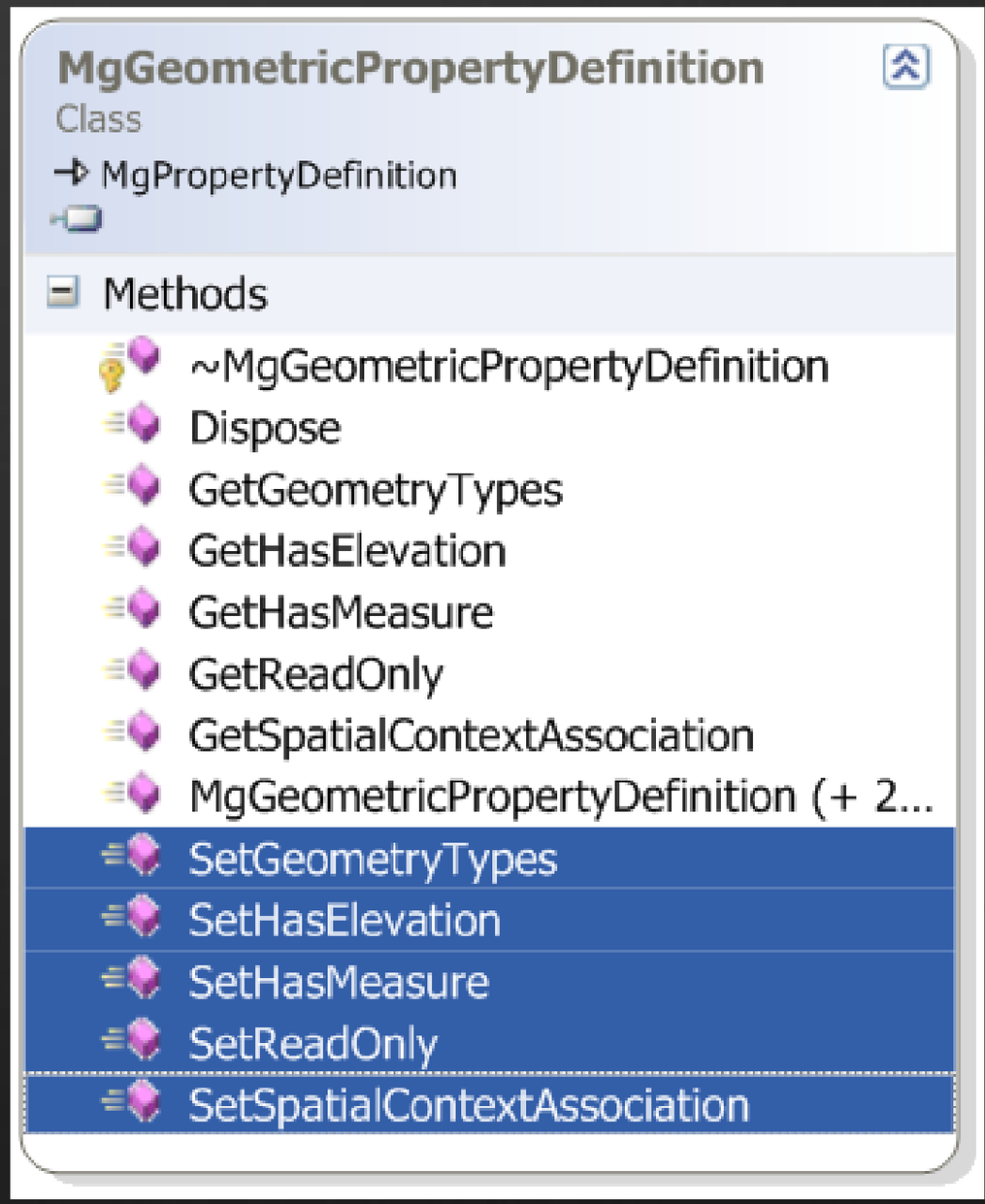
Autodesk

MgGeometricPropertyDefinition

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

Types of geometric properties

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



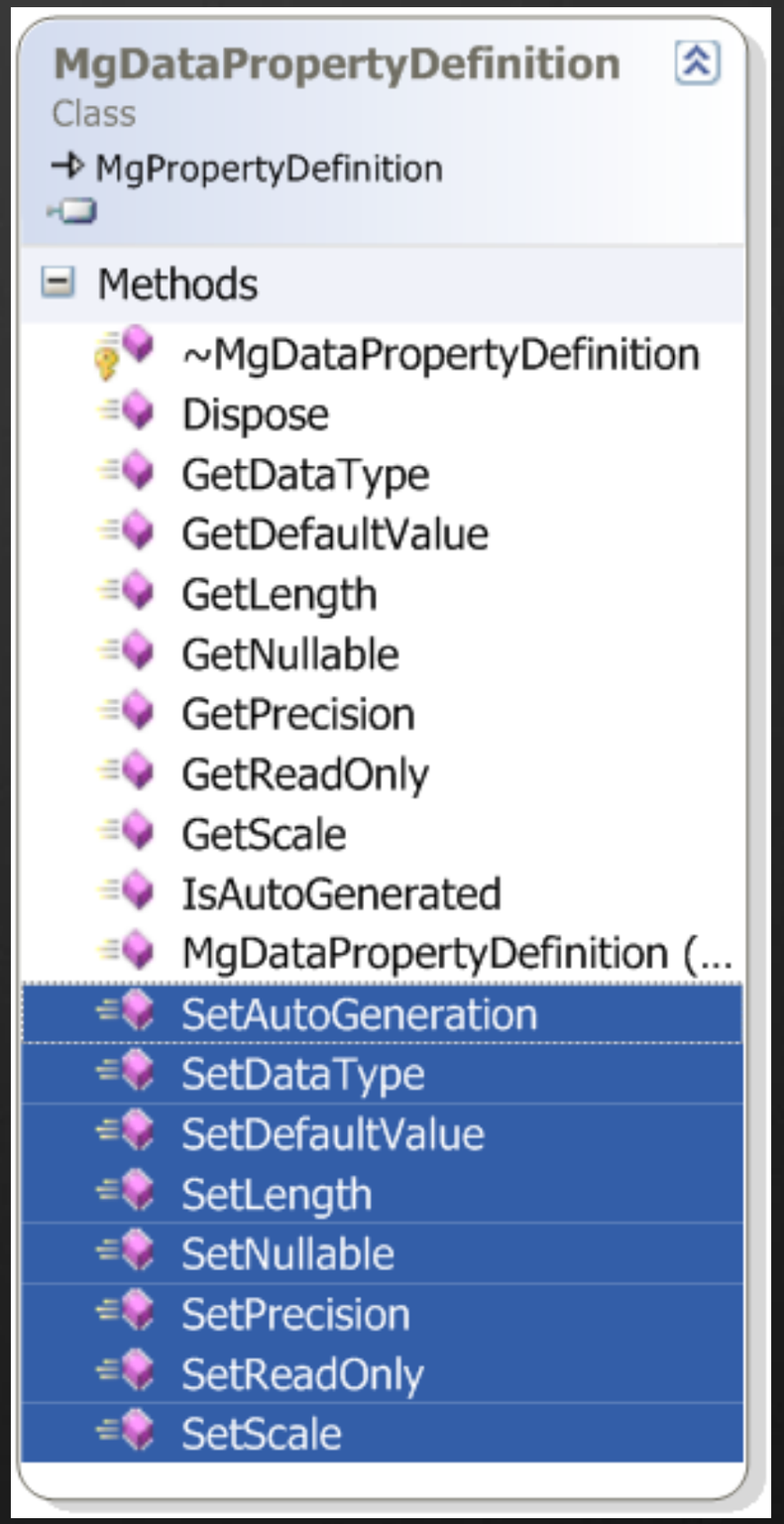
Autodesk

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

Creating a New Feature Class (FDO API)

Step 1

- Create a new (FDO) ClassDefinition object
- Get the property definition collection from the object

```
// The new ClassDefinition object
ClassDefinition parcelClass = new ClassDefinition ();
parcelClass. Name = "tempParcel";

// Get the property definition collection. We'll populate it
PropertyDefinitionCollection props = parcelClass.Properties;
```

Creating a New Feature Class

Step 2

- Populate the property collection with appropriate values

```
//Define the Id as a data property
DataPropertyDefinition id = new DataPropertyDefinition("ID“, “Identity property”);
id.DataType = DataType_Int32;
id. ReadOnly =true;
id. Nullable = false;
id.IsAutoGeneration = true;

//Add the Id as a data property
props.Add(id);

//Add the Id as an identity property
PropertyDefinitionCollection idProps = parcelClass.IdentityProperties;
idProps.Add(id);
```

Creating a New Feature Class

Step 2

- Populate the property collection with appropriate values

```
//Add other data property def. to the collection
DataPropertyDefinition acre = new DataPropertyDefinition("ACRE“, “Parcel acreage”);
acre.DataType = DataType_String;
acre.Length= 256;
props.Add(acre);

//Set up the geometric property definition and add it to property definition collection
GeometricPropertyDefinition geom = new GeometricPropertyDefinition("GEOM“, “Geometric
property”);
geom.GeometryTypes =  4 // for 2-d geometric primitives
geom.HasElevation = false;
geom.SpatialContextAssociation = "LL84";
props.Add(geom);
```

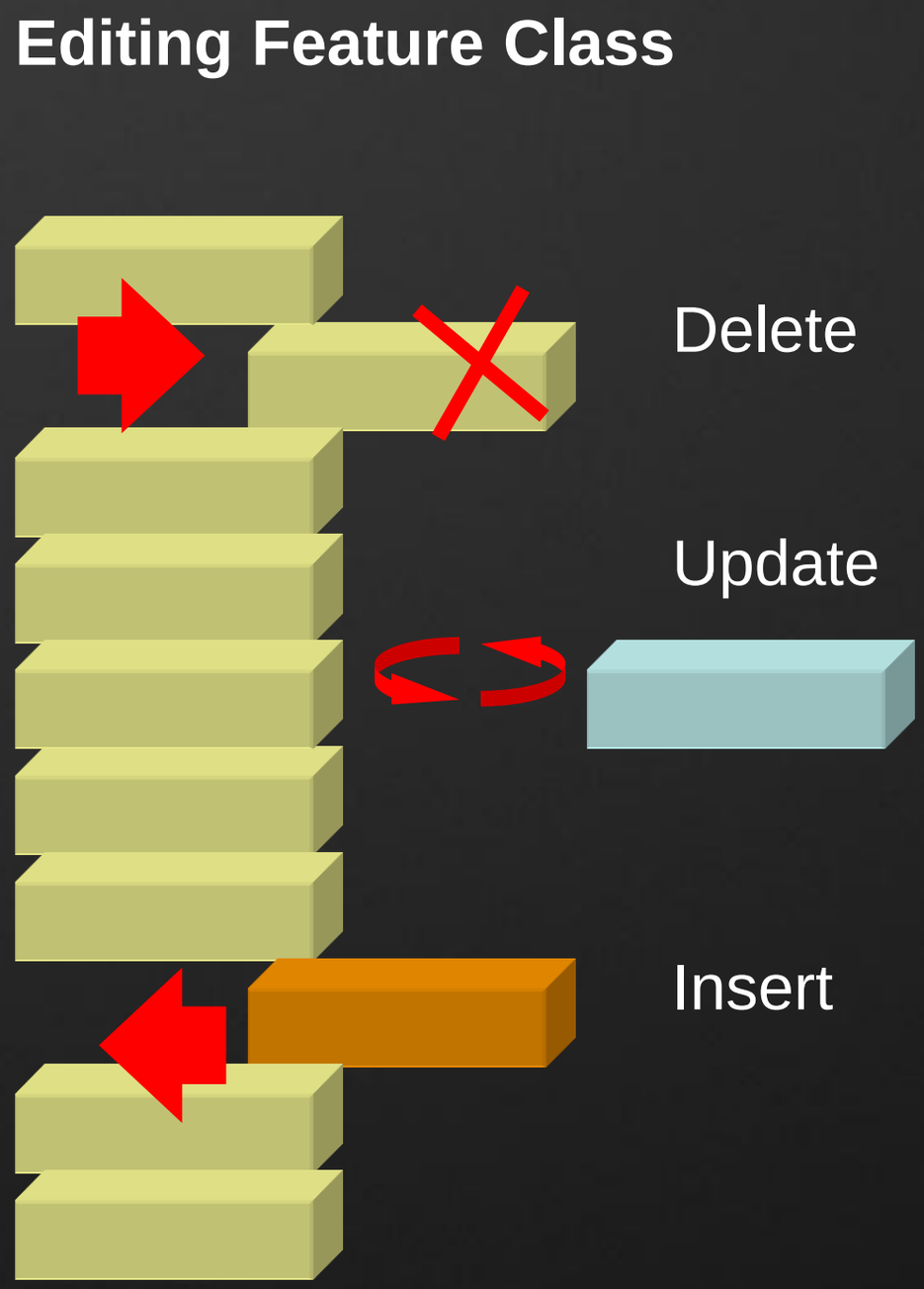
Step 3

- Add feature class to schema

```
//Add the feature class to the schema created earlier
schema.Classes.Add(parcelClass);
```

Editing Feature Class

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



Autodesk

Deleting Features

- Deleting features is performed using MgDeleteFeatures
- 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))')");
```

Updating Features

- Updating features is performed using MgUpdateFeatures
- Steps
  - Create an MgPropertyCollection object, which holds the new property values for the features to be updated.
  - Pass 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))')");
```

Inserting Features, MgInsertFeatures

- Inserting features is performed using MgInsertFeatures
- Steps
  - Create an MgPropertyCollection object, which holds the values of a new feature
  - Pass in the MgPropertyCollection 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);
```

Committing Edits to Feature Class

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

```
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);
```

MgPropertyCollection Class

MgPropertyCollection Class

- Holds the property values used to update a feature class.
- Must contain the values for the non-nullable properties.
- For example, to update a feature class with schema like the one on the right, the MgPropertyCollection object must contain 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();

MgGeometry geom; //assuming this is already defined

//Convert MgGeometry to MgByteReader
MgByteReader byteRdr = agfWriter.Write(geom);

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

Getting Features from Feature Source

- Use AcMapFeatureService ::SelectFeatures() to retrieve features from a feature source
  - Selection can be done with coordinates transformation
  - Enumerate through the returned MgFeatureReader object to get individual feature
  - Get the property values based on the types defined in the feature schema.
  - Always close MgFeatureReader object after use!

```
// Get all features in “VotingDistricts” feature class – no filtering!!
MgFeatureQueryOptions query = new MgFeatureQueryOptions();
MgFeatureReader featureReader = featureService.SelectFeatures(resId,
"VotingDistricts", query);
while (featureReader.ReadNext()) {
  int key = featureReader.GetInt32("ID"); //assuming “ID” is an int32 type
  MgByteReader byteReader = featureReader.GetGeometry(“GEOM");
  MgAgfReaderWriter geometryReaderWriter = new MgAgfReaderWriter();
  MgGeometry geometry = geometryReaderWriter.Read(byteReader);
  MgPoint point = geometry.GetCentroid();
}
featureReader.Close();
```

Query with Filter

- You can select a set of features from a source according to the criteria set in MgFeatureQueryOptions
- Selection can be performed on both feature attributes and feature geometries

```
// Define a  query option and specify it's criteria
MgFeatureQueryOptions query = new MgFeatureQueryOptions();
query.SetFilter("Year >= 1990");
query.SetSpatialFilter("SHPGEOM", geometry,
MgFeatureSpatialOperations.Inside);

// Get the features
MgFeatureReader featureReader =
featureService.SelectFeatures(resId, "feature_class_name", query);
```

Basic Filter

- Mainly used to perform queries on the attributes
- Contains conditions.
  - Comparison

    >, <, <>, >=, <=
  - Like
  - In
- Contains expressions functions.
  - Avg
  - Sum
  - Count
  - Min, Max
  - …

Filter String Examples

```
YEAR > 1990 and Year < 2000
NAME like “Richmond%”
RVALUE in (500000, 1000000)
DATE > ‘1995-3-15’
ADDRESS NULL
Ceil(PRICE1*1.2) < PRICE2*0.8
```

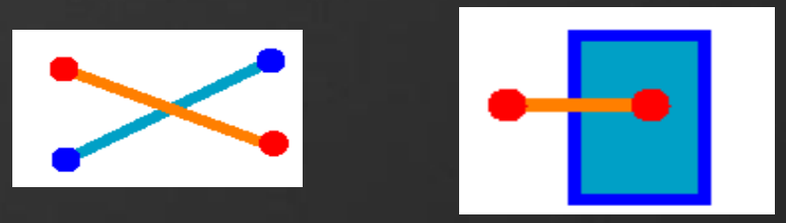
Spatial Filter Operations

- Contains
- Crosses
- Disjoint
- Equals
- Intersects
- Overlaps
- Touches
- Within
- Inside

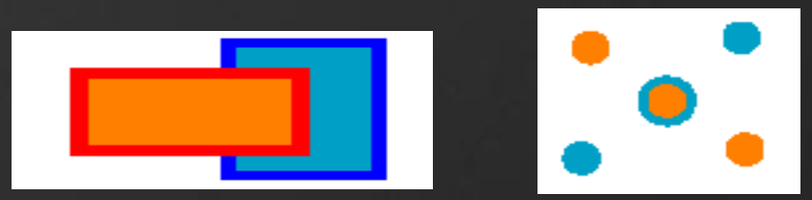
Note: Supported operation is determined by provider's capabilities

Spatial Relationship

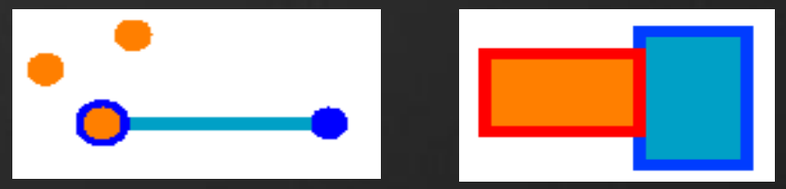
Crosses



Intersects



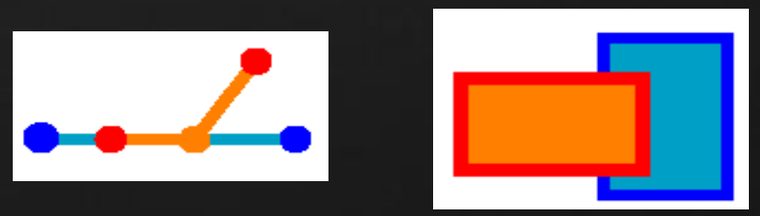
Touches



Within

Disjoint

Overlaps



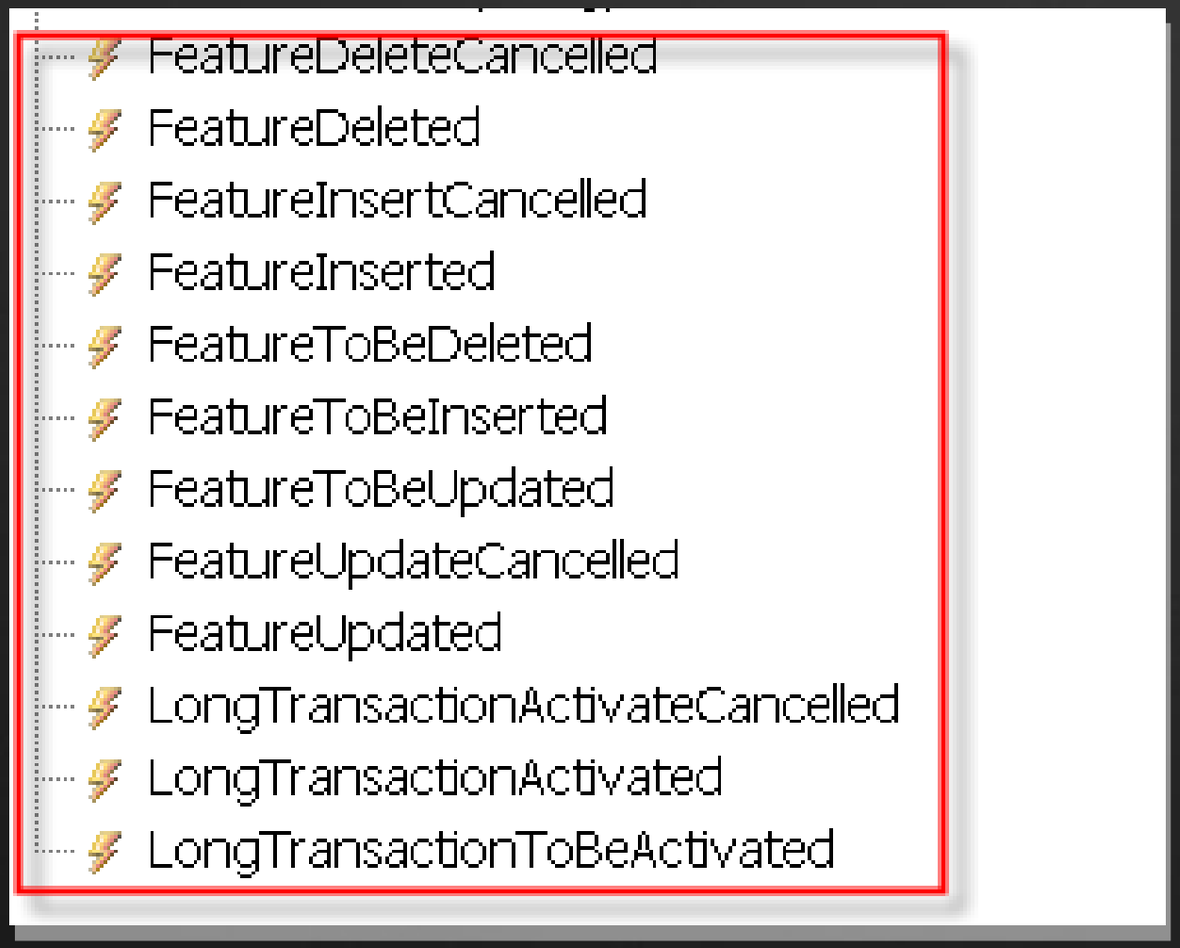
Spatial Filter

- Query based on the relationship of two geometries
- Two ways to set spatial filters on MgFeatureQueryOptions
  - SetFilter (uses the basic filter format)
  - SetSpatialFilter

```
MgFeatureQueryOptions query = new MgFeatureQueryOptions();
String areaWkt = “POLYGON ((0 0, 2 0, 2 2, 0 2, 0 0))”;
query.SetFilter(“SHPGEOM inside GEOMFROMTEXT(” + areaWkt + “)”);

//WKT representation of a geometry
string areaWkt = '"POLYGON XY((1874913.0097 430949.2330, 1874913.0097 463071.9745,1898944.6847 ";
areaWkt += "463071.9745,1898944.6847 430949.2330, 1874913.0097 430949.2330))"';

//spatial relationship inside a basic filter
MgFeatureQueryOptions query = new MgFeatureQueryOptions();
query.SetFilter("Geometry inside GEOMFROMTEXT(" + areaWkt + ")");

//Using SetSpatialFilter
MgGeometry aGeometryObject;
query.SetSpatialFilter("SHPGEOM", aGeometryObject, MgFeatureSpatialOperations.Inside);
```

Feature Service Events

- Occur when feature-related actions, such as deletion, insertion or update are initiated, concluded or cancelled



- Event functions are members of AcMapFeatureService

Autodesk

Feature Service Events

```
//Handling the FeatureInserted event
//declare delegate object
private static FeatureInsertedHandler g_eventHandler = null;
public bool RegisterEvent()
{
   AcMapFeatureService fs = AcMapServiceFactory.GetService(MgServiceType.FeatureService)
   as AcMapFeatureService;

  //Create an instance of the delegate
   g_eventHandler = new FeatureInsertedHandler(this.MyEventHandlerFunction);

  //Associate delegate with the event
   fs.FeatureInserted += g_eventHandler
}

//Implement handler function
private void MyEventHandlerFunction (object sender, AcMapFeatureEventArgs args
{
            MgResourceIdentifier resId = args.GetResourceIdentifier();
             AcMapFeature mapFeature = args.GetFeature();
             string featureClassName = mapFeature.GetClassDefinition().GetName();
}
```

Autodesk®