

AutoCAD® Map 3D 2009

# Geospatial Platform Developer's Guide

Autodesk®

© 2008 Autodesk, Inc. All Rights Reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

#### **Trademarks**

The following are registered trademarks or trademarks of Autodesk, Inc., in the USA and other countries: 3DEC (design/logo), 3December, 3December.com, 3ds Max, ActiveShapes, Actrix, ADI, Alias, Alias (swirl design/logo), AliasStudio, AliasWavefront (design/logo), ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Envision, Autodesk Insight, Autodesk Intent, Autodesk Inventor, Autodesk Map, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSnap, AutoSketch, AutoTrack, Backdraft, Built with ObjectARX (logo), Burn, Buzzsaw, CAiCE, Can You Imagine, Character Studio, Cinestream, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Create>what's>Next> (design/logo), Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, DesignStudio (design/logo), Design Your World, Design Your World (design/logo), DWF, DWG, DWG (logo), DWG TrueConvert, DWG TrueView, DXF, EditDV, Education by Design, Exposure, Extending the Design Team, FBX, Filmbox, FMDesktop, Freewheel, GDX Driver, Gmax, Heads-up Design, Heidi, HOOPS, HumanIK, i-drop, iMOUT, Incinerator, IntroDV, Inventor, Inventor LT, Kaydara, Kaydara (design/logo), LocationLogic, Lustre, Maya, Mechanical Desktop, MotionBuilder, Mudbox, NavisWorks, ObjectARX, ObjectDBX, Open Reality, Opticore, Opticore Opus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProjectPoint, ProMaterials, Reactor, RealDWG, Real-time Roto, Recognize, Render Queue, Reveal, Revit, Showcase, ShowMotion, SketchBook, SteeringWheels, StudioTools, Topobase, Toxik, ViewCube, Visual, Visual Bridge, Visual Construction, Visual Drainage, Visual Hydro, Visual Landscape, Visual Roads, Visual Survey, Visual Syllabus, Visual Toolbox, Visual Tugboat, Visual LISP, Voice Reality, Volo, Wiretap, and WiretapCentral

The following are registered trademarks or trademarks of Autodesk Canada Co. in the USA and/or Canada and other countries: Backburner, Discreet, Fire, Flame, Flint, Frost, Inferno, Multi-Master Editing, River, Smoke, Sparks, Stone, and Wire

All other brand names, product names or trademarks belong to their respective holders.

#### **Disclaimer**

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS." AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

Published By: Autodesk, Inc.  
111 McInnis Parkway  
San Rafael, CA 94903, USA

# Contents

<b>Chapter 1</b>	<b>Overview</b> . . . . .	<b>1</b>
	Introduction . . . . .	1
	Relationship to MapGuide . . . . .	1
	Geospatial Platform API Documentation . . . . .	2
	Setting Up Visual Studio . . . . .	3
	Sample Applications . . . . .	3
<b>Chapter 2</b>	<b>Resources</b> . . . . .	<b>5</b>
	Introduction . . . . .	5
	Working With Resources . . . . .	8
	Differences Between AutoCAD Map 3D and MapGuide . . . . .	10
	Sample - Using Resources . . . . .	11
	The XML Files . . . . .	12
	Adding The Resource . . . . .	13
	Points To Watch . . . . .	15
<b>Chapter 3</b>	<b>Feature Service</b> . . . . .	<b>17</b>
	Overview . . . . .	17
	Defining Feature Sources . . . . .	18
	Example: Defining a Vector Feature Source . . . . .	19
	Example: Defining a Raster Feature Source . . . . .	21
	Joins . . . . .	23
	Adding Feature Classes to a Map . . . . .	26

	Selecting Feature Data . . . . .	28
	Representation of Geometry . . . . .	29
	Selecting Using the API . . . . .	30
	Basic Filters . . . . .	31
	Examples . . . . .	31
	Spatial Filters . . . . .	32
	Creating Geometry Objects From Features . . . . .	33
	Updating Features . . . . .	34
	Edit Sets . . . . .	36
	Creating SDF files . . . . .	36
<b>Chapter 4</b>	<b>Maps and Layers . . . . .</b>	<b>41</b>
	Overview . . . . .	41
	Basic Layer Properties . . . . .	41
	Layer Groups . . . . .	42
	Layer Visibility . . . . .	43
	Example: Actual Visibility . . . . .	43
	Manipulating Layers . . . . .	43
	Changing Visibility . . . . .	44
	Layer Definition . . . . .	44
	Layers With Joined Feature Sources . . . . .	51
	Filtering Layers By Geometry . . . . .	51
	Modifying Layer Style . . . . .	52
<b>Chapter 5</b>	<b>Geometry . . . . .</b>	<b>57</b>
	Overview . . . . .	57
	Geometry Objects . . . . .	57
	Comparing Geometry Objects . . . . .	58
	Coordinate Systems . . . . .	59
	Measuring Distance . . . . .	61
	Creating a Buffer . . . . .	61
	<b>Index . . . . .</b>	<b>65</b>

# Overview



## Introduction

The AutoCAD Map 3D 2009 Geospatial Platform API is used for managing geospatial data in a map. It shares many classes and methods with Autodesk MapGuide® Enterprise and MapGuide Open Source, so applications written for one product can work in another with minimal modification.

The Geospatial Platform API is not the same as the AutoCAD Map 3D ObjectARX .NET API. For details about this API see the ObjectARX .NET Developer's Guide and the ObjectARX .NET API Reference.

## Relationship to MapGuide

Many parts of the Geospatial Platform API are shared between AutoCAD Map 3D and MapGuide Web Server Extensions. There are differences in the products, though, that affect how the API is used.

The MapGuide API is designed to be used in a web server environment, and is available in PHP, Java, and .NET. AutoCAD Map 3D is designed to be used in a desktop environment and exposes only a .NET API.

AutoCAD Map 3D stores some resource information directly in the DWG file, while MapGuide uses an external repository. This is explained in more detail in [Resources](#) on page 5. The MapGuide repository is structured like a file system, with a hierarchy of folders. To ensure as much consistency between the products as possible, AutoCAD Map 3D uses a similar structure for its internal repository.

Some methods in the Geospatial Platform API are not valid in AutoCAD Map 3D, and will throw an exception if called. Generally these will be methods that

do not have corresponding functionality in the AutoCAD Map 3D environment. For example, AutoCAD Map 3D does not have the concept of permissions on resources so any methods dealing with permissions are invalid. These are identified in the API Reference.

All map data in a MapGuide application is written and read through FDO connections. This means that updates occur immediately. AutoCAD Map 3D works with data stored in the DWG file as well as FDO data. Depending on the method chosen, updates in a AutoCAD Map 3D application may be cached and not written directly to the feature source.

Differences in operation between AutoCAD Map 3D and MapGuide mean that in some cases the Geospatial Platform API has a different implementation in the products. AutoCAD Map 3D includes a set of classes that extend the API. For example, the Geospatial Platform API has an `MgLayerBase` class. AutoCAD Map 3D extends this in the `AcMapLayer` class.

All classes that are part of the Geospatial Platform API begin with the prefix `Mg`. Classes that extend the Geospatial Platform API for use in AutoCAD Map 3D begin with the prefix `AcMap`.

AutoCAD Map 3D includes the following enums that are not part of the Geospatial Platform API:

- `EditMode`
- `HighlightMode`

AutoCAD Map 3D also includes event handling, which is not a part of the Geospatial Platform API. For more details see the Geospatial Platform Supplement Reference.

## Geospatial Platform API Documentation

The documentation for the Geospatial Platform API is in 3 places:

- AutoCAD Map 3D Geospatial Platform Developer's Guide (this guide)
- AutoCAD Map 3D Geospatial Platform API Reference
- AutoCAD Map 3D .NET Reference Supplement

The Geospatial Platform API reference contains documentation about classes and methods that are common to AutoCAD Map 3D and MapGuide. When

a method behaves differently in AutoCAD Map 3D this is noted in the reference.

The Geospatial Platform Supplement Reference contains documentation about classes and methods that are used only within AutoCAD Map 3D. In most cases the classes extend classes in the Geospatial Platform API Reference.

The Geospatial Platform API works with the Feature Data Objects (FDO) API. FDO documentation is available with AutoCAD Map 3D, both in the product installation help folder and in the SDK.

---

**NOTE** The API References are available in CHM format. The Developer's Guides are available in PDF and CHM format. To view all CHM documentation as part of one help system, open *sdk.doc.main.chm*.

---

## Setting Up Visual Studio

To use the Geospatial Platform API, follow the instructions in the *AutoCAD Map 3D ObjectARX .NET Developer's Guide*. Add the following reference to the project:

`AcMapApiMgd.dll`

This assembly contains the Geospatial Platform API and adds the namespace `OSgeo.MapGuide`. It is located in the AutoCAD Map 3D installation folder.

## Sample Applications

The SDK includes sample applications in the *MapSamples\Platform* folder. See the *Developer Samples Guide* for details. The samples serve as a good introduction to many aspects of the Geospatial Platform API and can be used as a starting point for developing custom applications.





# Resources

# 2

## Introduction

In the Geospatial Platform API, *resources* are the files and configuration information necessary to draw layers and maps. There are various types of resources required. For example, a `FeatureSource` describes the location, type, and other details needed for access to GIS feature data. A `LayerDefinition` defines the data and style for a layer.

Resources are stored in a *resource repository*. AutoCAD Map 3D has a single repository named `Library`. This repository is contained within the DWG file.

MapGuide uses multiple repositories. The `Library` repository contains persistent, site-wide data. There is a single `Library` repository for each MapGuide site. There are multiple `Session` repositories, each one containing data from a single MapGuide session. `Session` repositories are unique to an individual session and cannot be shared. All MapGuide repositories are managed on the site server.

Allowable resource types are defined as static members of the class `MgResourceType`. Some resource types apply only to MapGuide.

The following resource types are valid for both Map 3D and MapGuide:

Resource Type	Description
<code>FeatureSource</code>	Contains the required parameters for connecting to a geospatial feature source.
<code>LayerDefinition</code>	Contains the required parameters for displaying and styling a layer. Layers can be drawing layers, vector layers, or grid (raster) layers.

Resource Type	Description
SymbolDefinition	Defines a symbol to be displayed on a map.

The following resource types are only valid for MapGuide:

Resource Type	Description
ApplicationDefinition	Defines a flexible Web layout for the Fusion framework.
DrawingSource	Contains the required parameters for connecting to a DWG file.
Folder	A folder in the resource repository.
LoadProcedure	Contains the required parameters for loading new data into the MapGuide repository.
Map	Contains the run-time definition of a map.
MapDefinition	Defines an initial map state, used as the basis for creating a run-time map.
PrintLayout	Defines the components of a printed map created from MapGuide.
Selection	Contains selection information.
SymbolLibrary	Defines a library of symbols.
WebLayout	Defines the components of a Web layout for a MapGuide Viewer.

Resource repositories are structured like directories, with folders, subfolders, and documents. Each resource is an XML document in the repository, named with a unique resource identifier. A resource identifier is made up of the following parts:

- Repository type—either “Library” or “Session”. Map 3D only uses “Library”.

- Repository name—for library repositories, an empty string. For session repositories, a unique session identifier assigned by the site server.
- Path—the path to the folder containing the resource.
- Name—the resource name, without the extension.
- Resource type—the resource type (the extension). This must match one of the allowable types defined in `MgResourceType`.

For example, the following could be a name for a feature source in either Map 3D or MapGuide:

```
Library://Samples/CityOutline.FeatureSource
```

The following could be a name for a map definition in MapGuide:

```
Session:70ea89fe-0000-1000-8000-005056c00008_en//Map.MapDefinition
```

## Resource Service

Resources are managed by the resource service, an `MgResourceService` object.

In AutoCAD Map 3D, use `AcMapServiceFactory` to get the resource service:

```
MgResourceService resourceService =  
    AcMapServiceFactory.GetService(MgServiceType.ResourceService)  
    as MgResourceService;
```

In MapGuide, use `MgSiteConnection` to get the resource service:

```
MgSiteConnection siteConnection = new MgSiteConnection();  
siteConnection.Open(userInfo);  
MgResourceService resourceService =  
    siteConnection.CreateService(MgServiceType.ResourceService)  
    as MgResourceService;
```

## Resource Dependencies

Resources may depend on other items:

- Some resources are self sufficient and do not refer to any other resources or files.
- Some resources reference other resources. For example, layer definitions and feature sources are stored as separate resources. A layer definition resource contains a reference to the resources for the feature sources that are used in that layer.

- Some resources use associated *resource data*. For example, this is used to store configuration information for ODBC/WMS/Raster feature sources.

### Resource Schemas

Resources are stored in the repository as XML documents. Applications that add or modify resources must ensure that the XML validates against the appropriate schema.

For AutoCAD Map 3D, the schemas are available as part of the SDK, in the *Schema* folder.

For MapGuide, the schemas are installed with the server files, in *InstallDir\Server\Schema*.

The schemas are documented in the XML Schemas module of the *Geospatial Platform Reference*.

## Working With Resources

Resources in the repository can be managed by using the `MgByteSource`, `MgByteReader`, and `MgByteSink` objects.

To convert a string representation of the XML data into data for a resource, first create an `MgByteSource`. New `MgByteSource` objects can be initialized with a string or the contents of a file. Get an `MgByteReader` from the byte source and pass that to `MgResourceService.SetResource()`.

For example, if an external file contains the string representation of the XML for a layer definition, the following would read the contents of the file and store it in the repository.

```
MgResourceIdentifier rID_layer = new MgResourceIdentifier(
    @"Library://Data/Zoning.LayerDefinition");
MgByteSource layer_byteSource = new MgByteSource(xmlFilePath);
layer_byteSource.SetMimeType("text/xml");
resourceService.SetResource(rID_layer,
    layer_byteSource.GetReader(), null);
```

There are many ways of modifying the XML in a resource. The best method to use in a given situation depends on the development environment, the developer's familiarity with a given technique, and the desired result. Some possibilities are described in the following sections. Other techniques for working with XML data are equally valid.

### **xsd.exe**

For .NET development, used with AutoCAD Map 3D and the ASP.NET API of MapGuide, it is possible to use the tool `xsd.exe`, which is available with the SDK that comes with Microsoft Visual Studio. `xsd.exe` reads an XML schema and generates .NET classes for working with XML documents based on the schema. `xsd.exe` can generate both VB.NET and C# classes.

To use `xsd`, a typical command is:

```
xsd.exe LayerDefinition-1.0.0.xsd /c /l:cs /n:OSGeo.Map  
Guide.Schema.LayerDefinition
```

For an example of how to use these classes, see the *Developer Samples Guide*.

For more information on `xsd`, see <http://msdn.microsoft.com>.

### **layerdefinitionfactory.php**

MapGuide includes *layerdefinitionfactory.php*, which contains PHP methods for working with the *LayerDefinition* schema. These methods are designed to handle most basic tasks with layer definitions, but do not provide complete support for all possibilities. They create the XML by substituting strings into external templates. It is relatively easy to adapt the techniques from *layerdefinitionfactory.php* for use with a different schema or to translate them into one of the other development languages.

### **Autodesk MapGuide Studio**

One of the simplest ways to create XML templates is to use Autodesk MapGuide Studio with Autodesk MapGuide Enterprise or MapGuide Open Source. Using Studio, create the resources with the desired values, then save them as XML. The resulting file can be edited using a text editor, XML editor, or with techniques like those in *layerdefinitionfactory.php*.

### **Document Object Model**

Finally, all of the development languages support versions of the Document Object Model (DOM) API, which is a general-purpose API for working with and validating XML files. Using the DOM API requires converting data into a string, then converting that string into an XML document. For example, the following takes resource data from the repository and loads it into an XML document:

```
// using System.Xml;

MgByteReader byteReader =
    resourceService.GetResourceContent(resourceId);
XmlDocument doc = new XmlDocument();
doc.LoadXml(byteReader.ToString());
```

Modify the XML document using standard DOM methods, then convert it back to a string and re-save the resource.

```
String xmlString = doc.DocumentElement.OuterXml;
resourceService.SetResource(resourceId,
    new MgByteReader(xmlString, "text/xml"), null);
```

## Differences Between AutoCAD Map 3D and MapGuide

The Platform API has been made to be as consistent as possible between AutoCAD Map 3D and MapGuide. However, there are some differences, especially in the way they handle resources.

- AutoCAD Map 3D does not use `ApplicationDefinition`, `DrawingSource`, `Folder`, `LoadProcedure`, `Map`, `MapDefinition`, `PrintLayout`, `Selection`, `SymbolLibrary`, or `WebLayout` resources.
- In AutoCAD Map 3D, the resources are stored inside the DWG drawing file or a DWT template drawing, not in an external database. This means that resources and their paths are specific to that drawing. In other words, the same repository path in another drawing may not be defined, or may refer to a different resource. If you need to share resources between drawings, you must add them to each drawing.
- Session repositories are not supported in AutoCAD Map 3D. All resources must be stored in the `Library` repository.
- AutoCAD Map 3D does not use resource headers. For any method that allows them, for example `SetResource()`, enter `null` for that parameter.
- The following methods in `MgResourceService` are not supported in AutoCAD Map 3D:
  - `ApplyResourcePackage()`
  - `ChangeResourceOwner()`

- `GetRepositoryContent()`
  - `GetRepositoryHeader()`
  - `GetResourceHeader()`
  - `UpdateRepository()`
- The method of specifying the resource data for a file differs. In MapGuide, the procedure is:
- 1 In the feature source XML, provide the name of the SDF file. For example:
 

```
<Value>%MG_DATA_FILE_PATH%
HydrographicPolygons.sdf</Value>
```
  - 2 Use `ResourceService.SetResourceData()` to specify the location of the resource data (the SDF file) on disk.

In AutoCAD Map 3D, the procedure is to specify the absolute location of the resource data in the feature source XML. For example:

```
<Parameter>
  <Name>File</Name>
  <Value>C:\Map ObjectARX SDK\Map Samples\Platform\BuildMap\Data\SDF\Buffered.sdf</Value>
</Parameter>
```

No call to `ResourceService.SetResourceData()` is required.

AutoCAD Map 3D *does* use `ResourceService.SetResourceData()` for streams, but only for the configuration information for ODBC/WMS/Raster feature sources.

## Sample - Using Resources

This example shows how to add a resource programmatically, using external files that contain the XML for the resources.

## The XML Files

This sample uses data from the BuildMap developer sample. When you run BuildMap, it writes these files to the directory containing BuildMap.dll. See the *Developer Samples Guide* for details.

One file is *Zoning.layer*. This specifies all the properties for drawing one layer in a map. Note the <ResourceId> element, which refers to the resource for the feature source.

```
<?xml version="1.0" encoding="utf-8"?>
<LayerDefinition
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  version="1.0.0">
  <VectorLayerDefinition>
    <ResourceId>Library://Data/SDF/Zoning.FeatureSource
    </ResourceId>
    <FeatureName>Schema:Zoning</FeatureName>
    <FeatureNameType>FeatureClass</FeatureNameType>
    <Geometry>Geometry</Geometry>
    <VectorScaleRange>
      <AreaTypeStyle>
        <AreaRule>
          <LegendLabel />
          <AreaSymbolization2D>
            <Fill>
              <FillPattern>Solid</FillPattern>
              <ForegroundColor>a03cafda</ForegroundColor>
              <BackgroundColor>FF000000</BackgroundColor>
            </Fill>
            <Stroke>
              <LineStyle>Solid</LineStyle>
              <Thickness>0.0</Thickness>
              <Color>FF000000</Color>
              <Unit>Centimeters</Unit>
            </Stroke>
          </AreaSymbolization2D>
        </AreaRule>
      </AreaTypeStyle>
    </VectorScaleRange>
  </VectorLayerDefinition>
</LayerDefinition>
```



The second file is *Zoning.FeatureSource*, which defines the properties of an SDF feature source. The File parameter points to the location on disk. See [Feature Service](#) on page 17 for more details.

```
<?xml version="1.0" encoding="utf-8"?>
<FeatureSource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Provider>OSGeo.SDF.3.3</Provider>
  <Parameter>
    <Name>File</Name>
    <Value>C:\Map 3D SDK\Map Samples\Platform\BuildMap\Data\SDF\Zoning.sdf</Value>
  </Parameter>
  <Parameter>
    <Name>ReadOnly</Name>
    <Value>False</Value>
  </Parameter>
</FeatureSource>
```

## Adding The Resource

To use the XML files to create the layer and the feature source, use `MgResourceService.SetResource()`. For example:

```

// Add the layer definition.
// It will be stored in the repository using the resource
// identifier.
MgResourceIdentifier rID_layer = new MgResourceIdentifier(
    @"Library://Data/Zoning.LayerDefinition");

// Note: Modify this directory to point to the location of
// the XML files.

String xmlDirectory = @"C:\Map 3D SDK\
    + @"Map Samples\Platform\BuildMap\bin\Debug\";

// Read the XML file, then save its contents in the resource
// repository.

MgByteSource layer_byteSource =
    new MgByteSource(xmlDirectory + "Zoning.layer");
layer_byteSource.SetMimeType("text/xml");
resourceService.SetResource(rID_layer,
    layer_byteSource.GetReader(), null);

// Add the feature source
// Note: The layer definition XML refers to
// 'Library://Data/SDF/Zoning.FeatureSource' so
// we must use that name here.
MgResourceIdentifier rID_feature_source =
    new MgResourceIdentifier(
        @"Library://Data/SDF/Zoning.FeatureSource");
MgByteSource feature_byteSource = new
    MgByteSource(xmlDirectory + "Zoning.FeatureSource");
feature_byteSource.SetMimeType("text/xml");
resourceService.SetResource(rID_feature_source,
    feature_byteSource.GetReader(), null);

```

Then add the layer to the current map. For example:

```

MgLayerBase layer = AcMapLayer.Create(rID_layer,
    resourceService);
layer.SetName("NewLayer");
AcMapMap currentMap = AcMapMap.GetCurrentMap();
currentMap.GetLayers().Add(layer);

```

## Points To Watch

- In the feature source XML, the provider name (for example, `OSGeo.SDF.3.3`) must match one of the provider names listed in the output of `MgFeatureService.GetFeatureProviders()`.

- In the layer definition, the `ResourceId` value must match the path of the feature source. In the example above, this path is used in the layer definition:

```
<ResourceId>Library://Data/SDF/Zoning.FeatureSource</ResourceId>
```

So the corresponding path must be used when the feature source is created. For example:

```
MgResourceIdentifier rID_feature_source =  
    new MgResourceIdentifier(  
        @"Library://Data/SDF/Zoning.FeatureSource");
```

- Create new layers using the static method `AcMapLayer.Create()`.
- Create new layer groups using the `AcMapLayerGroup` constructor `AcMapLayerGroup(groupName)`.



# Feature Service

# 3

## Overview

A feature source represents a single FDO (Feature Data Objects) connection. FDO is an API for reading and writing geospatial data in a variety of formats. For more details about FDO, see the documentation included with AutoCAD Map 3D and the SDK, or visit [fdo.osgeo.org](http://fdo.osgeo.org).

Feature Service provides a common API for reading and writing feature data from data sources for which an FDO provider exists.

Different feature sources have different capabilities. For example, an Oracle Spatial database will have more capabilities than an ODBC connection. Some feature sources can have multiple schemas.

One schema in a feature source can describe multiple feature classes. For example, a single SDF file could contain a feature class for roads and another feature class for hydrography.

In AutoCAD Map 3D, a single FDO feature class corresponds to a single map layer. There cannot be more than one feature class per layer, and a layer containing feature data cannot contain AcDb entities.

---

**NOTE** All Feature Service operations work with features belonging to the current map. To get the current map, call `AcMapMap.GetCurrentMap()`.

---

## Defining Feature Sources

To define a feature source within AutoCAD Map 3D, create a resource identifier for the feature source and store it in the resource repository. Then call `MgResourceService.SetResource()` with three parameters:

- An `MgResourceIdentifier` that contains the path in the resource repository for the resource.
- An `MgByteReader` that provides the XML describing the feature source. This conforms to `FeatureSource.xsd`.
- A null parameter for the resource header. This is not used by AutoCAD Map 3D.

The repository path is of the form

```
Library://resourcePath/resourceName.FeatureSource
```

`resourcePath` is optional, but `FeatureSource` (case sensitive) is mandatory. For example, the following creates an `MgResourceIdentifier`:

```
MgResourceIdentifier parcelId =  
    new MgResourceIdentifier("Library://parcels.FeatureSource");
```

The exact form of the XML describing the feature source varies depending on the FDO provider. See the schema documentation for `FeatureSource.xsd` and the *Connection API* section of *The Essential FDO* for details. Different FDO providers accept different parameters, as described in *The Essential FDO*. For example, the `Autodesk.Raster.3.2` provider accepts a single parameter, `DefaultRasterFileLocation`. This is set in a `Name/Value` parameter in the XML data.

```

<?xml version="1.0" encoding="utf-8"?>
<FeatureSource
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Provider>Autodesk.Raster.3.2</Provider>
  <Parameter>
    <Name>DefaultRasterFileLocation</Name>
    <Value>C:\some\folder\rasterfile.jpg</Value>
  </Parameter>
  <ConfigurationDocument>config://rasterfile.jpg
  </ConfigurationDocument>
  <LongTransaction />
</FeatureSource>

```

Some providers require additional configuration. For example, the `Autodesk.Raster.3.2` provider requires georeferencing information to define the location and resolution of the raster data for some raster file formats. This can be in a world file in the same folder as the raster file, or it can be in a separate FDO configuration file. WMS and ODBC providers can also require separate configuration files.

For those providers that do require a configuration file, the `FeatureSource` XML contains a `<ConfigurationDocument>` element that points to resource data. Call `MgResourceService.SetResourceData()` to set the data. The data name must match the name in the `<ConfigurationDocument>` element. The `BuildMap` sample uses `config://` as part of the resource data name, but this is not required.

## Example: Defining a Vector Feature Source

Defining a vector feature source requires creating the feature source definition and storing it in the resource repository using the feature source resource identifier. The following example assumes that `SDFpath` is a string containing the file path of an SDF file. It creates a feature source with a resource identifier of `Library://feature.FeatureSource`.

There are many ways to create a feature source definition. One method, used in the sample applications and the following example, uses classes generated using the Visual Studio tool `xsd.exe` on *FeatureSource.xsd*. `FeatureSourceType` and `NameValuePairType` are classes generated this way.

---

**NOTE** Resource service requires data to be UTF-8 encoded.

---

```

// Get the services

MgResourceService rs;
rs = AcMapServiceFactory.GetService(MgServiceType.ResourceService)
    as MgResourceService;
MgFeatureService fs;
fs = AcMapServiceFactory.GetService(MgServiceType.FeatureService)
    as MgFeatureService;
MgResourceIdentifier fsId = new MgResourceIdentifier(
    "Library://feature.FeatureSource");

// Create the feature source definition with a required
// parameter of File and an optional parameter of ReadOnly

string xmlString;
FeatureSourceType fsType = new FeatureSourceType();

fsType.Provider = "OSGeo.SDF.3.2";

NameValuePairType param = new NameValuePairType();
param.Name = "File";
param.Value = SDFpath;
NameValuePairType param2 = new NameValuePairType();
param2.Name = "ReadOnly";
param2.Value = "false";
fsType.Parameter = new NameValuePairType[] { param, param2 };

// Serialize the feature source object model to xml string
using (StringWriter writer = new StringWriter())
{
    XmlSerializer xs = new XmlSerializer(fsType.GetType());
    xs.Serialize(writer, fsType);
    xmlString = writer.ToString();
}

// Convert the Unicode string to UTF8 bytes for Resource Service

byte[] unicodeBytes = Encoding.Unicode.GetBytes(xmlString);
byte[] utf8Bytes = Encoding.Convert(Encoding.Unicode,
    Encoding.UTF8, unicodeBytes);

// Create a byte reader containing the XML feature
// source definition. Store the definition in the repository

```



```
MgByteSource xmlSource = new MgByteSource(utf8Bytes,
    utf8Bytes.Length);
rs.SetResource(fsId, xmlSource.GetReader(), null);
```

## Example: Defining a Raster Feature Source

Defining a raster feature source is similar to defining a vector feature source. The parameters vary depending on the provider. Most raster files do not contain coordinate system information, so it must be obtained some other way. This can be done using a world file or through additional configuration for the FDO provider. The following example uses the second method, adding resource data containing the configuration information.

The resource identifier contains the location in the resource repository for the feature source definition.

```
MgResourceIdentifier rasterId = new MgResourceIdentifier(
    "Library://rasterFeature.FeatureSource");
```

```

string xmlString;
FeatureSourceType fsType = new FeatureSourceType();

fsType.Provider = "Autodesk.Raster.3.2";

NameValuePairType param = new NameValuePairType();
param.Name = "DefaultRasterFileLocation";
param.Value = @"C:\some\folder\rasterfile.jpg";
fsType.Parameter = new NameValuePairType[] { param };

fsType.ConfigurationDocument = "config://rasterfile.jpg";
fsType.LongTransaction = "";

// Serialize the feature source object model to an xml string
using (StringWriter writer = new StringWriter())
{
    XmlSerializer xs = new XmlSerializer(fsType.GetType());
    xs.Serialize(writer, fsType);
    xmlString = writer.ToString();
}

byte[] unicodeBytes = Encoding.Unicode.GetBytes(xmlString);
byte[] utf8Bytes = Encoding.Convert(Encoding.Unicode,
    Encoding.UTF8, unicodeBytes);
MgByteSource xmlSource =
    new MgByteSource(utf8Bytes, utf8Bytes.Length);

MgResourceService rs;
rs = AcMapServiceFactory.GetService(MgServiceType.ResourceService)
    as MgResourceService;
rs.SetResource(rasterId, xmlSource.GetReader(), null);

```

The configuration information varies depending on the FDO provider. The WMS, ODBC, and Raster providers may need additional configuration information. Refer to the FDO documentation in the Map SDK and in the AutoCAD Map 3D installation folder for more details.

The following reads the configuration from a file and stores it in the resource repository. Ensure that the data name matches the `<ConfigurationDocument>` element in the feature source definition.

```

MgByteSource source;
source = new MgByteSource(@"C:\some\folder\rasterconfig.xml");
rs.SetResourceData(rasterId, "config://rasterfile.jpg",
    MgResourceDataType.Stream, source.GetReader());

```

# Joins

Joins extend a feature source by combining it with data from another feature source, similar to a database join. They are commonly used to combine GIS data with data from a database.

For example, an SDF file containing the following properties:

- parcel geometry
- parcel ID

could be joined with an ODBC database containing the following properties:

- parcel ID
- owner name
- assessment value
- land use designation

The join is defined in the feature source definition. Each join requires 2 feature classes. The primary feature class contains the geometry for display in AutoCAD Map 3D. The secondary feature class does not need geometry, but it must have a property that matches a property in the primary feature class. In the example above, the SDF file contains the primary feature class and the ODBC database contains the secondary feature class.

In most cases the feature classes will be from different feature sources, but it is possible to create a self-referencing join within 1 feature class or a join between 2 feature classes in the same feature source.

To create a joined feature source, begin with the secondary feature source. This does not require anything special. For example, the following is a simple connection to an ODBC database:

```

<FeatureSource
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:noNamespaceSchemaLocation="MapFeatureSource-1.0.0.xsd"
  version="1.0.0">
  <Provider>OSGeo.ODBC.3.2</Provider>
  <Parameter>
    <Name>Password</Name>
    <Value></Value>
  </Parameter>
  <Parameter>
    <Name>GenerateDefaultGeometryProperty</Name>
    <Value>>false</Value>
  </Parameter>
  <Parameter>
    <Name>ConnectionString</Name>
    <Value></Value>
  </Parameter>
  <Parameter>
    <Name>DataSourceName</Name>
    <Value>OWNERS</Value>
  </Parameter>
  <Parameter>
    <Name>UserId</Name>
    <Value></Value>
  </Parameter>
  <ConfigurationDocument></ConfigurationDocument>
  <LongTransaction></LongTransaction>
</FeatureSource>

```

The primary feature source defines the join. It must identify the secondary feature source and which properties are to be used for the join. For example, the following is a feature source definition for an SDF file with a join to an ODBC database:

```

<FeatureSource
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:noNamespaceSchemaLocation="MapFeatureSource-1.0.0.xsd"
  version="1.0.0">
  <Provider>OSGeo.SDF.3.2</Provider>
  <Parameter>
    <Name>ReadOnly</Name>
    <Value>false</Value>
  </Parameter>
  <Parameter>
    <Name>File</Name>
    <Value>C:\some\folder\Parcels.sdf</Value>
  </Parameter>
  <ConfigurationDocument></ConfigurationDocument>
  <LongTransaction></LongTransaction>
  <Extension>
    <Name>Parcels_Joins1</Name>
    <FeatureClass>SHP_Schema:Parcels</FeatureClass>
    <AttributeRelate>
      <AttributeClass>Fdo:Parcels</AttributeClass>
      <ResourceId>Library://ODBC_1</ResourceId>
      <Name>Parcels</Name>
      <AttributeNameDelimiter>|</AttributeNameDelimiter>
      <RelateType>LeftOuter</RelateType>
      <ForceOneToOne>true</ForceOneToOne>
      <RelateProperty>
        <FeatureClassProperty>APN</FeatureClassProperty>
        <AttributeClassProperty>APN</AttributeClassProperty>
      </RelateProperty>
    </AttributeRelate>
  </Extension>
</FeatureSource>

```

The `<Extension>` element defines the join. `<Name>` is the name of the join, used in a layer definition that references the feature source. If a feature source defines more than one join they must have different names. `<FeatureClass>` is a feature class in the primary feature source.

`<AttributeRelate>` defines a join to a single feature source. It is possible for the primary feature source to have joins to more than one secondary feature source. This is represented by multiple `<AttributeRelate>` elements. As an example, the owner information could be in one database table and assessment information could be in another.

For each join, `<AttributeClass>` defines the schema and feature class in the secondary feature source. `<ResourceId>` is the resource identifier. `<RelateType>` is the type of join. This release of AutoCAD Map 3D supports left outer joins and inner joins.

The join can have multiple `<RelateProperty>` elements. These define the fields that must match for the join.

Store the feature source definition using `MgResourceService.SetResource()`, as described in [Defining Feature Sources](#) on page 18.

## Adding Feature Classes to a Map

To add a feature class to a map, create a layer definition for the new layer and add it to the resource repository. Create an `AcMapLayer` and set its layer definition. Add the layer to the map using

```
AcMapMap.GetCurrentMap().GetLayers().Add().
```

Creating the layer definition is similar to creating the feature source definition. See [Defining Feature Sources](#) on page 18. The layer definition uses the `LayerDefinition.xsd` schema, which includes styling information. See [Layer Definition](#) on page 44 for details.

The `<FeatureName>` element in the layer definition is of the form

```
schemaName:className
```

where *schemaName* and *className* are valid for the feature source. Call `MgFeatureService.GetSchemas()` to get the schema names for a feature source and `MgFeatureService.GetClasses()` to get the class names for a schema.

The following example creates a layer definition for a raster layer and adds the layer to the current map.

```

AcMapMap currentMap = AcMapMap.GetCurrentMap();
string layerDefName = "Library://rasterlayer.LayerDefinition";
MgResourceIdentifier layerId = new
    MgResourceIdentifier(layerDefName);

// Use classes from xsd.exe to build the layer definition

LayerDefinitionType layerDef = new LayerDefinitionType();
GridLayerDefinitionType gridLayerDef = new
    GridLayerDefinitionType();
layerDef.Item = gridLayerDef;
gridLayerDef.ResourceId = rasterId.ToString();
gridLayerDef.FeatureName = "rasters:classname";
gridLayerDef.Geometry = "Image";

GridScaleRangeType[] ranges = new GridScaleRangeType[1];
gridLayerDef.GridScaleRange = ranges;

ranges[0] = new GridScaleRangeType();
ranges[0].ColorStyle = new GridColorStylizationType();

GridColorRuleType[] colorRules = new GridColorRuleType[1];
ranges[0].ColorStyle.ColorRule = colorRules;

colorRules[0] = new GridColorRuleType();
colorRules[0].LegendLabel = "";
colorRules[0].Color = new GridColorType();
colorRules[0].Color.ItemElementName = ItemChoiceType.Band;
colorRules[0].Color.Item = "1";
ranges[0].RebuildFactor = 1;

// Serialize the layer definition to XML

string layerDefString;
using (StringWriter writer = new StringWriter())
{
    XmlSerializer xs = new XmlSerializer(layerDef.GetType());
    xs.Serialize(writer, layerDef);
    layerDefString = writer.ToString();
}

// Convert Unicode to UTF-8

```

```

unicodeBytes = Encoding.Unicode.GetBytes(layerDefString);
utf8Bytes = Encoding.Convert(Encoding.Unicode, Encoding.UTF8,
    unicodeBytes);

// Create a byte reader containing the layer source definition

xmlSource = new MgByteSource(utf8Bytes, utf8Bytes.Length);
rs.SetResource(layerId, xmlSource.GetReader(), null);

// Add the layer to the Map
MgLayerBase layer = new AcMapLayer(layerId, rs);
layer.Name = "newLayer";
currentMap.GetLayers().Add(layer);

```

This adds the new layer to the end of the layer collection, at the top of the draw order.

Layers can be organized into groups. To add a layer to a group call `AcMapLayer.SetGroup()` with the group name.

## Selecting Feature Data

Individual features within a feature source are identified by unique feature IDs. The form of the ID depends on the feature source. Call `MgFeatureService.GetClassDefinition()` to get the feature class definition. Call `MgClassDefinition.GetIdentityProperties()` to get a list of all properties that are identity properties.

To get the currently selected features for a map, call `AcMapMap.GetFeatureSelection()`. This returns an `MgSelectionBase` object. A single selection can contain features from multiple layers and classes. Call `MgSelectionBase.GetLayers()` to get all layers containing selected objects, or select an individual layer.

For example, to find selected features on the `Parcels` layer,



```

AcMapMap currentMap = AcMapMap.GetCurrentMap();
MgFeatureService fs =
    AcMapServiceFactory.GetService(MgServiceType.FeatureService)
    as MgFeatureService;

MgLayerCollection layers = currentMap.GetLayers();
MgLayerBase parcelsLayer = layers.GetItem("Parcels");
string fcName = parcelsLayer.GetFeatureClassName();

MgSelectionBase selection = currentMap.GetFeatureSelection();
string selectionFilter = selection.GenerateFilter(parcelsLayer,
    fcName );

MgFeatureQueryOptions queryOpts = new MgFeatureQueryOptions();
queryOpts.SetFilter(selectionFilter);
MgFeatureReader featureReader;

MgResourceIdentifier fsId = new
    MgResourceIdentifier(parcelsLayer.GetFeatureSourceId());
featureReader = fs.SelectFeatures(fsId, fcName, queryOpts);

```

A selection consists of one or more feature IDs. To process the features in a selection, create an `MgFeatureReader` that contains the selected features, then call `MgFeatureReader.ReadNext()` to advance through the feature reader. To create the `MgFeatureReader()` for the selected features in a layer, call `MgSelectionBase.GenerateFilter()`, which creates a filter to select the features. Then create and initialize an `MgFeatureQueryOptions` object. Finally, call `AcMapFeatureService.SelectFeatures()`.

An `MgFeatureReader` iterates through a list of features from a single feature source. To advance to the next feature in the reader, call `MgFeatureReader.ReadNext()`.

## Representation of Geometry

AutoCAD Map 3D can represent geometric data in 3 different forms:

- AGF text format, which is an extension of the Open Geospatial Consortium (OGC) Well Known Text (WKT) format. This is used to represent geometry as a character string.
- Binary AGF format. The is used by the FDO technology supporting the Feature Service.

- Internal representation, using `MgGeometry` and classes derived from it.

To convert between AGF text and the internal representation, use an `MgWktReaderWriter` object. Call `MgWktReaderWriter.Read()` to convert AGF text to `MgGeometry`. Call `MgWktReaderWriter.Write()` to convert `MgGeometry` to AGF text.

To convert between binary AGF and the internal representation, use an `MgAgfReaderWriter` object. Call `MgAgfReaderWriter.Read()` to convert binary AGF to `MgGeometry`. Call `MgAgfReaderWriter.Write()` to convert `MgGeometry` to binary AGF.

For example, if you have a WKT representation of the geometry, you could create a geometry object as follows:

```
MgWktReaderWriter wktReaderWriter = new MgWktReaderWriter();  
MgGeometry geometry = wktReaderWriter.Read(wktGeometry);
```

## Selecting Using the API

Selections can be created programatically with the Platform API. This is done by querying data in a feature source, creating a feature reader that contains the features, then converting the feature reader to a selection (`MgSelection` object).

To create a feature reader, apply a selection filter to a feature class in the feature source. A selection filter can be a *basic filter*, a *spatial filter*, or a combination of the two. The filter is stored in an `MgFeatureQueryOptions` object.

Basic filters are used to select features based on the values of feature properties. For example, you could use a basic filter to select all roads that have four or more lanes.

Spatial filters are used to select features based on their geometry. For example, you could use a spatial filter to select all roads that intersect a certain area.

## Basic Filters

Basic filters perform logical tests of feature properties. You can construct complex queries by combining expressions. Expressions use the comparison operators below:

Operator	Meaning
=	Equality
<>	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE	Used for string comparisons. The "%" wildcard represents any sequence of 0 or more characters. The "_" wildcard represents any single character. For example, "LIKE Schmitt%" will search for any names beginning with "Schmitt".

The comparison operators can be used with numeric or string properties, except for the `LIKE` operator, which can only be used with string properties.

Combine or modify expressions with the standard boolean operators `AND`, `OR`, and `NOT`.

## Examples

These examples assume that the feature class you are querying has an integer property named `year` and a string property named `owner`. To select all features newer than 2001, create a filter like this:

```
MgFeatureQueryOptions queryOptions = new MgFeatureQueryOptions();
queryOptions.SetFilter("year > 2001");
```

To select all features built between 2001 and 2004, create a filter like this:

```
queryOptions.SetFilter("year >= 2001 and year <= 2004");
```

To select all features owned by Davis or Davies, create a filter like this:

```
queryOptions.SetFilter("owner LIKE 'Davi%s'");
```

## Spatial Filters

With spatial filters, you can do comparisons using geometric properties. For example, you can select all features that are inside an area on the map, or that intersect an area.

There are two ways of using spatial filters:

- Create a separate spatial filter to apply to the feature source, using the `MgFeatureQueryOptions.SetSpatialFilter()` method.
- Include spatial properties in a basic filter created with the `MgFeatureQueryOptions.SetFilter()` method.

The `MgFeatureQueryOptions.SetSpatialFilter()` method requires an `MgGeometry` object to define the geometry and a spatial operation to compare the feature property and the geometry. The spatial operations are defined in class `MgFeatureSpatialOperations`.

To include spatial properties in a basic filter, define the geometry using WKT format. Use the `GEOMFROMTEXT()` function in the basic filter, along with one of the following spatial operations:

- CONTAINS
- COVEREDBY
- CROSSES
- DISJOINT
- EQUALS
- INTERSECTS
- OVERLAPS

- TOUCHES
- WITHIN
- INSIDE

---

**NOTE** The spatial operations are not case sensitive, so “CONTAINS” and “contains” produce the same result.

---

For example, the following selects features that cross an area:

```
queryOptions.SetFilter(
    "SHPGEOM crosses GEOMFROMTEXT(" + wktGeom + ")" );
```

The same filtering syntax can be used in a layer definition, to create a layer containing only those features that pass the filter. See [Filtering Layers By Geometry](#) on page 51 for details.

---

**NOTE** Not all spatial operations can be used on all features. It depends on the capabilities of the FDO provider that supplies the data. This restriction applies to separate spatial filters and spatial properties that are used in a basic filter.

---

## Creating Geometry Objects From Features

You may want to use an existing feature as part of a spatial query. To retrieve the feature’s geometry and convert it into an appropriate format for a query, perform the following steps:

- Create a query that will select the feature.
- Query the feature class containing the feature using `AcMapLayer.SelectFeatures()` **OR** `MgFeatureService.SelectFeatures()`.
- Obtain the feature from the query using the `MgFeatureReader.ReadNext()` method.
- Get the geometry data from the feature using the `MgFeatureReader.GetGeometry()` method. This data is in AGF binary format.
- Convert the AGF data to an `MgGeometry` object using the `MgAgfReaderWriter.Read()` method.

For example, the following sequence creates an `MgGeometry` object representing the boundaries of zone 1 in the layer named “zones”. It creates an `MgGeometry` object and the WKT representation of that object.

```
MgLayerCollection layers = currentMap.GetLayers();
MgLayerBase layer = layers.GetItem("zones");
string fsId = layer.GetFeatureSourceId();
string className = layer.GetFeatureClassName();
MgFeatureQueryOptions query =
    new MgFeatureQueryOptions();
query.SetFilter("ZONE_ID = 1");
MgResourceIdentifier resId =
    new MgResourceIdentifier(fsId);

MgFeatureReader featureReader =
    fs.SelectFeatures(resId,
        className, query);
if (featureReader.ReadNext())
{
    string geometryName = layer.GetFeatureGeometryName();
    MgByteReader geometryData =
        featureReader.GetGeometry(geometryName);
    MgAgfReaderWriter agfReaderWriter = new MgAgfReaderWriter();
    MgGeometry geometry = agfReaderWriter.Read(geometryData);

    MgWktReaderWriter wktReaderWriter = new MgWktReaderWriter();
    string wkt = wktReaderWriter.Write(geometry);
}
```

The following assumes that another feature class has a geometry property `SHPGEOM`. It uses the WKT string in a query to find features in the other feature class that intersect the zone:

```
MgFeatureQueryOptions queryOpts = new MgFeatureQueryOptions();
queryOpts.SetFilter("SHPGEOM intersects GEOMFROMTEXT(" +
    wkt + ")");
```

## Updating Features

All updates to features in a feature source are performed using an `MgFeatureCommandCollection`. The feature command collection can contain commands to add new features, delete existing features, update existing features, lock features, or unlock features.

To add a new feature, create an `MgInsertFeatures` object. `MgInsertFeatures` has two constructors. One has an `MgPropertyCollection` parameter and is used to insert a single feature. The other has an `MgBatchPropertyCollection` parameter and is used to insert multiple features in a single command.

The `MgPropertyCollection` contains values for the feature properties, including the feature geometry property. Create an empty `MgPropertyCollection`, then add properties using `MgPropertyCollection.Add()`. The different property types, such as `MgGeometryProperty` or `MgInt32Property`, are all derived from the base class `MgProperty`.

Most feature classes will require a geometry property, which is used to display the feature. To create a geometry property, start with an `MgGeometry` object.

Convert the `MgGeometry` to binary AGF format suitable for FDO.

```
MgByteReader geometryStream;  
MgAgfReaderWriter agfReaderWriter = new MgAgfReaderWriter;  
geometryStream = agfReaderWriter.Write(geometry);
```

Create a property collection to contain the feature properties. Add the geometry property. Add any other required properties. The property names are case sensitive and must match the names defined in the feature class.

```
MgPropertyCollection properties = new MgPropertyCollection();  
MgGeometryProperty geomProp;  
geomProp = new MgGeometryProperty( "SHPGEOM", geometryStream);  
properties.Add(geomProp);
```

Create a feature command to insert the feature, and add it to a feature command collection.

```
MgFeatureCommandCollection commands =  
    new MgFeatureCommandCollection();  
MgInsertFeatures insertCommand;  
insertCommand = new MgInsertFeatures(className, properties);  
commands.Add(insertCommand);
```

Call `AcMapLayer.UpdateFeatures()` to add the new feature.

```
mapLayer.UpdateFeatures(commands);
```

Deleting and updating features is similar. The constructors for `MgDeleteFeatures` and `MgUpdateFeatures` require different parameters, but the feature commands can be added to the same `MgFeatureCommandCollection`.

## Edit Sets

There are two different modes for updates to features:

- Edit set
- Direct update

---

**NOTE** Some FDO providers, including the file-based providers SDF, SHP, and raster, do not support edit set mode.

---

Updates made using edit set mode are made to the working copy inside the AutoCAD Map 3D application, but are not committed to the feature source. Edits made using direct update are changed in the feature source immediately.

There are two methods for updating features:

- `AcMapLayer.UpdateFeatures()`
- `MgFeatureService.UpdateFeatures()`

`AcMapLayer.UpdateFeatures()` works directly with features in the map layer. This is the preferred method when working in edit set mode.

`MgFeatureService.UpdateFeatures()` updates the features in the feature source. This method can be used when the feature source is not being displayed in a layer. For example, an AutoCAD Map 3D application could create an SDF file containing result data without ever having to display the data.

To set the update mode for a layer, call `AcMapLayer.SetEditSetMode()`.

To commit changes made in edit set mode, call `AcMapLayer.SaveFeatureChanges()`. This updates the feature source. To discard the changes, call `AcMapLayer.DiscardFeatureChanges()`.

## Creating SDF files

Many geospatial operations require creating new layers, adding features to the layers, and adding the layers to the map. The layers can be temporary or permanent. In AutoCAD Map 3D this is handled using SDF files.

An SDF file contains a single schema that can define multiple feature classes.



To create an SDF file, first define the schema and any feature classes used in the schema. For each feature class, define a list of properties. In most cases, the properties should include an identity property and a geometry property, as well as any other properties.

Use the FDO API to create the schema.

The following creates a simple schema with an autogenerated ID property, a string property, and a geometry property.

```
// Create the feature class definition

FeatureClass classDefinition =
    new FeatureClass("newClass", "class description");

// Add an autogenerated identity property

DataPropertyDefinition idProp =
    new DataPropertyDefinition("id", "property desc");
idProp.DataType = DataType.DataType_Int32;
idProp.IsAutoGenerated = true;
classDefinition.Properties.Add(idProp);
classDefinition.IdentityProperties.Add(idProp);

// Add a name property

DataPropertyDefinition nameProp =
    new DataPropertyDefinition("Name", "property desc");
nameProp.DataType = DataType.DataType_String;
classDefinition.Properties.Add(nameProp);

// Add a geometry property

GeometricPropertyDefinition geomProp =
    new GeometricPropertyDefinition("SHPGEOM", "property desc");
geomProp.GeometryTypes = MgGeometryType.Polygon;
classDefinition.Properties.Add(geomProp);
classDefinition.GeometryProperty = geomProp;

// Create the schema and add the class definition

FeatureSchema schema =
    new FeatureSchema("SHP_Schema", "Schema description");
schema.Classes.Add(classDefinition);
```

Once the schema has been created, use the FDO API to create the SDF file, set the spatial context, and apply the schema. The following is modified from the utility classes in the sample applications.

```

// Set connection properties

IConnectionPropertyDictionary connProperties =
    conn.ConnectionInfo.ConnectionProperties;
connProperties.SetProperty("File", sdfPath);
connProperties.SetProperty("ReadOnly", false.ToString());

// Create data store, ie. SDF file

ICreateDataStore createdSCmd =
    conn.CreateCommand(CommandType.CommandType_CreateDataStore)
    as ICreateDataStore;
createdSCmd.DataStoreProperties.SetProperty("File", sdfPath);
createdSCmd.Execute();

try
{
    // Open the connection
    ConnectionState connState = conn.Open();
    int retryRound = 0;
    while (++retryRound < 5 // Try 5 times
        && (connState == ConnectionState.ConnectionState_Pending
            || connState == ConnectionState.ConnectionState_Busy))
    {
        connState = conn.Open();
    }
    if (retryRound >= 5)
    {
        throw new
            InvalidOperationException("Failed to connect to file "
                + sdfPath);
    }

    // Create spatial context, including coordinate system

    AcMapMap currentMap = AcMapMap.GetCurrentMap();
    string mapSRS = currentMap.GetMapSRS();

    ICreateSpatialContext createSCCmd =
        conn.CreateCommand(CommandType.CommandType_CreateSpatialContext)
        as ICreateSpatialContext;
    createSCCmd.CoordinateSystemWkt = mapSRS;
    createSCCmd.Name = "";
}

```

```

createSCCmd.CoordinateSystem = mapSRS;
createSCCmd.Extent = new byte[] { 0, 0, 0, 0 };
createSCCmd.Description = "Description";
createSCCmd.XYTolerance = 0.0;
createSCCmd.ZTolerance = 0.0;
createSCCmd.Execute();

// Apply schema
if (schema.ElementState !=
    SchemaElementState.SchemaElementState_Unchanged)
{
    IApplySchema applySchemaCmd =
        conn.CreateCommand(CommandType.CommandType_ApplySchema)
        as IApplySchema;
    applySchemaCmd.FeatureSchema = schema;
    applySchemaCmd.Execute();
}
}
catch (System.Exception e)
{
    throw e; // Or add exception handling
}
finally
{
    // Close connection
    if (conn != null)
    {
        conn.Close();
    }
}
}

```

# Maps and Layers

# 4

## Overview

A map is composed of layers, where each layer represents data from a feature source.

AutoCAD Map 3D and Autodesk MapGuide differ in some of the ways they handle map layers. Because MapGuide is a web-based mapping product, it must coordinate layer display based on information from the client and the server. AutoCAD Map 3D can handle all layer display directly from the application.

---

**NOTE** The Geospatial Platform API deals strictly with layers from FDO feature sources. If a map in AutoCAD Map 3D includes other layers, they must be handled using the appropriate methods from the .NET API.

---

A layer (`AcMapLayer` object), has a `LayerDefinition` property that points to a layer definition in the resource repository. The layer definition content is XML that conforms to the `LayerDefinition.xsd` schema. Among other elements, it contains a `ResourceId` element that identifies the feature source for the layer.

## Basic Layer Properties

A map contains one or more layers (`AcMapLayer` objects) that are rendered to create a composite image. The `AcMapLayer` class, which applies only to AutoCAD Map 3D, is derived from `MgLayerBase`, which is part of the common Geospatial Platform API shared between AutoCAD Map 3D and MapGuide.

Each layer has properties that determine how it displays in the map and map legend. Some of the properties are:

- **Layer name:** A unique identifier
- **Visibility:** whether the layer should be displayed in the map. Note that actual visibility is dependent on more than just the visibility setting for a layer. See [Layer Visibility](#) on page 43 for further details.
- **Selectable:** Whether features in the layer are selectable. This only applies to layers containing feature data. It does not apply to layers containing raster data.

`AcMapMap.GetLayers()` returns an `MgLayerCollection` object that contains all the layers in the map. `MgLayerCollection.GetItem()` returns an individual `MgLayerBase` object, by either index number in the collection or layer name.

Layers in the collection are sorted by drawing order, with the top layers at the beginning of the collection.

## Layer Groups

Layers can be optionally grouped into layer groups. Layers in the same group are displayed together in the Display Manager.

The visibility for all layers in a group can be set at the group level. If the group visibility is turned off then none of the layers in the group will be visible, regardless of their individual visibility settings. If the group visibility is turned on, then individual layers within the group can be made visible or not visible separately.

Layer groups can be nested so a group can contain other groups. This provides a finer level of control for handling layer visibility or for grouping layers into legend groups.

`AcMapMap.GetLayerGroups()` returns an `MgLayerGroupCollection` object that contains all the top-level layer groups in the map.

Each layer group in a map must have a unique name, even if it is nested within another group.

## Layer Visibility

Whether a layer is visible in a given map depends on three criteria:

- The visibility setting for the layer
- The visibility settings for any groups that contain the layer
- The map view scale and the layer definition for that view scale

In order for a layer to be visible, its layer visibility must be on, the visibility for any group containing the layer must be on, and the layer must have a style setting defined for the current map view scale.

### Example: Actual Visibility

For example, assume that there is a layer named Roads that is part of the layer group Transportation. The layer has view style defined for the scale ranges 0–10000 and 10000–24000.

The following table shows some possible settings of the various visibility and view scale settings, and their effect on the actual layer visibility.

Layer Visibility	Group Visibility	View Scale	Actual Visibility
On	On	10000	On
On	On	25000	Off
On	Off	10000	Off
Off	On	10000	Off

## Manipulating Layers

Modifying basic layer properties and changing layer visibility settings can be done directly by setting properties like `AcMapLayer.Name`, `AcMapLayer.Selectable`, and `AcMapLayer.Visible`. More complex manipulation requires modifying layer resources in the resource repository.

## Changing Visibility

To query the actual layer visibility, call `AcMapLayer.IsVisible()`. There is no method to set actual visibility because it depends on other visibility settings.

To query the visibility setting for a layer, call `AcMapLayer.GetVisible()`. To change the visibility setting for a layer, call `AcMapLayer.SetVisible()`.

To query the visibility setting for a layer group, call `AcMapLayerGroup.GetVisible()`. To change the visibility setting for a layer group, use `AcMapLayerGroup.SetVisible()`.

To change the layer visibility for a given view scale, modify the layer resource and save it back to the repository.

## Layer Definition

The feature source and styling for a layer are set using a `LayerDefinition`, stored in the resource repository. The layer definition conforms to the `LayerDefinition.xsd` schema. Raster layers and vector layers have different styling requirements and capabilities.

For both raster and vector data, a single layer definition can contain multiple scale ranges, which define the styling for particular view scales. For example, a vector layer could have the following scale ranges:

- 0 to 10,000
- 10,000 to 500,000
- 500,000 to infinity

Features in the layer could be styled differently or be hidden completely for the different scale ranges.

For a vector layer, each scale range can contain styling information for the following:

- `AreaTypeStyle`, used for polygons
- `LineTypeStyle`, used for lines and curves
- `PointTypeStyle`, used for points



The layer definition can be any valid XML that conforms to the schema. To help generate the XML, the AutoCAD Map 3D samples use classes generated from `LayerDefinition.xsd` by `xsd.exe`.

For example, to create a vector layer definition that contains 2 scale ranges, do the following:

- Create the two scale ranges. For each, specify the minimum and maximum scale for the range. Define the necessary elements, like fill and stroke settings.
- Create the vector layer definition. Add the scale ranges.
- Set the feature name to the schema/feature class in the feature source.
- Set the resource id to point to the feature source containing the layer data.
- Set the geometry to the geometry property of the feature class.
- Create a layer definition that contains the vector layer definition.
- Store the layer definition in the resource repository.
- Create a new layer that references the layer definition. Add the layer to the layer collection for the current map.

The example code that follows uses the following namespace declarations:

- `using System.IO` (for the `StringWriter` class)
- `using System.Text` (for the `Encoding` class)
- `using System.Xml.Serialization` (for XML serialization routines)
- `using OSGeo.MapGuide.Schema.LayerDefinition` (generated by `xsd.exe`. Code for this class is available with the AutoCAD Map 3D samples.)

The following assumes that the feature source contains only polygons. Feature sources containing line or point data would require styling for those feature types as well.

```

// Create the first scale range

VectorScaleRangeType range = new VectorScaleRangeType();

AreaRule areaRule = new AreaRule();

AreaSymbolizationFillType areaSymFillType =
    new AreaSymbolizationFillType();

FillType fillType = new FillType();
fillType.FillPattern = "Solid";
fillType.ForegroundColor = "9900ffaa";
fillType.BackgroundColor = "FF000000";

StrokeType strokeType = new StrokeType();
strokeType.LineStyle = "Solid";
strokeType.Thickness = "0.0";
strokeType.Color = "FF000000";
strokeType.Unit = LengthUnitType.Centimeters;

areaSymFillType.Fill = fillType;
areaSymFillType.Stroke = strokeType;

areaRule.Item = areaSymFillType;
areaRule.LegendLabel = "";

object[] items = new object[1];
items[0] = new AreaRule[] { areaRule };
range.Items = items;
range.MinScale = 0.0;
range.MaxScale = 10000000.0;
range.MinScaleSpecified = true;
range.MaxScaleSpecified = true;

// Second scale range

VectorScaleRangeType range2 = new VectorScaleRangeType();

AreaRule areaRule2 = new AreaRule();

AreaSymbolizationFillType areaSymFillType2 =
    new AreaSymbolizationFillType();

```

```

FillType fillType2 = new FillType();
fillType2.FillPattern = "Solid";
fillType2.ForegroundColor = "ffffff00";
fillType2.BackgroundColor = "FF000000";

StrokeType strokeType2 = new StrokeType();
strokeType2.LineStyle = "Solid";
strokeType2.Thickness = "0.0";
strokeType2.Color = "FF000000";
strokeType2.Unit = LengthUnitType.Centimeters;

areaSymFillType2.Fill = fillType2;
areaSymFillType2.Stroke = strokeType2;

areaRule2.Item = areaSymFillType2;
areaRule2.LegendLabel = "";

object[] items2 = new object[1];
items2[0] = new AreaRule[] { areaRule2 };
range2.Items = items2;
range2.MinScale = 10000000.0;
range2.MaxScale = Double.MaxValue;
range2.MinScaleSpecified = true;
range2.MaxScaleSpecified = true;

// Now create a vector layer definition and add the scale ranges.

VectorLayerDefinitionType vectorLayerDef =
    new VectorLayerDefinitionType();
vectorLayerDef.VectorScaleRange = new VectorScaleRangeType[2];
vectorLayerDef.VectorScaleRange.SetValue(range, 0);
vectorLayerDef.VectorScaleRange.SetValue(range2, 1);

// Set the resource id to point to the feature source for
// the layer. Set the feature name to a feature class in
// the feature source. Set the geometry property to the
// default geometry property for the feature class.

vectorLayerDef.ResourceId = "Library://Data/SAMPLE.FeatureSource";
vectorLayerDef.FeatureName = "Schema1:FeatureClass1";
vectorLayerDef.Geometry = "Geometry";

// Create a layer definition containing the vector layer

```

```

// definition.

LayerDefinition layerDef = new LayerDefinition();
layerDef.Item = vectorLayerDef;
layerDef.version = "1.0.0";

// Convert to an XML string.

using (StringWriter writer = new StringWriter())
{
    XmlSerializer xs = new XmlSerializer(typeof(LayerDefinition));
    xs.Serialize(writer, layerDef);

    // Convert to UTF-8.

    byte[] unicodeBytes =
        Encoding.Unicode.GetBytes(writer.ToString());
    byte[] utf8Bytes =
        Encoding.Convert(Encoding.Unicode, Encoding.UTF8,
            unicodeBytes);

    MgByteSource source =
        new MgByteSource(utf8Bytes, utf8Bytes.Length);

    // Update the resource in Map.

    MgResourceService rs =
        AcMapServiceFactory.GetService(MgServiceType.ResourceService)
        as MgResourceService;

    // Store the layer definition in the repository.

    MgResourceIdentifier newLayerDefId =
        new MgResourceIdentifier(
            "Library://myNewLayer.LayerDefinition");
    rs.SetResource(newLayerDefId, source.GetReader(), null);

    // Create a new layer that references the layer definition.
    // Add this to the layers in the current map.

    AcMapMap currentMap = AcMapMap.GetCurrentMap();
    MgLayerCollection layers = currentMap.GetLayers();
    AcMapLayer layer;

```

```
layer = new AcMapLayer(newLayerDefId, rs);  
layer.Name = "testlayer";  
layers.Add(layer);  
}
```

This creates the following XML:

```

<?xml version="1.0" encoding="utf-16"?>
<LayerDefinition
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  version="1.0.0">
  <VectorLayerDefinition>
    <ResourceId>Library://Data/SAMPLE.FeatureSource</ResourceId>
    <FeatureName>Schemal:FeatureClass1</FeatureName>
    <FeatureNameType>FeatureClass</FeatureNameType>
    <Geometry>Geometry</Geometry>
    <VectorScaleRange>
      <MinScale>0</MinScale>
      <MaxScale>10000000</MaxScale>
      <AreaTypeStyle>
        <AreaRule>
          <LegendLabel />
          <AreaSymbolization2D>
            <Fill>
              <FillPattern>Solid</FillPattern>
              <ForegroundColor>9900ffaa</ForegroundColor>
              <BackgroundColor>FF000000</BackgroundColor>
            </Fill>
            <Stroke>
              <LineStyle>Solid</LineStyle>
              <Thickness>0.0</Thickness>
              <Color>FF000000</Color>
              <Unit>Centimeters</Unit>
            </Stroke>
          </AreaSymbolization2D>
        </AreaRule>
      </AreaTypeStyle>
    </VectorScaleRange>
    <VectorScaleRange>
      <MinScale>10000000</MinScale>
      <MaxScale>1.7976931348623157E+308</MaxScale>
      <AreaTypeStyle>
        <AreaRule>
          <LegendLabel />
          <AreaSymbolization2D>
            <Fill>
              <FillPattern>Solid</FillPattern>
              <ForegroundColor>ffffff00</ForegroundColor>
              <BackgroundColor>FF000000</BackgroundColor>
            </Fill>
            <Stroke>
              <LineStyle>Solid</LineStyle>
              <Thickness>0.0</Thickness>
              <Color>FF000000</Color>
              <Unit>Centimeters</Unit>
            </Stroke>
          </AreaSymbolization2D>
        </AreaRule>
      </AreaTypeStyle>
    </VectorScaleRange>
  </VectorLayerDefinition>
</LayerDefinition>

```

```

        </Fill>
        <Stroke>
            <LineStyle>Solid</LineStyle>
            <Thickness>0.0</Thickness>
            <Color>FF000000</Color>
            <Unit>Centimeters</Unit>
        </Stroke>
    </AreaSymbolization2D>
</AreaRule>
</AreaTypeStyle>
</VectorScaleRange>
</VectorLayerDefinition>
</LayerDefinition>

```

## Layers With Joined Feature Sources

To create a layer definition for a layer that uses a feature source containing a join, create the layer definition as described in [Layer Definition](#) on page 44. Set the `<FeatureName>` element to the join name as defined in the feature source. Set the `<FeatureNameType>` element to `NamedExtension`. For example, the following portion of a layer definition uses the join named “Parcels\_Joins1”.

```

<?xml version="1.0" encoding="utf-16"?>
<LayerDefinition
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    version="1.0.0">
    <VectorLayerDefinition>
        <ResourceId>Library://Data/SAMPLE.FeatureSource</ResourceId>
        <FeatureName>Parcels_Joins1</FeatureName>
        <FeatureNameType>NamedExtension</FeatureNameType>
    </VectorLayerDefinition>
</LayerDefinition>

```

See [Joins](#) on page 23 for details about the feature source definition.

## Filtering Layers By Geometry

The layer definition for a vector layer can include a `<Filter>` element to select only certain features. The filter is a string that contains a valid FDO selection expression. See [Selecting Using the API](#) on page 30 for a description of filters.

For example, assume the following:

- `geomPoly` is an `MgPolygon` representing the area of interest.
- `geometry` is the default geometry property for a feature class.

The following creates a filter to select only those features that intersect the polygon.

```
MgWktReaderWriter rw = new MgWktReaderWriter();
string wktGeom = rw.Write(geomPoly);
string filter = "geometry intersects GeomFromText(" +
    wktGeom + ")" );
```

To modify the sample in [Layer Definition](#) on page 44 to use the filter, set the `<Filter>` element as follows:

```
vectorLayerDef.Filter = filter;
```

## Modifying Layer Style

To modify the styling for a layer, retrieve the layer definition from the repository, make changes to it, store it back to the repository, and refresh the layer.

To retrieve the layer definition, get its resource identifier from the map layer. Get the resource content and deserialize it using an `XmlSerializer` object.



```

AcMapMap currentMap = AcMapMap.GetCurrentMap();
MgLayerCollection layers = currentMap.GetLayers();
MgResourceService rs =
    AcMapServiceFactory.GetService(MgServiceType.ResourceService)
    as MgResourceService;

// Get the layer definition. Convert it to an XML string
// stored in a byte reader.

AcMapLayer layer1 = layers.GetItem(layerName) as AcMapLayer;
MgResourceIdentifier layerDefId = layer1.GetLayerDefinition();
MgByteReader layerReader = rs.GetResourceContent(layerDefId);

// Deserialize it to create a LayerDefinition object.

XmlSerializer layerDefSerializer = new
    XmlSerializer(typeof(LayerDefinition));
LayerDefinition xmlLayerDef =
    layerDefSerializer.Deserialize(new
        StringReader(layerReader.ToString()))
    as LayerDefinition;

```

Updating the `LayerDefinition` object is similar to creating a new one. Find the correct elements and set their values. For example, the following changes the foreground color for the area rule in the first scale range, then it updates the layer definition in the resource repository and refreshes the layer.

```

// Should be a VectorLayerDefinitionType

if (xmlLayerDef.Item is VectorLayerDefinitionType)
{
    VectorLayerDefinitionType xmlVectorLayerDef =
        xmlLayerDef.Item as VectorLayerDefinitionType;

    // Check the rules for the first scale range.
    // Only update rules if they are the correct type.

    object[] items = xmlVectorLayerDef.VectorScaleRange[0].Items;
    foreach (object item in items)
    {
        Type type = item.GetType();

        if (type.Name == typeof(AreaRule[]).Name)
        {
            AreaRule[] areaRules = item as AreaRule[];

            // OK, got the correct area rule. Set the foreground color.

            if (areaRules.Length > 0)
            {
                areaRules[0].Item.Fill.ForegroundColor = newColor;
            }
        }
    }

    // Now convert it back to an XML string and update the
    // resource.

    using (StringWriter writer = new StringWriter())
    {
        // Get the xml string
        layerDefSerializer.Serialize(writer, xmlLayerDef);

        // Create the resource
        byte[] unicodeBytes =
            Encoding.Unicode.GetBytes(writer.ToString());
        byte[] utf8Bytes =
            Encoding.Convert(Encoding.Unicode,
                Encoding.UTF8, unicodeBytes);
    }
}

```

```
MgByteSource source =  
    new MgByteSource(utf8Bytes, utf8Bytes.Length);  
  
// Update the resource in Map  
rs.SetResource(layer1.LayerDefinition,  
    source.GetReader(), null);  
  
layer1.ForceRefresh();  
}  
}
```



# Geometry

# 5

## Overview

The AutoCAD Map 3D Geospatial Platform API includes a complete set of classes for working with geometry. This includes the following:

- Describing simple and complex geometric objects
- Performing spatial comparisons between objects
- Creating new objects based on the intersection, difference, or union of existing objects
- Converting between different coordinate systems
- Creating buffers around objects

## Geometry Objects

`MgGeometry` is the base class for all the geometry types. The simple geometry types are:

- `MgPoint` — a single point
- `MgLineString` — a series of connected line segments
- `MgCurveString` — a series of connected curve segments
- `MgPolygon` — a polygon with sides formed from line segments
- `MgCurvePolygon` — a polygon with sides formed from curve segments

The curve segments are circular arcs, defined by a start point, an end point, and a control point.

Complex types are formed by aggregating simple types. The complex types are:

- `MgMultiPoint` — a group of points
- `MgMultiLineString` — a group of line strings
- `MgMultiCurveString` — a group of curve strings
- `MgMultiPolygon` — a group of polygons
- `MgMultiCurvePolygon` — a group of curve polygons
- `MgMultiGeometry` — a group of simple geometry objects of any type

## Comparing Geometry Objects

The `MgGeometry` class contains methods for comparing different geometry objects. These are similar to the spatial filters described in [Selecting Using the API](#) on page 30. Methods to test spatial relationships include:

- `Contains()`
- `Crosses()`
- `Disjoint()`
- `Equals()`
- `Intersects()`
- `Overlaps()`
- `Touches()`
- `Within()`

For example, if you have an `MgLineString` object `line` and an `MgPolygon` object `polygon`, you can test if the line crosses the polygon with a call to

```
line.Crosses(polygon)
```

Methods to create new geometry objects from the point set of two other geometries include:

- `Difference()`
- `Intersection()`
- `SymmetricDifference()`
- `Union()`

Complete details are in the Geometry module of the Geospatial Platform API reference, under Spatial Relationships.

## Coordinate Systems

A single map will often combine data from different sources, and the different sources may use different coordinate systems. The map has its own coordinate system, and any feature sources used in the map may have different coordinate systems. It is important for display and analysis that all locations are transformed to the same coordinate system.

---

**NOTE** A coordinate system can also be called a spatial reference system (SRS) or a coordinate reference system (CRS). This guide uses the abbreviation SRS.

---

AutoCAD Map 3D supports three different types of coordinate system:

- Arbitrary X-Y
- Geographic, or latitude/longitude
- Projected

An `MgCoordinateSystem` object represents a coordinate system.

---

**NOTE** You cannot transform between arbitrary X-Y coordinates and either geographic or projected coordinates.

---

To create an `MgCoordinateSystem` object from an `AcMapMap` object,

- Get the WKT representation of the map coordinate system, using `AcMapMap.GetMapSRS()`.

- Create an `MgCoordinateSystem` object, using  
`MgCoordinateSystemFactory.Create()`.

To create an `MgCoordinateSystem` object from a map layer,

- Get the feature source for the layer.
- Get the active spatial context for the feature source.
- Convert the spatial context to a WKT.
- Create an `MgCoordinateSystem` object from the WKT.

To transform geometry from one coordinate system to another, create an `MgCoordinateSystemTransform` object using the two coordinate systems. Apply this transform to the `MgGeometry` object.

For example, if you have geometry representing a feature on a layer that uses one coordinate system, and you want to compare it to a feature on another layer that uses a different coordinate system, perform the following steps:

```
string featureSource1 = layer1.GetFeatureSourceId();
MgSpatialContextsReader contexts1 =
    featureService.GetSpatialContexts(
        featureSource1, true);
contexts1.ReadNext();
string srs1 = contexts1.GetCoordinateSystemWkt();

string featureSource2 = layer2.GetFeatureSourceId();
MgSpatialContextsReader contexts2 =
    featureService.GetSpatialContexts(
        featureSource2, true);
contexts2.ReadNext();
string srs2 = contexts2.GetCoordinateSystemWkt();

MgCoordinateSystemFactory coordsysFactory =
    new MgCoordinateSystemFactory();
MgCoordinateSystemTransform xform =
    coordsysFactory.GetTransform(srs1, srs2);
MgGeometry geometry1xform = geometry1.Transform(xform);
```



## Measuring Distance

Measuring distance in geographic or projected coordinate systems requires great circle calculations. Both `MgGeometry.Buffer()` and `MgGeometry.Distance()` accept a measurement parameter that defines the great circle to be used. If the measurement parameter is null, the calculation is done using a linear algorithm.

Create the measurement parameter, an `MgCoordinateSystemMeasure` object, from the `MgCoordinateSystem` object.

Distance is calculated in the units of the SRS. `MgCoordinateSystem` includes two methods, `ConvertCoordinateSystemUnitsToMeters()` and `ConvertMetersToCoordinateSystemUnits()`, to convert to and from linear distances.

For example, to calculate the distance between two `MgGeometry` objects `a` and `b`, using the coordinate system `srs`, perform the following steps:

```
MgCoordinateSystemMeasure measure = srs.GetMeasure();  
double distInMapUnits = a.Distance(b, measure);  
double distInMeters = srs.ConvertCoordinateSystemUnitsToMeters(  
    distInMapUnits);
```

Another way to calculate the distance is to use

`MgCoordinateSystemMeasure.GetDistance()`, as in the following:

```
distInMapUnits = measure.GetDistance(a, b);
```

## Creating a Buffer

To create a buffer around a feature, use the `MgGeometry.Buffer()` method. This returns an `MgGeometry` object that you can use for further analysis. For example, you could display the buffer by creating a feature in a temporary feature source and adding a new layer to the map. You could also use the buffer geometry as part of a spatial filter. For example, you might want to find all the features within the buffer zone that match certain criteria, or you might want to find all roads that cross the buffer zone.

To create a buffer, get the geometry of the feature to be buffered. If the feature is being processed in an `MgFeatureReader` as part of a selection, this requires getting the geometry data from the feature reader and converting it to an `MgGeometry` object. For example:

```
MgByteReader geometryData =
    featureReader.GetGeometry(geometryName);
MgGeometry featureGeometry = agfReaderWriter.Read(geometryData);
```

If the buffer is to be calculated using coordinate system units, create an `MgCoordinateSystemMeasure` object from the coordinate system for the map. For example:

```
string mapWktSrs = currentMap.GetMapSRS();
MgCoordinateSystemFactory coordSysFactory =
    new MgCoordinateSystemFactory();
MgCoordinateSystem srs = coordSysFactory.Create(mapWktSrs);
MgCoordinateSystemMeasure srsMeasure = srs.GetMeasure();
```

Use the coordinate system measure to determine the buffer size in the coordinate system, and create the buffer object from the geometry to be buffered.

```
double srsDist =
    srs.ConvertMetersToCoordinateSystemUnits(bufferDist);
MgGeometry bufferGeometry =
    featureGeometry.Buffer(srsDist, srsMeasure);
```

To display the buffer in the map, perform the following steps:

- Create a feature source for the buffer. This may require creating a new file for the feature source. See [Creating SDF files](#) on page 36.
- Create a layer that references the feature source. Add it to the map and make it visible.
- Create a new feature using the buffer geometry and insert it into the feature source.

For example, the following code assumes that the layer `resLayer` has been created using a feature source with a geometry property of `Geometry`. It adds a new feature using the buffer geometry.

```

MgPropertyCollection properties = new MgPropertyCollection();
properties.Add(new MgGeometryProperty("Geometry",
    agfReaderWriter.Write(bufferGeometry)));

MgFeatureCommandCollection commands =
    new MgFeatureCommandCollection();
commands.Add(new MgInsertFeatures(resFeatureClassName,
    properties));

resLayer.UpdateFeatures(commands);

```

To use the buffer as part of a query, create a spatial filter using the buffer geometry, and use this in a call to `AcMapLayer.SelectFeatures()`. For example, the following code uses a basic filter and a spatial filter to select parcels inside the buffer area that are of type “MFG”. You can use the `MgFeatureReader` to perform tasks like generating a report of the parcels, or creating a new layer that puts point markers on each parcel.

```

MgFeatureQueryOptions queryOptions = new MgFeatureQueryOptions();
queryOptions.SetFilter("PARCELTYPE = 'MFG'");
queryOptions.SetSpatialFilter('SHPGEOM', bufferGeometry,
    MgFeatureSpatialOperations.Inside);

MgResourceIdentifier featureResId = new MgResourceIdentifier(
    "Library://Data/Parcels.FeatureSource");
MgFeatureReader featureReader =
    layer.SelectFeatures(queryOptions);

```



# Index

= 31  
> 31  
>= 31

## A

AcMapApiMgd assembly 3  
AcMapFeatureService.SelectFeatures() 29  
AcMapLayer objects 41  
AcMapLayer.DiscardFeatureChanges() 36  
AcMapLayer.SaveFeatureChanges() 36  
AcMapLayer.SelectFeatures() 33  
AcMapLayer.SetEditSetMode() 36  
AcMapLayer.UpdateFeatures() 36  
AcMapMap.GetCurrentMap() 17  
AcMapMap.GetFeatureSelection() 28  
Add() method of MgLayerCollection 26  
Add() method of  
    MgPropertyCollection 35  
adding features 34  
AGF geometry format 29  
ApplicationDefinition resource type 6,  
    10  
arbitrary X-Y coordinates 59  
Autodesk MapGuide Studio 9

## B

basic selection filters 30–31  
buffer, creating 61  
buffers 61

## C

ConfigurationDocument element, in  
    feature source definition 19  
CONTAINS 32  
coordinate reference system 59  
coordinate systems 59–60  
    transforming between 60  
COVEREDBY 32

creating geometry from feature 33  
creating SDF files 36  
CROSSES 32  
CRS 59

## D

deleting features 34  
direct update 36  
DiscardFeatureChanges() method of  
    AcMapLayer 36  
DISJOINT 32  
distance, measuring 61  
Document Object Model 9  
DOM 9  
drawing order 42  
DrawingSource resource type 6, 10

## E

edit sets 36  
encoding, UTF-8 for Resource Service 19  
EQUALS 32

## F

FDO 17, 19  
    raster provider 19  
feature classes 17, 26  
    creating layers from 26  
Feature Data Objects 17  
Feature Service 17  
feature source 17–18  
    XML representation 18  
feature source schema 17  
FeatureName element in LayerDefinition  
    schema 26  
features 28, 33–34, 36  
    adding 34  
    deleting 34  
    getting geometry from 33

- selecting 28
- updating 34, 36
- FeatureSource resource type 5
- FeatureSource.xsd 18
- Folder resource type 6, 10

## G

- GenerateFilter() method of
  - MgSelectionBase 29
- geographic coordinates 59
- geometry 58
  - comparing spatial relationships 58
- geometry formats 29
- geometry types 57
- GEOMFROMTEXT() 32
- GetClassDefinition() method of
  - MgFeatureService 28
- GetClasses() method of
  - MgFeatureService 26
- GetFeatureSelection() method of
  - AcMapMap 28
- GetGeometry() method of
  - MgFeatureReader 33
- GetIdentityProperties() method of
  - MgClassDefinition 28
- GetLayers() method of
  - MgSelectionBase 28
- GetSchemas() method of
  - MgFeatureService 26
- great circle calculation, in measuring distance 61

## I

- INSIDE 32
- INTERSECTS 32

## L

- latitude/longitude coordinates 59
- layer groups 42
- layer name 42
- layer properties 41
- layer visibility 43

- layer visibility, and layer groups 42
- LayerDefinition resource type 5
- LayerDefinition.xsd schema 26
- layerdefinitionfactory.php 9
- layers 26, 36
  - creating from feature classes 26
  - creating new with SDF file 36
- Library repository 5
- LIKE 31
- LoadProcedure resource type 6, 10

## M

- Map 3D 10
  - and MapGuide, differences 10
- map layer, and feature source 17
- Map resource type 6, 10
- MapDefinition resource type 6, 10
- MapGuide 10
  - and Map 3D, differences 10
- MapGuide Studio 9
- measuring distance 61
- MgAgfReaderWriter 30
- MgAgfReaderWriter.Read() 30, 33
- MgAgfReaderWriter.Write() 30
- MgByteReader 8
- MgByteSink 8
- MgByteSource 8
- MgClassDefinition.GetIdentityProperties() 28
- MgCoordinateSystem 59
- MgCoordinateSystemTransform 60
- MgCurvePolygon 57
- MgCurveString 57
- MgFeatureCommandCollection 34
- MgFeatureQueryOptions 29–30
- MgFeatureQueryOptions.SetFilter() 32
- MgFeatureQueryOptions.SetSpatialFilter() 32
- MgFeatureReader 29
- MgFeatureReader.GetGeometry() 33
- MgFeatureReader.ReadNext() 29, 33
- MgFeatureService.GetClassDefinition() 28
- MgFeatureService.GetClasses() 26
- MgFeatureService.GetSchemas() 26
- MgFeatureService.SelectFeatures() 33
- MgFeatureService.UpdateFeatures() 36

- MgGeometry 29, 33, 57
  - creating from feature 33
- MgInsertFeatures 35
- MgLayerCollection 42
  - GetItem() 42
- MgLayerCollection object 42
- MgLayerCollection.Add() 26
- MgLineString 57
- MgMap 42
  - GetLayers() 42
- MgMultiCurvePolygon 58
- MgMultiCurveString 58
- MgMultiGeometry 58
- MgMultiLineString 58
- MgMultiPoint 58
- MgMultiPolygon 58
- MgPoint 57
- MgPolygon 57
- MgPropertyCollection.Add() 35
- MgResourceService.SetResource() 8, 18
- MgResourceType 5
- MgSelection 30
- MgSelectionBase 28
- MgSelectionBase.GenerateFilter() 29
- MgSelectionBase.GetLayers() 28
- MgWktReaderWriter 30
- MgWktReaderWriter.Read() 30
- MgWktReaderWriter.Write() 30

## O

OVERLAPS 32

## P

path, in repository 18  
 PrintLayout resource type 6, 10  
 projected coordinates 59  
 properties, of layers 41

## R

raster provider 19  
 Read() method of  
     MgAgfReaderWriter 30, 33

Read() method of  
     MgWktReaderWriter 30  
 ReadNext() method of  
     MgFeatureReader 29, 33  
 repository 5  
 repository name 7  
 repository path 7, 18  
 repository type 6  
 resource data 8  
 resource dependencies 7  
 resource headers, in Map 3D 10  
 resource name 7  
 resource paths, in DWG file 10  
 resource schemas 8  
 resource service 7  
 Resource Service 5, 19  
     UTF-8 encoding 19  
 resource type 7  
 resource types 5  
 resources 5

## S

SaveFeatureChanges() method of  
     AcMapLayer 36  
 schema 17  
     feature source 17  
 schemas, resource 8  
 SDF files 36  
     creating 36  
 SelectFeatures() method of  
     AcMapFeatureService 29  
 SelectFeatures() method of  
     AcMapLayer 33  
 SelectFeatures() method of  
     MgFeatureService 33  
 selecting 30  
     with the Platform API 30  
 selecting features 28  
 selection filters 30–32  
     basic 31  
     spatial 32  
 Selection resource type 6, 10  
 Session repositories 5  
 Session repositories, in Map 3D 10

- SetEditSetMode() method of
  - AcMapLayer 36
- SetFilter() method of
  - MgFeatureQueryOptions 32
- SetResource() method, in
  - MgResourceService 18
- SetSpatialFilter() method of
  - MgFeatureQueryOptions 32
- spatial filters 32
- spatial operators 32
- spatial reference system 59
- spatial relationships, between geometry
  - objects 58
- spatial selection filters 30
- SRS 59
- Studio 9
- SymbolDefinition resource type 6
- SymbolLibrary resource type 6, 10

## T

- templates, for XML resources 9
- TOUCHES 32
- transforming coordinate systems 60

## U

- update modes 36

- UpdateFeatures() method of
  - AcMapLayer 36
- UpdateFeatures() method of
  - MgFeatureService 36
- updating features 34
- UTF-8 encoding, for Resource Service 19

## V

- visibility 43
  - rules 43

## W

- WebLayout resource type 6, 10
- Well Known Text 29
- WITHIN 32
- WKT (Well Known Text) 29
- Write() method of
  - MgAgfReaderWriter 30
- Write() method of
  - MgWktReaderWriter 30

## X

- XML templates 9
- XML, for feature source 18
- xsd.exe 9