AutoCAD® Map 3D 2013 Platform API Training

Autodesk

Introduction to the FDO API

Autodesk

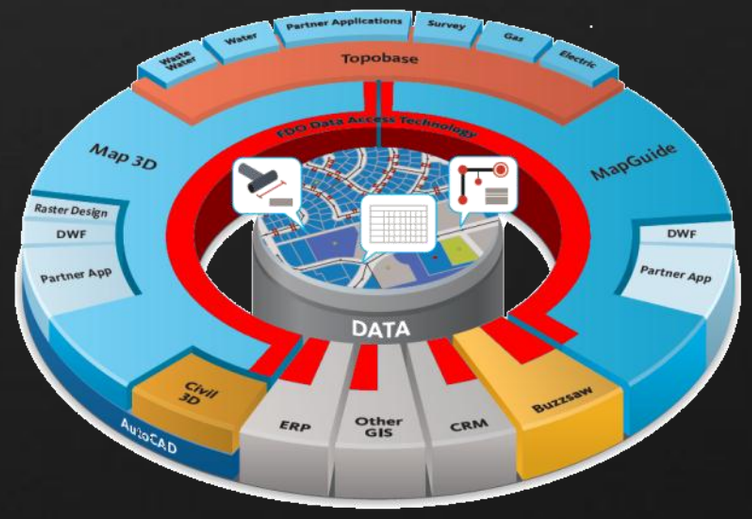
Contents

- FDO API Overview
- FDO Connection
- Commands
- Schema Management
- Creating SDF

FDO Data Access Technology

Work seamlessly with geospatial data

- Rich, consistent access to geospatial data
- Focus on solving business problems
- Embrace and extend open standards
- Platform and technology neutral
- Extensible

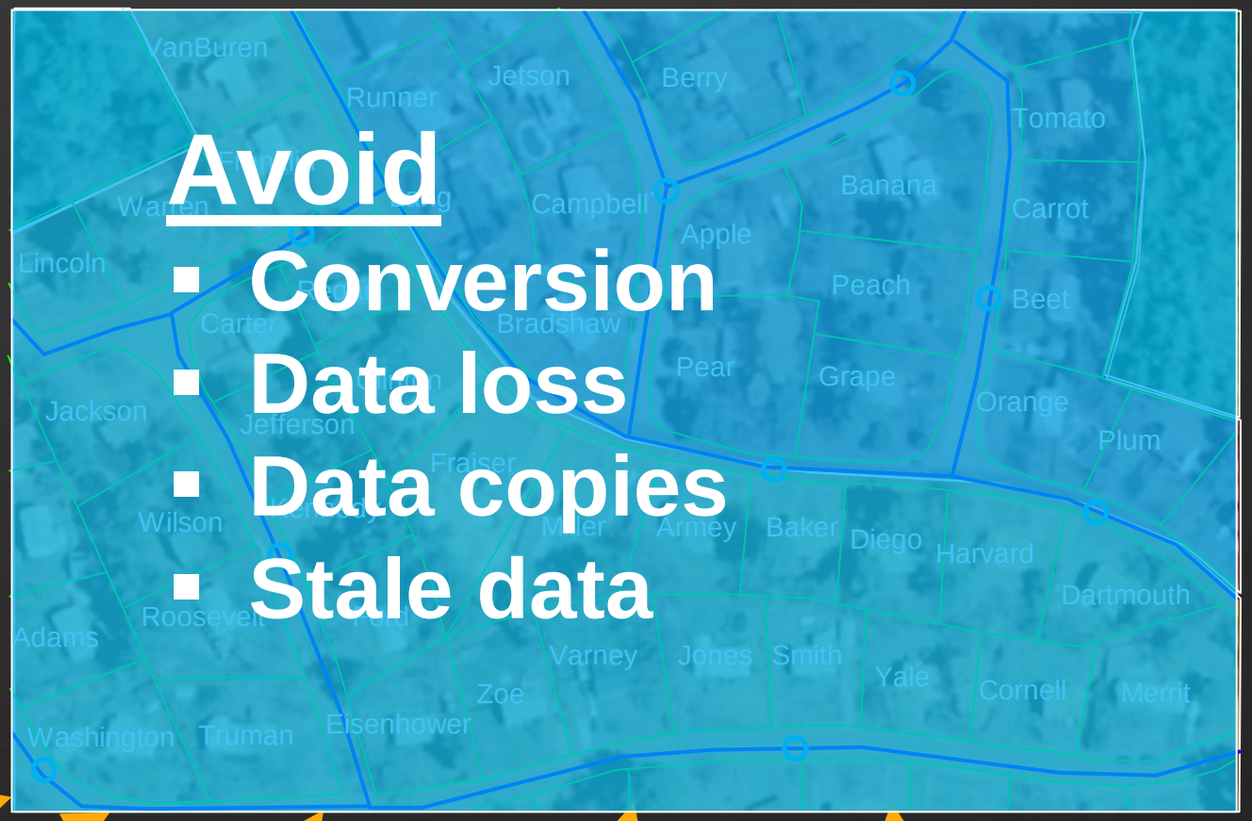


Autodesk

FDO Data Access Technology

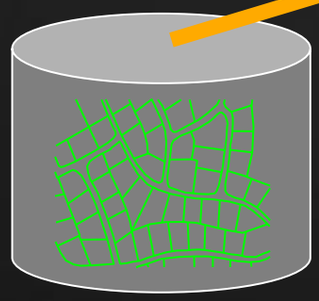
Work seamlessly with geospatial data

Directly access spatial data using Feature Data Objects (FDO) technology

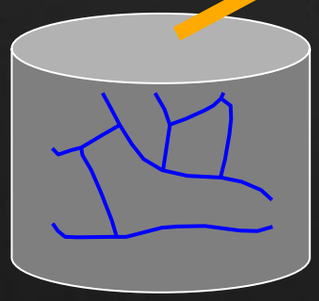


Avoid

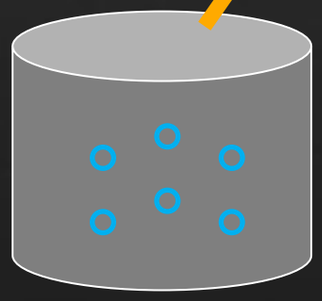
- **Conversion**
- **Data loss**
- **Data copies**
- **Stale data**



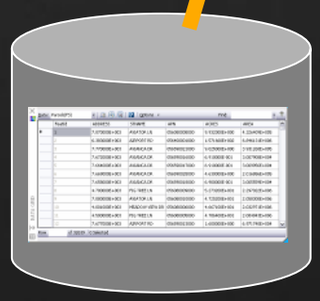
Parcel data from ESRI SHP file



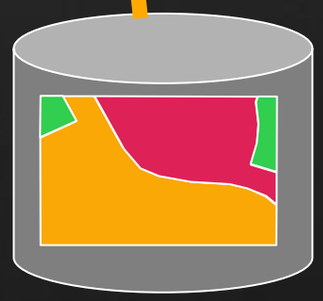
Utility data from DWG™ files



Hydrant locations from Survey



Property data from Microsoft Office Access®



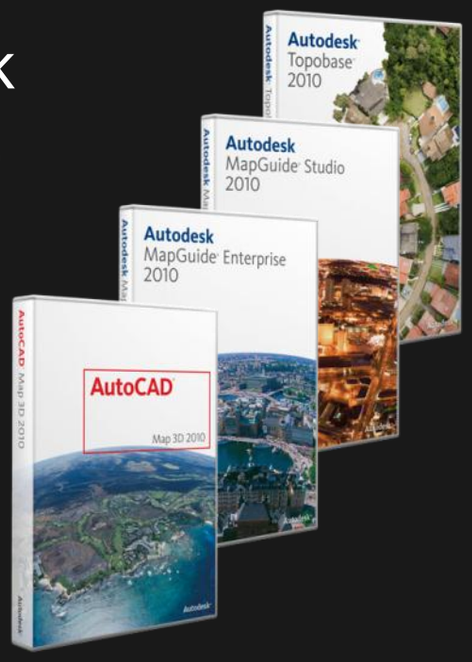
Zoning data ESRI ArcSDE®

Aerial photos

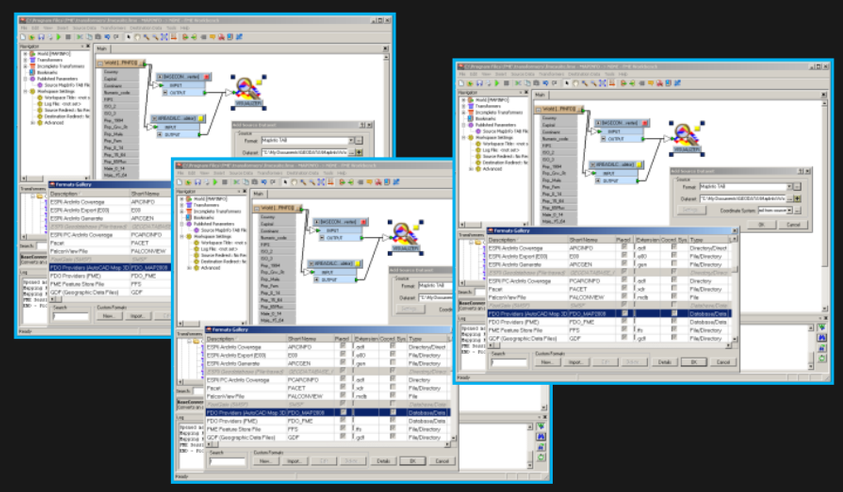
Autodesk

FDO Data Access Technology

Autodesk Products



Third-Party Solutions



Autodesk Certified Providers

Providers supporting **multiple** formats

- **ODBC** — Oracle and MS Access, Excel®
- **Raster** — 15+ raster formats
- **ArcSDE®** — Oracle and SQL Server®

Providers supporting **single** format

- SDF
- SHP
- WMS
- WFS
- SQL Server 2005
- Oracle
- MySQL / Sun
- SQL Server 2008
- GE Energy Smallworld

Providers from Third-Party & Open Source

Providers supporting **multiple** formats

- **FME** — 150+ vector and raster formats
- **OGR** — 25+ vector formats
- **GDAL** — 25+ raster formats

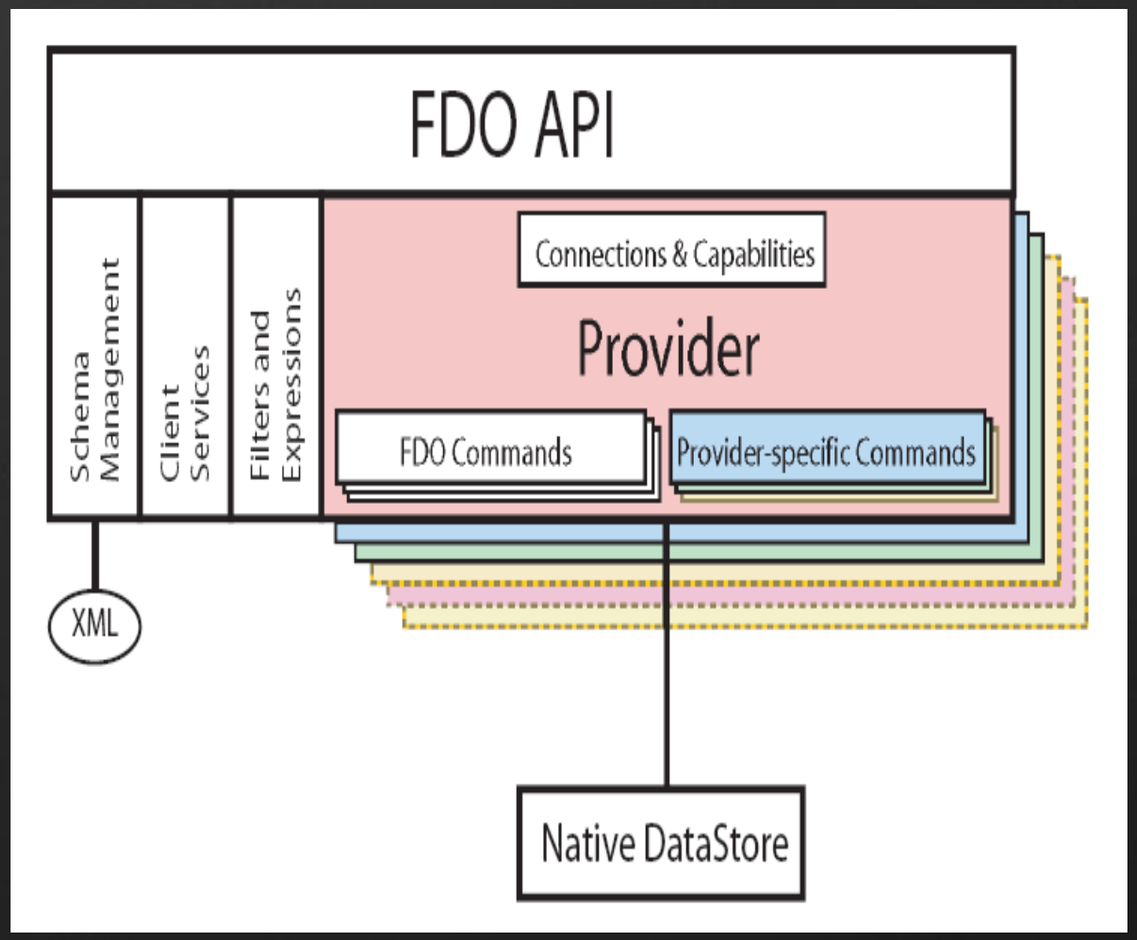
Providers supporting **single** format

- PostGIS
- SuperMap
- Oracle
- IBM Informix Dynamic
- SQL Server 2005
- KML

Autodesk

FDO API

- An interface specification of the abstraction layer
- Provides a common abstraction layer for accessing geospatial data from a variety of data sources
- Geospatial Platform API exposes most FDO API functionality
- The API can be used independent of AutoCAD Map



Autodesk

Getting a list of installed providers

- Get the provider registry (IProviderRegistry)
  - FeatureAccessManager::GetProviderRegistry()
- Get the provider collection (ProviderCollection)
  - IProviderRegistry ::GetProviders()
- Go through the provider collection and get their names

```
//Get the provider registry
  IProviderRegistry providerReg = FeatureAccessManager.GetProviderRegistry();
  ProviderCollection providers = providerReg.GetProviders();
int nProviders = providers.Count;
Provider provider = null;
System.Collections.IEnumerator enumerator = providers.GetEnumerator() ;

 //Output all providers
while (enumerator.MoveNext())
{
    provider = (Provider)enumerator.Current;
   ed.WriteMessage (provider.Name);
}
```

Working with the Connection Object

- Get the connection manager
  - FeatureAccessManager::GetConnectionManager()
- Create connection (IConnection)
  - IConnectionManager::CreateConnection()
- Set connection properties
  - IConnectionPropertyDictionary::SetProperty()
- Open connection, if operation on data store is required
  - IConnection::Open()
- Close open connection with IConnection::Close() after all operations

```
// Get the connection manager
IConnectionManager connMgr =
OSGeo.FDO.ClientServices.FeatureAccessManager.GetConnectionManager();
// Create connection
IConnection conn = connMgr.CreateConnection("Autodesk.Oracle.3.4");
// Get the connection property dictionary
IConnectionPropertyDictionary propDict = conn.ConnectionInfo.ConnectionProperties;
// Set ALL required connection properties and open connection
propDict. SetProperty(“Service”, “ORCL”); propDict. SetProperty(“Username”, “Jack”);
Conn.Open();
```

Connection states

- Open
  - enum Connections.ConnectionState.ConnectionState_Open
  - Connection is open
- Closed
  - enum Connections.ConnectionState.ConnectionState_Closed
  - Connection is closed
- Busy
  - enum Connections.ConnectionState.ConnectionState_Busy
  - Connection is busy executing one or more commands. Executing another command while the connection is busy will result in an exception
- Pending
  - enum Connections.ConnectionState.ConnectionState_Pending
  - indicates that the application has issued an Open request, but the open is not completed because the application needs to include additional connection property values

Connection Capabilities

Getting connection capabilities

- Command capabilities
  - Describes the commands a provider supports
  - ICommandCapabilities = IConnection::CommandCapabilities
- Filter capabilities
  - Describes the provider's level of support for filter classes
  - IFilterCapabilities = IConnection::FilterCapabilities
- Geometry capabilities
  - Describes the provider's support for geometry
  - IGeometryCapabilities = IConnection::GeometryCapabilities
- Topology capabilities
  - Describes the provider's support for topology
  - ITopologyCapabilities = IConnection::TopologyCapabilities

Connection Capabilities

Getting connection capabilities

- Raster capabilities
  - Describes the provider's support for raster images
  - IRasterCapabilities = IConnection::RasterCapabilities
- Schema capabilities
  - Describes the provider's support for feature schema
  - ISchemaCapabilities = IConnection::SchemaCapabilities

```
// Does the SHP provider support multiple schema?
IConnectionManager connMgr =
OSGeo.FDO.ClientServices.FeatureAccessManager.GetConnectionManager();
// Create connection
IConnection conn = connMgr.CreateConnection("OSGeo.SHP.3.4");
// Get the schema capabilities interface
ISchemaCapabilities schemaCpb  = conn.SchemaCapabilities;
// Check the if connection supports multiple schemas
bool supportMultSchemas = schemaCpb.SupportsMultipleSchemas;
```

Connection Capabilities

Getting connection capabilities

- Command capabilities

```
// Create an FDO connection for the provider
OSGeo.FDO.IConnectionManager connManager =
FeatureAccessManager.GetConnectionManager();
IConnection conn = connManager.CreateConnection(providerName);

//Get some supported command capabilities
 ICommandCapabilities comCap = conn.CommandCapabilities
int[] capArray = comCap.Commands;
int nCaps = capArray.Length;
for (int i = 0; i < nCaps; i++)
{
    if (capArray[i] == (int)CommandType.CommandType_Select)
            ed.WriteMessage("\nSelect command supported");
   if (capArray[i] == (int)CommandType.CommandType_CreateDataStore)
            ed.WriteMessage("\nCreate data store command supported");
}
```

Schema Management

- A schema is a logical description of the data types used to model real-world objects
- A feature source can have more than one schema

Listing the schemas in a feature source

- Create “Describe Schema” command type
  - IConnection::CreateCommand(CommandType.CommandType_DescribeSchema)
- Execute the command - IDescribeSchema::Execute()

```
IConnectionManager connMgr =
OSGeo.FDO.ClientServices.FeatureAccessManager.GetConnectionManager();
IConnection conn = connMgr.CreateConnection("Autodesk.Oracle.3.4");
// Create the command
IDescribeSchema cmd = conn. CreateCommand
 CommandType.CommandType_DescribeSchema)
// Execute the command
FeatureSchemaCollection schemas = cmd.Execute();
// Get the schemas
String schema_Name = schemas.get_Item(i).Name;
```

Creating Commands

- **Most operations in the data store are carried out through command execution**

- **enum CommandType lists all command types**

CommandType_Select - Select command
CommandType_Update - Update command
CommandType_DescribeSchema - DescribeSchema command
CommandType_ApplySchema - ApplySchema command
CommandType_DestroySchema - DestroySchema command
CommandType_GetSpatialContexts - GetSpatialContexts command

- **Create a command using IConnection::CreateCommand(), passing in command type. Return type is Icommand which should be cast to the appropriate subclass**

Creating Commands

- Each provider supports only a subset of available commands
  - Check command support using IConnection::CommandCapabilities
- Execute command using ICommand::Execute(), to return IFeatureReader object

```
// Creating and executing a Select command
// Get the connection manager
IConnectionManager connMgr =
OSGeo.FDO.ClientServices.FeatureAccessManager.GetConnectionManager();
// Create connection
IConnection conn = connMgr.CreateConnection("Autodesk.Oracle.3.4");
// Get the connection property dictionary
IConnectionPropertyDictionary propDict = conn.ConnectionInfo.ConnectionProperties;
// Set ALL required connection properties and open connection
propDict. SetProperty(“Service”, “ORCL”); propDict. SetProperty(“Username”, “Jack”);
Conn.Open();
// Create a select command
ISelect selCmd =
(ISelect)conn.CreateCommand CommandType.CommandType_Select    // Set ALL
required conmmand properties and execute command
selCmd.Filter  = OSGeo.FDO.Filter.Filter.Parse("ID >1");
IFeatureReader ftrRdr = selCmd.Execute
```

Schema Management

Feature Class (FeatureClass)

- Feature class contains properties of features
- **Property types**:
  - Geometry
  - Data
  - Raster
  - Object
  - Association
- Identity properties are used to uniquely identify a feature in a feature class

Schema Management

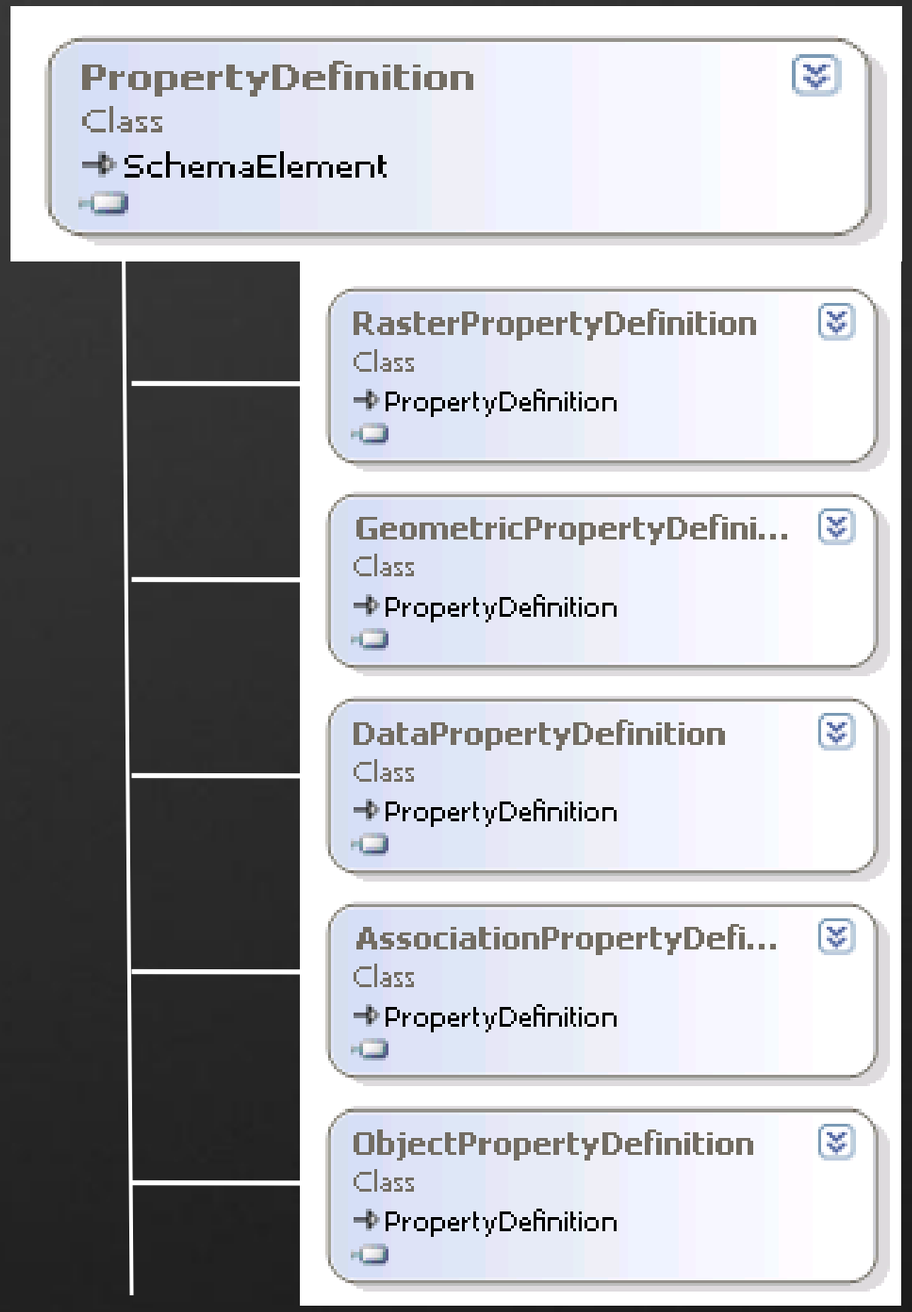
Important FeatureClass Properties

- ClassDefinition::Properties
  - returns PropertyDefinitionCollection
    - contains PropertyDefinition
- ClassDefinition::IdentityProperties
  - returns DataPropertyDefinitionCollection
    - contains DataPropertyDefinition
- FeatureClass::GeometryProperty
  - returns GeometryPropertyDefinition

PropertyDefinition Class

PropertyDefinition

- The details of a feature class property
- Has 5 subclasses or types
  - DataPropertyDefinition
  - GeometricPropertyDefinition
  - RasterPropertyDefinition
  - ObjectPropertyDefinition
  - AssociationPropertyDefinition



Autodesk

Schema Management

Creating a schema

- Instantiate a feature class object
- Instantiate schema object
- Add feature class to schema class collection
- Set up data and geometry property definitions and add them to feature class properties definition collection

```
// Create schema and feature class objects
FeatureSchema schema = new
FeatureSchema(“SchemaName”, "");
FeatureClass fc = new FeatureClass(“ClassName”, "");

//Add feature class to schema class collection
schema.Classes.Add(fc);

//Set up a data property definition
DataPropertyDefinition p = new
DataPropertyDefinition(“idPropertyName”, "");
p.DataType = DataType.DataType_Int32;
p.IsAutoGenerated = true;

//Add property definitions to feature class
//properties collection
fc.Properties.Add(p);
fc.IdentityProperties.Add(p);

GeometricPropertyDefinition gp = new
GeometricPropertyDefinition(“GeometryPropertyName”,
"");
gp.GeometryTypes = geometryType;
fc.Properties.Add(gp);
fc.GeometryProperty = gp;
```

Schema Management

Creating a schema (contd.)

- Create “Apply Schema” command
  - IConnection::CreateCommand
- Assign schema object to command’s feature schema
- Execute command
  - IApplySchema::Execute

```
//Create “apply schema” command
IApplySchema applySchemaCmd =
conn.CreateCommand(CommandType.CommandType_ApplySchema) as
IApplySchema;

//Specify the command’s feature schema
applySchemaCmd.FeatureSchema = schema;

//Execute command to create schema
applySchemaCmd.Execute();
```

Schema Management

Destroying a schema

- Create “Destroy Schema” command
  - IConnection::CreateCommand
  - Specify the schema to destroy
- Execute command
  - IDestroySchema::Execute

```
//Create “destroy schema” command
IDestroySchema destroySchemaCmd =
conn.CreateCommand(CommandType.CommandType_DestroySchema)
as IDestroySchema;

//Specify schema name
destroySchemaCmd.SchemaName = “TestSchema”;

//Execute command to destroy schema
destroySchemaCmd.Execute();
```

Creating a SDF

- Create the connection object and set its properties
  - IConnectionManager::CreateConnection
- Create the data store and set its properties
  - IConnection::CreateCommand(CommandType.CommandType_CreateDataStore)
- Create a spatial context for the data store
  - IConnection::CreateCommand(CommandType.CommandType_CreateSpatialContext)
- Create a schema for the data store and apply it
  - IConnection::CreateCommand(CommandType.CommandType_ApplySchema)

Autodesk®