

Feature Data Objects (FDO)

Developer's Guide

The Autodesk logo is displayed in white text on a black rectangular background. The word "Autodesk" is written in a bold, sans-serif font, oriented vertically from bottom to top.

March 2009

© 2009 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, 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 Web Format, Discreet, DWF, DWG, DWG (logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DXF, Ecotect, Exposure, Extending the Design Team, Face Robot, FBX, Filmbox, Fire, Flame, Flint, FMDesktop, Freewheel, Frost, GDX Driver, Gmax, Green Building Studio, Heads-up Design, Heidi, HumanIK, IDEA Server, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Inventor, Inventor LT, Kaydara, Kaydara (design/logo), Kynapse, Kynogon, LandXplorer, LocationLogic, Lustre, Matchmover, Maya, Mechanical Desktop, Moonbox, MotionBuilder, Movimento, Mudbox, NavisWorks, ObjectARX, ObjectDBX, Open Reality, Opticore, Opticore Opus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProjectPoint, ProMaterials, RasterDWG, Reactor, RealDWG, Real-time Roto, REALVIZ, Recognize, Render Queue, Retimer, Reveal, Revit, Showcase, ShowMotion, SketchBook, Smoke, Softimage, Softimage|XSI (design/logo), SteeringWheels, Stitcher, Stone, StudioTools, Topobase, Toxik, TrustedDWG, ViewCube, Visual, Visual Construction, Visual Drainage, Visual Landscape, Visual Survey, Visual Toolbox, Visual LISP, Voice Reality, Volo, Vtour, Wire, Wiretap, WiretapCentral, XSI, and XSI (design/logo).

The following are registered trademarks or trademarks of Autodesk Canada Co. in the USA and/or Canada and other countries: Backburner, Multi-Master Editing, River, and Sparks.

The following are registered trademarks or trademarks of MoldflowCorp. in the USA and/or other countries: Moldflow, MPA, MPA (design/logo), Moldflow Plastics Advisers, MPI, MPI (design/logo), Moldflow Plastics Insight, MPX, MPX (design/logo), Moldflow Plastics Xpert.

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

| | | |
|------------------|---|-----------|
| Chapter 1 | About This Guide | 1 |
| | Audience and Purpose | 1 |
| | How This Guide Is Organized | 1 |
| | What's New | 3 |
| Chapter 2 | Introduction | 5 |
| | What Is the FDO API? | 5 |
| | From the Perspective of the Client Application User | 5 |
| | From the Perspective of the Client Application Engineer | 5 |
| | Getting Started | 6 |
| | FDO Architecture and Providers | 7 |
| | What Is a Provider? | 9 |
| | Developing Applications | 11 |
| Chapter 3 | FDO Concepts | 13 |
| | Data Concepts | 13 |
| | Operational Concepts | 18 |
| Chapter 4 | Development Practices | 21 |
| | Memory Management | 21 |
| | Collections | 23 |
| | Exception Handling | 23 |

| | | |
|------------------|--|------------|
| | Exception Messages | 24 |
| | Managing FdoPtr Behaviors | 25 |
| Chapter 5 | Establishing a Connection | 27 |
| | Connection Semantics | 27 |
| | Establishing a Connection | 29 |
| Chapter 6 | FDO Capabilities | 35 |
| | FDO Capabilities | 35 |
| | Introduction | 35 |
| | Provider Type | 36 |
| | Command | 37 |
| | Connection | 43 |
| | Expression | 46 |
| | Filter | 65 |
| | Geometry | 68 |
| | Raster | 69 |
| | Schema | 70 |
| | Expressible as Boolean | 72 |
| | Not Expressible as a Boolean | 74 |
| Chapter 7 | Schema Management | 77 |
| | Schema Package | 77 |
| | Schema Mappings | 79 |
| | Schema Overrides | 80 |
| | Working with Schemas | 80 |
| | FDOFeatureClass | 82 |
| | FDOClass | 82 |
| | Non-Feature Class Issues | 83 |
| | Modifying Models | 85 |
| | Schema Element States | 86 |
| | Rollback Mechanism | 87 |
| | FDO XML Format | 87 |
| | Creating and Editing a GML Schema File | 93 |
| | Schema Management Examples | 103 |
| Chapter 8 | Data Maintenance | 109 |
| | Data Maintenance Operations | 109 |
| | Inserting Values | 109 |
| | Updating Values | 115 |
| | Deleting Values | 116 |
| | Related Class Topics | 117 |

| | | |
|-------------------|--|------------|
| Chapter 9 | Performing Queries | 119 |
| | Creating a Query | 119 |
| | Query Example | 120 |
| Chapter 10 | Long Transaction Processing | 125 |
| | What Is Long Transaction Processing? | 125 |
| | Supported Interfaces | 126 |
| Chapter 11 | Filter and Expression Languages | 129 |
| | Filters | 129 |
| | Expressions | 130 |
| | Filter and Expression Text | 130 |
| | Language Issues | 131 |
| | Provider-Specific Constraints on Filter and Expression Text | 131 |
| | Filter Grammar | 131 |
| | Expression Grammar | 133 |
| | Filter and Expression Keywords | 134 |
| | Data Types | 134 |
| | Identifier | 134 |
| | Parameter | 134 |
| | String | 134 |
| | Integer | 135 |
| | Double | 135 |
| | DateTime | 135 |
| | Operators | 135 |
| | Special Character | 137 |
| | Geometry Value | 137 |
| Chapter 12 | The Geometry API | 141 |
| | Introduction | 141 |
| | FGF and WKB | 141 |
| | FGF Binary Specification | 142 |
| | FGF Text | 148 |
| | Abstract and Concrete Classes | 148 |
| | Geometry Types | 149 |
| | Mapping Between Geometry and Geometric Types | 150 |
| | Spatial Context | 150 |
| | Specify Dimensionality When Creating Geometries Using String | |
| | Specifications | 151 |
| | Inserting Geometry Values | 152 |
| Chapter 13 | FDO Cookbook | 153 |
| | Introduction | 153 |

| | |
|---|------------|
| Recommendations | 153 |
| Registry | 153 |
| Connection | 154 |
| Capabilities | 157 |
| Data Store | 161 |
| User Management | 163 |
| Spatial Context | 165 |
| Basic Schema Operations | 168 |
| Insert Data | 171 |
| Select Data | 172 |
| Select Aggregate Data | 173 |
| Delete Data | 174 |
| Schema Overrides | 174 |
| Xml Serialize/Deserialize | 184 |
| C# | 184 |
| Geometry | 186 |
| Construction | 186 |
| C# Namespaces | 186 |
| C# Geometry Scaffolding Classes | 187 |
| C# Geometries Constructed using IDirectPositionImpl | 189 |
| C# Geometries Constructed Using Geometry Subcomponents | 190 |
| C# Aggregate Geometries | 191 |
| C# Geometries from Text Specifications | 193 |
| Deconstruction | 193 |
| Appendix A Autodesk FDO Provider for Oracle | 199 |
| What Is FDO Provider for Oracle? | 199 |
| FDO Provider for Oracle General Requirements | 200 |
| Create the f_user role | 200 |
| FDO Provider for Oracle Connection | 200 |
| FDO Provider for Oracle and Foreign Schemas | 201 |
| Foreign Schema Settings | 201 |
| Settings on the FDO Schema Instance | 201 |
| Settings on the Foreign Schema Instance | 202 |
| Oracle Identity Property | 202 |
| Read-Write Privileges | 202 |
| Foreign Schema Limitations | 202 |
| Ensuring Valid Views When Applying a Feature Schema Against a Foreign Schema | 203 |
| Table Name Restrictions When Working with a Foreign Schema | 205 |
| Schema Access on a Different Oracle Instance | 205 |
| Logical to Physical Schema Mapping | 205 |
| Physical to Logical Schema Mapping | 206 |

| | |
|--|------------|
| FDO Provider for Oracle and Schema Overrides | 209 |
| Schema Override Set | 209 |
| Class Table Overrides | 210 |
| Data Property Overrides | 210 |
| Object Property Overrides | 210 |
| Geometric Property Overrides | 210 |
| Oracle-Specific Schema Creation Restrictions | 211 |
| FDOFeatureClass | 211 |
| Classes | 211 |
| Properties | 211 |
| Data Properties | 211 |
| Identity Properties | 211 |
| String Properties | 212 |
| Decimal Properties | 212 |
| Geometric Properties | 212 |
| Object Properties | 212 |
| Oracle-Specific Schema Modification Restrictions | 212 |
| Schema Element Descriptions | 212 |
| Data Properties | 213 |
| Views | 213 |
| Oracle-Specific Deletion Restrictions | 215 |
| FDOClassDefinition | 215 |
| FDOClass | 215 |
| Property | 215 |
| Oracle Reserved Words Used with Filter and Expression Text | 215 |
| Locking and Long Transactions | 216 |
| OWM and FDO Lock Types | 217 |
| Example: AllLongTransactionExclusiveLock | 218 |
| FDO Provider for Oracle Capabilities | 219 |
| Appendix B OSGeo FDO Provider for ArcSDE | 227 |
| What Is FDO Provider for ArcSDE? | 227 |
| FDO Provider for ArcSDE Software Requirements | 227 |
| Installed Components | 227 |
| External Dependencies | 228 |
| FDO Provider for ArcSDE Limitations | 228 |
| ArcSDE Limitations | 228 |
| Relative to ArcObjects API and ArcGIS Server API | 229 |
| Curved Segments | 229 |
| Locking and Versioning | 229 |
| Table Creation | 230 |
| Identity Row ID Column and Enable Row Locking | 230 |
| Disable Row Locking and Enable Versioning | 231 |
| FDO Provider for ArcSDE Connection | 232 |
| Data Type Mappings | 232 |
| Creating a Feature Schema | 233 |

| | | |
|-------------------|---|------------|
| | Logical to Physical Schema Mapping | 239 |
| | Physical to Logical Schema Mapping | 239 |
| | FDO Provider for ArcSDE Capabilities | 243 |
| Appendix C | OSGeo FDO Provider for MySQL | 249 |
| | What Is FDO Provider for MySQL? | 249 |
| | Logical to Physical Schema Mapping | 250 |
| | Physical to Logical Schema Mapping | 251 |
| | FDO Provider for MySQL Capabilities | 253 |
| Appendix D | OSGeo FDO Provider for ODBC | 259 |
| | What Is FDO Provider for ODBC? | 259 |
| | Physical to Logical Schema Mappings | 260 |
| | FDO Provider for ODBC Capabilities | 265 |
| Appendix E | Autodesk FDO Provider for Raster | 271 |
| | What Is FDO Provider for Raster? | 271 |
| | FDO Provider for Raster Capabilities | 273 |
| Appendix F | OSGeo FDO Provider for SDF | 277 |
| | What Is FDO Provider for SDF? | 277 |
| | FDO Provider for SDF Capabilities | 278 |
| Appendix G | OSGeo FDO Provider for SHP | 285 |
| | What Is FDO Provider for SHP? | 285 |
| | Creating SHP Files for the Logical to Physical Schema Mapping | 286 |
| | Creating SHP Files for the Physical to Logical Schema Mapping | 287 |
| | Logical to Physical Schema Mapping | 289 |
| | Physical to Logical Schema Mapping | 289 |
| | FDO Provider for SHP Capabilities | 291 |
| Appendix H | Autodesk FDO Provider for SQL Server | 297 |
| | What Is FDO Provider for SQL Server? | 297 |
| | Logical to Physical Schema Mapping | 297 |
| | Physical to Logical Schema Mapping | 298 |
| | FDO Provider for SQL Server Capabilities | 300 |
| Appendix I | OSGeo FDO Provider for SQL Server Spatial | 307 |
| | Logical to Physical Schema Mapping | 307 |
| | Physical to Logical Schema Mapping | 308 |

| | | |
|-------------------|-----------------------------------|------------|
| Appendix J | OSGeo FDO Provider for WFS | 311 |
| | What Is FDO Provider for WFS? | 311 |
| | FDO Provider for WFS Capabilities | 312 |
| Appendix K | OSGeo FDO Provider for WMS | 317 |
| | What Is FDO Provider for WMS? | 317 |
| | FDO Provider for WMS Capabilities | 318 |
| Appendix L | Expression Functions | 323 |
| | Introduction | 323 |
| | Expression Function List | 323 |
| | Aggregate Expression Functions | 324 |
| | Conversion Expression Functions | 324 |
| | Date Expression Functions | 325 |
| | Geometry Expression Functions | 325 |
| | Mathematical Expression Functions | 325 |
| | Numeric Expression Functions | 326 |
| | String Expression Functions | 327 |
| | Index | 329 |

About This Guide

1

The *FDO Developer's Guide* introduces the Feature Data Objects (FDO) application programming interface (API) and explains how to use its customization and development features.

NOTE For detailed information about installing the FDO SDK and getting started using the FDO API, see *The Essential FDO* (FET_TheEssentialFDO).

Audience and Purpose

This guide is intended to be used by developers of FDO applications. It introduces the FDO API, explains the role of a feature provider, and provides detailed information and examples about how to code your application.

How This Guide Is Organized

This guide consists of the following chapters and appendixes:

- [Introduction](#) (page 5), provides an overview of the FDO API and the function of FDO feature providers.
- [FDO Concepts](#) (page 13), describes the key data and operational concepts upon which FDO is constructed.
- [Development Practices](#) (page 21), discusses the best practices to follow when using FDO for application development.
- [Establishing a Connection](#) (page 27), describes how to establish a connection to an FDO provider.
- Capabilities, discusses the Capabilities API, which is used to determine the capabilities of a particular provider.

- [Schema Management](#) (page 77), describes how to create and work with schemas and presents the issues related to schema management.
- [Data Maintenance](#) (page 109), provides information about using the FDO API to maintain the data.
- [Performing Queries](#) (page 119), describes how to create and perform queries.
- [Long Transaction Processing](#) (page 125), discusses long transactions (LT) and how to implement LT processing in your application.
- [Filter and Expression Languages](#) (page 129), discusses the use of filter expressions to specify to an FDO provider how to identify a subset of the objects of an FDO data store.
- [The Geometry API](#) (page 141), discusses the various Geometry types and formats and describes how to work with the Geometry API to develop FDO-based applications.
- [Autodesk FDO Provider for Oracle](#) (page 199), discusses development issues that apply when using FDO Provider for Oracle.
- [OSGeo FDO Provider for ArcSDE](#) (page 227), discusses development issues that apply when using FDO Provider for ESRI® ArcSDE®.
- [OSGeo FDO Provider for MySQL](#) (page 249), discusses development issues that apply when using FDO Provider for MySQL.
- [OSGeo FDO Provider for ODBC](#) (page 259), discusses development issues that apply when using FDO Provider for ODBC.
- [Autodesk FDO Provider for Raster](#) (page 271), discusses development issues that apply when using FDO Provider for Raster.
- [OSGeo FDO Provider for SDF](#) (page 277), discusses development issues that apply when using FDO Provider for SDF.
- [OSGeo FDO Provider for SHP](#) (page 285), discusses development issues that apply when using FDO Provider for SHP (Shape).
- [Autodesk FDO Provider for SQL Server](#) (page 297), discusses development issues that apply when using FDO Provider for SQL Server.
- [OSGeo FDO Provider for WFS](#) (page 311), discusses development issues that apply when using FDO Provider for WFS.
- [OSGeo FDO Provider for WMS](#) (page 317), discusses development issues that apply when using FDO Provider for WMS.

- [Expression Functions](#) (page 323), outlines the signatures and implementation details for the enhanced expression functions.

What's New

This section summarizes the changes and enhancements you will find in this version of FDO.

Enhanced Set of Expression Functions

The enhanced set includes aggregate, conversion, date, mathematical, numeric, string and geometry functions. All functions are supported by all providers, with the exception of the Raster, WFS and WMS providers.

For more information and implementation details about the expression function signatures, the RDBMS-specific built-in support for some of the functions, and the provider-specific support, see the appendix [Expression Functions](#) (page 323).

Introduction

2

You can use the APIs in the FDO API to manipulate, define, and analyze geospatial information.

This chapter introduces application development with the FDO API and explains the role of a feature provider.

What Is the FDO API?

From the Perspective of the Client Application User

The FDO API is a set of APIs used for creating, managing, and examining information, enabling Autodesk GIS products to seamlessly share spatial and non-spatial information, with minimal effort.

FDO is intended to provide consistent access to feature data, whether it comes from a CAD-based data source, or from a relational data store that supports rich classification. To achieve this, FDO supports a model that can readily support the capabilities of each data source, allowing consumer applications functionality to be tailored to match that of the data source. For example, some data sources may support spatial queries, while others do not. Also, a flexible metadata model is required in FDO, allowing clients to adapt to the underlying feature schema exposed by each data source.

From the Perspective of the Client Application Engineer

The FDO API provides a common, general purpose abstraction layer for accessing geospatial data from a variety of data sources. The API is, in part, an interface specification of the abstraction layer. A provider, such as Autodesk FDO Provider for Oracle, is an implementation of the interface for a specific type of data source (for example, for an Oracle relational database). The API supports the standard

data store manipulation operations, such as querying, updating, versioning, locking, and others. It also supports analysis.

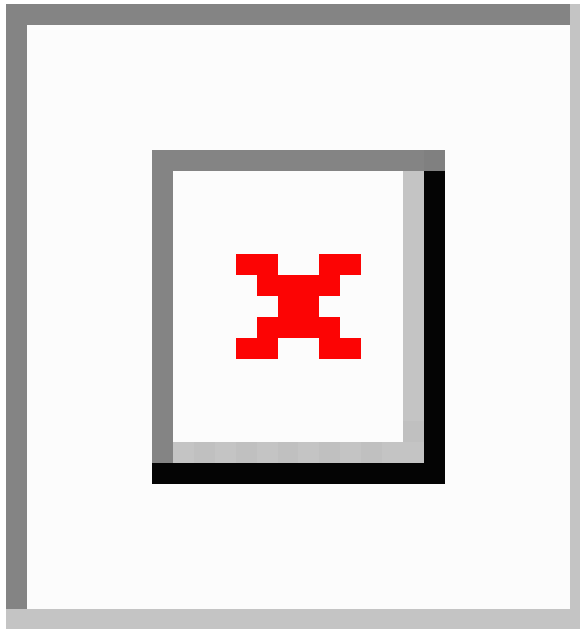
The API includes an extensive set of methods that return information about the capabilities of the underlying data source. For example, one method indicates whether the data source supports the creation of multiple schemas, and another indicates whether the data source supports schema modification.

A core set of services for providers is also available in the API, such as provider registration, schema management, filter and expression construction, and XML serialization and deserialization.

The API uses an object-oriented model for the construction of feature schema. A feature is a class, and its attributes, including its geometry, are a property of the class. The instantiation of a feature class, a Feature Data Object (FDO), can contain other FDOs.

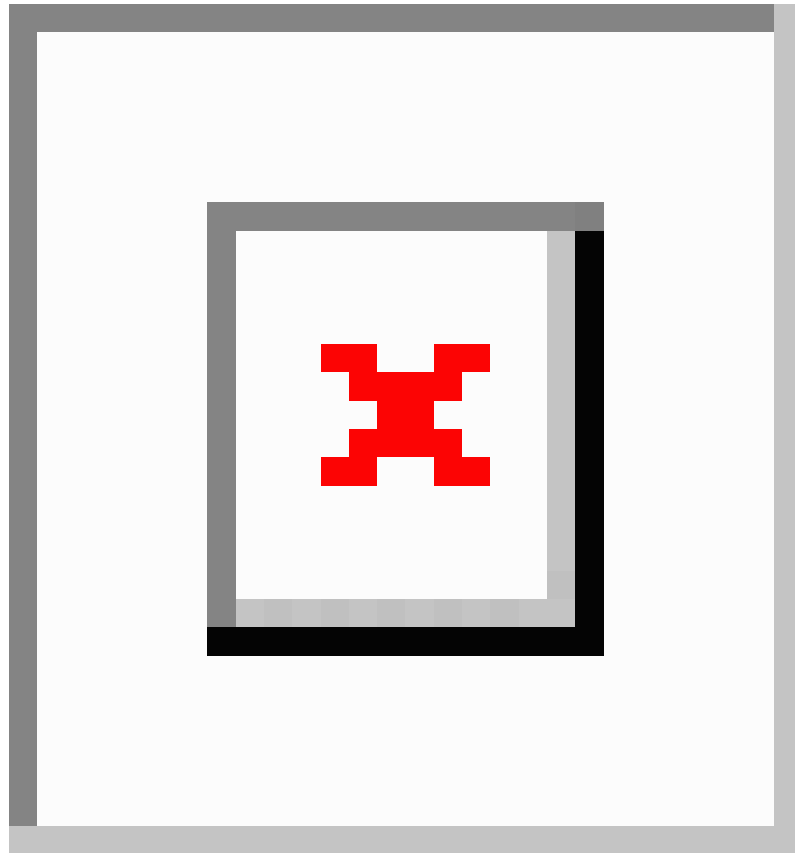
Getting Started

For detailed information to help you install and get started using Feature Data Objects (FDO), see *The Essential FDO*. It provides details about connecting to and configuring providers, data store management (create/delete), user IDs (create, grant permissions), and spatial context.



FDO Architecture and Providers

The following diagram shows the high-level overview architecture of the FDO API and included FDO providers.



FDO Architecture and Providers

FDO Packages

FDO is assembled in conceptual packages of similar functionality. This conceptual packaging is reflected in the substructure of the FDO SDK “includes” folder. For more information about the structure, see *The Essential FDO*.

FDO commands, provider-specific commands, and connections and capabilities provide access to native data stores through each different FDO provider. Schema management (through XML), client services, and filters and expressions

are provider-independent packages that tie into the FDO API. Each of these are explained in more detail in subsequent sections.

The FDO API consists of classes grouped within the following packages:

- **Commands package.** Contains a collection of classes that provide the commands allowing the application to select and update features, define new types of feature classes, lock features, and perform analysis on features. Each Command object executes a specific type of command against the underlying data store. In addition, FDO providers expose one or more Command objects.
- **Connections/Capabilities.** Contains a collection of classes that establish and manage the connection to the underlying data store. Connection objects implement the `FdoIConnection` interface. Capabilities API provides the code for retrieving the various FDO provider capability categories, such as connection or schema capabilities. You can use this this API to determine the capabilities of a particular provider.
- **Filters and Expression package.** Contains a collection of classes that define filters and expression in FDO, which are used to identify a subset of objects of an FDO data store.
- **Client Services package.** Contains a collection of classes that define the client services in FDO that, for example, enable support for dynamic creation of connection objects given a provider name.
- **Schema package and FDO XML.** Contains a collection of classes that provides a logical mechanism for specifying how to represent geospatial features. The FDO feature schema is based somewhat on a subset of the OpenGIS and ISO feature models. FDO feature schemas can be written to an XML file. The `FdoFeatureSchema` and `FdoFeatureSchemaCollection` classes support the `FdoXmlSerializable` interface.

In addition, FDO is integrated with the Geometry API, which includes the classes that support specific Autodesk applications and APIs, including FDO.

For more information about each of the FDO packages, see *FDO API Reference Help* (FDO_API.chm) and subsequent chapters in this guide.

Provider API(s) complete the FDO API configuration. Each provider has a separate API reference Help (for example, SDF_Provider_API.chm).

What Is a Provider?

A provider is a specific implementation of the FDO API. It is the software component that provides access to data in a particular data store.

For this release, the providers that are included are as follows:

NOTE Autodesk FDO Provider for Oracle and FDO Provider for ArcSDE are listed first because they were included in previous releases. The remaining providers are new to this release and are in alphabetical order. Providers referenced in this document with “Autodesk” as part of their name are included only with Autodesk software. Other providers are open source. For more information, see the Open Source Geospatial Foundation website at www.OSGeo.org.

- **Autodesk FDO Provider for Oracle.** Read/write access to feature data in an Oracle-based data store. Supports spatial indexing, long transactions, and persistent locking. A custom API can gather provider information, transmit client services exceptions, list data stores, and create connection objects.
- **OsGeo FDO Provider for ArcSDE.** Read/write access to feature data in an ESRI ArcSDE-based data store (that is, with an underlying Oracle or SQL Server database). Supports describing schema, and inserting, selecting, updating, and deleting feature data in existing schemas; does not support creating or deleting schemas.
- **OsGeo FDO Provider for MySQL.** Read/write access to feature data in a MySQL-based data store. Supports spatial data types and spatial query operations. A custom API can gather information, transmit exceptions, list data stores, and create connection objects. MySQL architecture supports various storage engines, characteristics, and capabilities.
- **OsGeo FDO Provider for ODBC.** Read/write access to feature data in an ODBC-based data store. Supports XYZ feature objects and can define feature classes for any relational database table with X, Y, and optionally Z columns; does not support creating or deleting schema. Object locations are stored in separate properties in the object definition.
- **Autodesk FDO Provider for Raster.** Read-only access to feature data in raster-based file format. Supports various image and GIS data formats (for example, JPEG, PNG, MrSID, and others). Supports georeferenced file-based raster images and file-based grid coverages. Pixel-based images, such as satellite images, are useful underneath vector data.
- **OsGeo FDO Provider for SDF.** Read-write access to feature data in an SDF-based data store. Autodesk’s geospatial file format, SDF, supports

multiple features/attributes, provides high performance for large data sets and interoperability with other Autodesk products, and spatial indexing. The SDF provider a valid alternative to database storage. Note that this release of the SDF provider supports version 3.0 of the SDF file format.

- **OsGeo FDO Provider for SHP.** Read/write access to existing spatial and attribute data in an ESRI SHP-based data store, which consists of separate shape files for geometry, index, and attributes. Each SHP and its associated DBF file is treated as a feature class with a single geometry property. This is a valid alternative to database storage but does not support locking.
- **Autodesk FDO Provider for SQL Server.** Read/write access to feature data in a Microsoft SQL Server-based data store. A custom API supports schema read/write access, and geospatial and non-geospatial data read/write access.
- **OsGeo FDO Provider for WFS.** Read-only access to feature data in an OGC WFS-based data store. Supports client/server environment and retrieves geospatial data encoded in GML from one or more Web Feature Services sites. Client/server communication is encoded in XML with the exception of feature geometries, which are encoded in GML. Note that there is no public API documentation for this provider; all functionality is accessible via the base FDO API.
- **OsGeo FDO Provider for WMS.** Read-only access to feature data in an OGC WMS-based data store. Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information, which are generally rendered in PNG, GIF, or JPEG, or as vector-based Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

FDO supports retrieval and update of spatial and non-spatial GIS feature data through a rich classification model that is based on OpenGIS and ISO standards.

An overview of the relationships between providers, data sources, data stores, and schemas is presented in the FDO Architecture and Providers graphic.

For more detailed information about the providers, see the appropriate appendix in this document. Data sources and data stores are discussed in the [Establishing a Connection](#) (page 27) chapter. Schema concepts are discussed in the [Schema Management](#) (page 77) chapter.

Developing Applications

You will need to perform several major tasks in using the FDO API to develop a custom application. Each of these tasks breaks down into a number of more detailed coding issues.

The major development tasks are:

- Working with the Build Environment
- Establishing a Connection
- Schema Management
- Data Maintenance
- Creating Queries
- Using Custom Commands (Provider-Specific)

These tasks are explored in detail in the chapters that follow.

FDO Concepts

3

Before you can work properly with the FDO API, you need to have a good understanding of its basic, underlying concepts. This chapter defines the essential constructs and dynamics that comprise the FDO API. The definitions of these constructs and dynamics are grouped into two interdependent categories:

- **Data Concepts.** Definitions of the data constructs that comprise the FDO API
- **Operational Concepts.** Definitions of the operations that are used to manage and manipulate the data.

Data Concepts

All concepts that are defined in this section relate to the data that FDO is designed to manage and manipulate.

What Is a Feature?

A feature is an abstraction of a natural or man-made real world object. It is related directly or indirectly to geographic locations. A spatial feature has one or more geometric properties. For example, a road feature might be represented by a line, and a hydrant might be represented by a point. A non-spatial feature does not have geometry, but can be related to a spatial feature which does. For example, a road feature may contain a sidewalk feature that is defined as not containing a geometry.

What Is a Schema?

A schema is a logical description of the data types used to model real-world objects. A schema is not the actual data instances (that is, not a particular road or land parcel), rather it is metadata. A schema is a model of the types of data that would be found in a data store. For example, a schema which models the

layout of city streets has a class called Road, and this class has a property called Name. The definition of Road and its associated classes constitute the schema.

What Is a Schema Override?

A schema override comprises instructions to override the default schema mappings. For example, an RDBMS-type FDO provider could map a feature class to a table of the same name by default. A schema override might map the class to a differently named table, for example, by mapping the "pole" class to the "telco_pole" table.

What is a Schema Mapping

A Schema Mapping is a correspondence between a Schema Element and a physical object in a data store. For example, OSGeo FDO Provider for MySQL maps each Feature Class onto a table in the MySQL database where the data store resides. The physical structure of data stores for each FDO provider can vary greatly, so the types of Schema Mappings can also vary between providers. Each provider defines a set of default schema mappings. For example, OSGeo FDO Provider for MySQL maps a class to a table of the same name by default. These defaults can be overridden by specifying Schema Overrides.

What Are Elements of a Schema?

A schema consists of a collection of schema elements. In the FDO API, schema elements are related to one another by derivation and by aggregation. An element of a schema defines a particular type of data, such as a feature class or a property, or an association. For example, a feature class definition for a road includes the class name (for example, Road), and the class properties (for example, Name, NumberLanes, PavementType, and Geometry).

What Is a Class Type?

A class type is a specialization of the base FDO class definition (FdoClassDefinition). It is used to represent the complex properties of spatial and non-spatial features.

What is a Feature Class?

A feature class is a schema element that describes a type of real-world object. It includes a class name and property definitions, including zero or more geometric properties. It describes the type of data that would be included in object instances of that type.

What Is a Property?

A property is a single attribute of a class and a class is defined by one or more property definitions. For example, a Road feature class may have properties called Name, NumberLanes, or Location. A property has a particular type, which can be a simple type, such as a string or number, or a complex type defined by a class, such as an Address type, which itself is defined by a set of properties, such as StreetNumber, StreetName, or StreetType.

There are five kinds of properties: association properties, data properties, geometric properties, object properties, and raster properties.

Individual properties are defined in the following sections.

What Is an Association Property?

The `FdoAssociationPropertyDefinition` class is used to model a peer-to-peer relationship between two classes. This relationship is defined at schema creation time and instantiated at object creation time. The association property supports various cardinality settings, cascading locks, and differing delete rules. An FDO filter can be based on association properties and `FdoIFeatureReader` can handle associated objects through the `GetObject()` method.

What Is a Data Property?

A data property is a non-spatial property. An instance of a data property contains a value whose type is either boolean, byte, date/time, decimal, single, double, `Int16`, `Int32`, `Int64`, string, binary large object, or character large object.

What Is Dimensionality?

Dimensionality, and the concept of dimension, has two different meanings in the discussion of geometry and geometric property.

The first is called shape dimensionality, and it is defined by the `FdoGeometricType` enumeration. The four shapes are point (0 dimensions), curve (1 dimensions), surface (2 dimensions), and solid (3 dimensions).

The other is called ordinate dimensionality, and it is defined by the `FdoDimensionality` enumeration. There are four ordinate dimensions: XY, XYZ, XYM, and XYZM. M stands for measure.

What Is a Geometric Property?

An instance of a geometric property contains an object that represents a geometry value. The definition of the geometric property may restrict an

object to represent a geometry that always has the same shape, such as a point, or it could allow different object instances to have different dimensions. For example, one geometric property object could represent a point and another could represent a line. Any combination of shapes is permissible in the specification of the geometric types that a geometry property definition permits. The default geometric property specifies that an object could represent a geometry that is any one of the four shapes.

With respect to ordinate dimensionality, all instances of a geometric property must have the same ordinate dimension. The default is XY.

Geometric property definitions have two attributes regarding ordinate dimensionality: HasElevation for Z and HasMeasure for M.

What is a Geometry?

A geometry is represented using geometric constructs either defined as lists of one or more XY or XYZ points or defined parametrically, for example, as a circular arc. While geometry typically is two- or three-dimensional, it may also contain the measurement dimension (M) to provide the basis for dynamic segments.

The geometry types are denoted by the `FdoGeometryType` enumeration and describe the following:

- Point
- LineString (one or more connected line segments, defined by positions at the vertices)
- CurveString (a collection of connected circular arc segments and linear segments)
- Polygon (a surface bound by one outer ring and zero or more interior rings; the rings are closed, connected line segments, defined by positions at the vertices)
- CurvePolygon (a surface bound by one outer ring and zero or more interior rings; the rings are closed, connected curve segments)
- MultiPoint (multiple points, which may be disjoint)
- MultiLineString (multiple LineStrings, which may be disjoint)
- MultiCurveString (multiple CurveStrings, which may be disjoint)
- MultiPolygon (multiple Polygons, which may be disjoint)
- MultiCurvePolygon (multiple CurvePolygons, which may be disjoint)

- MultiGeometry (a heterogeneous collection of geometries, which may be disjoint)

Most geometry types are defined using either curve segments or a series of connected line segments. Curve segments are used where non-linear curves may appear. The following curve segment types are supported:

- CircularArcSegment (circular arc defined by three positions on the arc)
- LineStringSegment (a series of connected line segments, defined by positions are the vertices)

There are currently no geometries of type “solid” (3D shape dimensionality) supported.

The `FdoIConnection::GetGeometryCapabilities()` method can be used to query which geometry types and ordinate dimensionalities are supported by a particular provider.

What Is an Object Property?

An object property is a complex property type that can be used within a class, and an object property, itself, is defined by a class definition. For example, the Address type example described previously in the Property definition. An object property may define a single instance for each class object instance (for example, an address property of a land parcel), or may represent a list of instances of that class type per instance of the owning class (for example, inspection records as a complex property of an electrical device feature class).

What is a Raster Property?

A raster property defines the information needed to process a raster image, for example, the number of bits of information per pixel, the size in pixels of the X dimension, and the size in pixels of the Y dimension, needed to process a raster image.

What Is a Spatial Context?

A spatial context describes the general metadata or parameters within which geometry for a collection of features resides. In particular, the spatial context includes the definition of the coordinate system, spheroid parameters, units, spatial extents, and so on for a collection of geometries owned by features.

Spatial context can be described as the “coordinate system plus identity.” Any geometries that are to be spatially related must be in a common spatial context.

The identity component is required in order to support separate workspaces, such as schematic diagrams, which are non-georeferenced. Also, it supports georeferenced cases. For example, two users might create drawings using some default spatial parameters (for example, rectangular and 10,000x10,000), although each drawing had nothing to do with the other. If the drawings were put into a common database, the users could preserve not only the spatial parameters, but also the container aspect of their data, using spatial context.

For more information about spatial context, see [Spatial Context](#) (page 150).

What is a Data Store?

A data store is a repository of an integrated set of objects. The objects in a data store are modeled either by classes or feature classes defined within one or more schemas. For example, a data store may contain data for both a LandUse schema and a TelcoOutsidePlant schema. Some data stores can represent data in only one schema, while other data stores can represent data in many schemas (for example, RDBMS-based data stores, such as MySQL).

Operational Concepts

The concepts that are defined in this section relate to the FDO operations used to manage and manipulate data.

What Is a Command?

In FDO, the application uses a command to select and update features, define new types of feature classes, lock features, version features, and perform some analysis of features. Each Command object executes a specific type of command against the underlying data store. Interfaces define the semantics of each command, allowing them to be well-defined and strongly typed. Because FDO uses a standard set of commands, providers can extend existing commands and add new commands, specific to that provider. Feature commands execute against a particular connection and may execute within the scope of a transaction.

An FDO command is a particular FDO interface that is used by the application to invoke an operation against a data store. A command may retrieve data from a data store (for example, a Select command), may update data in a data store (for example, an Update or Delete command), may perform some analysis (for example, an Activate Spatial Context command), or may cause some other change in a data store or session (for example, a Begin Transaction command).

What Is an Expression?

An expression is a construct that an application can use to build up a filter. An expression is a clause of a filter or larger expression. For example, “Lanes >=4 and PavementType = 'Asphalt'” takes two expressions and combines them to create a filter.

For more information about using expressions with FDO, see [Filter and Expression Languages](#) (page 129).

What Is a Filter?

A filter is a construct that an application specifies to an FDO provider to identify a subset of objects of an FDO data store. For example, a filter may be used to identify all Road type features that have 2 lanes and that are within 200 metres of a particular location. Many FDO commands use filter parameters to specify the objects to which the command applies. For example, a Select command uses a filter to identify the objects that the application wants to retrieve. Similarly, a Delete command uses a filter to identify the objects that the application wants to delete from the data store.

For more information about using filters with FDO, see [Filter and Expression Languages](#) (page 129).

What Is Locking?

A user can use locking to gain update control of an object in the data store to the exclusion of other users. There are two general types of locks—transaction locks and persistent locks. Transaction locks are temporary and endure only for the duration of the transaction (see [What Is a Transaction?](#) (page 19)).

Persistent locks applied to objects by a user remain with the object until either that user removes those locks or the locks are removed by another user with the appropriate authority.

What Is a Transaction?

A transaction changes the data store in some way. The way these changes affect the data store is determined by the transaction’s properties. For example, the Atomic property specifies that either all changes happen or non happen. In transaction processing the data store treats a series of commands as a single atomic unit of change to that data store. Either all changes generated by the commands are successful or the whole set is cancelled. A transaction is a single atomic unit of changes to a data store. The application terminates a transaction with either a “commit,” which applies the set of changes, or a “rollback,” which cancels the set of changes. Further, the data store may automatically

roll back a transaction if it detects a severe error in any of the commands within the transaction. A transaction has the following properties:

- **Atomic.** Either all changes generated by the commands within a transaction happen or none happen.
- **Consistent.** The transaction leaves the data store in a consistent state regarding any constraints or other data integrity rules.
- **Isolated.** Changes being made within a transaction by one user are not visible to other users until after that transaction is committed.
- **Durable.** After a transaction is completed successfully, the changes are persistent in the data store on disk and cannot be lost if the program or processor fails.

What Is a Long Transaction?

A long transaction (LT) is an administration unit used to group conditional changes to objects. Depending on the situation, such a unit might contain conditional changes to one or to many objects. Long transactions are used to modify as-built data in the database without permanently changing the as-built data. Long transactions can be used to apply revisions or alternates to an object.

What Is a Root Long Transaction?

A root long transaction is a long transaction that represents permanent data. Any long transaction has a root long transaction as an ancestor in its long transaction dependency graph.

Development Practices

4

This chapter explains several practices to follow when working with the FDO API and provides examples of how to follow these practices.

Memory Management

Some FDO functions (for example, the Create methods) allocate memory when they are called. This memory needs to be freed to prevent memory leaks. All destructors on FDO classes are protected, so you must call a `Release()` function to destroy them (thus freeing their allocated memory). Each class inherits from the `FdoIDisposable` class, which defines the `Release()` method and the `AddRef()` method.

In addition, these classes are reference counted, and the count is increased (by `AddRef()`) when you retrieve them through a `Get` function. After finishing with the object, you need to release it (just as with COM objects). The object is destroyed only when the reference count hits 0. Two macros are defined to help in the use of the `Release()` and `AddRef()` methods.

FDO_SAFE_RELEASE (*ptr)

If the “*ptr” argument is not null, `FDO_SAFE_RELEASE` calls the `release()` method of the object pointed to by the “*ptr” argument and then sets the local pointer to the object to `NULL`. The macro definition is `#define FDO_SAFE_RELEASE(x)`

```
{if (x) (x)->Release(); (x) = NULL;}
```

```
FdoFeatureClass* pBase = myClass->GetBaseClass();  
...  
// Must release reference added by GetBaseClass when done.  
FDO_SAFE_RELEASE(pBase);
```

FDO_SAFE_ADDREF (*ptr)

If the “*ptr” argument is not null, FDO_SAFE_ADDREF calls the AddRef() method of the object pointed to by the “*ptr” argument. The macro definition is `#define FDO_SAFE_ADDREF(x) ((x != NULL) ? (x)->AddRef(), (x):(NULL))`.

- `return FDO_SAFE_ADDREF(value)` returns NULL if value equals NULL or increments the reference count of the object that value points to and returns value.
- `m_list[index] = FDO_SAFE_ADDREF(value)` assigns NULL to the array entry if value is NULL or increments the reference count of the object that value points to and assigns value to the array entry.

FdoPtr

An `FdoPtr` smart pointer is provided to help manage memory. You wrap an FDO object in a `FdoPtr`. The requirement is that the object’s type must inherit from `FdoIDisposable`. The object is then released automatically when the `FdoPtr` goes out of scope. The following code illustrates how to use `FdoPtr`:

```
FdoPtr<FdoFeatureClass> pBase = myClass->GetBaseClass();  
...  
// No need to call FDO_SAFE_RELEASE.  
// Before it is destroyed, pBase calls Release() on the FdoFeature  
Class object
```

NOTE If, for some reason, you wanted to use `FDO_SAFE_RELEASE` on an `FdoPtr`, you would have to use an `FdoPtr` method to get a pointer to the object that `FdoPtr` wraps and pass that pointer to `FDO_SAFE_RELEASE`.

You can use `FdoPtr` for your own classes by inheriting from the abstract class `FdoIDisposable` and providing an implementation for the `Dispose()` method (typically `delete this;`).

FdoPtr Typedefs

Typedefs are provided that define identifiers representing Fdo classes wrapped by `FdoPtr`. An example is `typedef FdoPtr<FdoClass> FdoClassP`.

Collections

You can use FDO collection template classes to store your own objects. The requirements for your collection class and the class used to instantiate the template are the same as those for wrapping a class in a `FdoPtr`.

Exception Handling

In the FDO API, `FdoCommandException` class is the exception type thrown from classes in the Commands package, and `FdoConnectionException` class is the exception type thrown from classes in the Connections package. Both of these exception types derive from a language-level exception class that is environment-specific.

All exceptions are derived from the `FdoException` class. To catch and process specific exception types, nest catch statements as in the following example:

```
try {
    ... code
}
catch (FdoCommandException *ex) {
    .. process message
}
catch (FdoException *ex) {
    .. process message
}
```

In some cases, underneath an FDO command, the FDO level throws an `FdoException`. The FDO command then traps the `FdoException` and wraps it in an `FdoCommandException` (or `FdoSchemaException` for a schema command). In this case, several messages are returned by one exception. The following example shows how to process multiple messages from one exception:

```
catch ( FdoSchemaException* ex ) {
    // Loop through all the schema messages
    FdoException* currE = ex;
    while ( currE ) {
        CW2A msg(currE->GetExceptionMessage());
        acutPrintf ("FdoConnectionException: %s\n", msg);
        currE = currE->GetCause();
    }
}
```

An application function may need to catch and then re-throw exceptions in order to clean up memory. However, the need to do this can be eliminated by using `FdoPtr`. The following example cleans up memory on error:

```
FdoFeatureClass* pBase = NULL;
try {
    pBase = myClass->GetBaseClass();
    ...
}
catch (...) {
    FDO_SAFE_RELEASE(pBase);
    throw;
}
// Must release reference added by GetBaseClass when done.
FDO_SAFE_RELEASE(pBase);
```

The catch and rethrow is unnecessary when `FdoPtr` is used:

```
FdoPtr<FdoFeatureClass> pBase = myClass->GetBaseClass();
...
```

Exception Messages

Exception messages are localized. On Windows the localized strings are in resource-only DLLs, and on Linux they are in catalogs. The message DLLs are in the *bin* folder; the DLL name contains Message or Msg. The catalog files are in the */usr/local/fdo-3.2.0/nls* directory; the names of these files ends in *.cat*. NLS stands for National Language Support.

On Linux set the `NLSPATH` environment variable so that the runtime code can locate the message catalogs. For example, `export NLSPATH=/usr/local/fdo-3.2.0/nls/%N`.

On Windows you do not have to do anything special to enable the runtime code to locate the message DLLs.

The contents of the exception message files are indexed. When you call one of the `FdoException::NLSGetMessage` methods declared in `Exception.h`, you provide a message number argument. You may also provide a default message string argument. In the event that the exception message resource file cannot be found, the default message is substituted instead. If the default message string is not provided and the resource file cannot be found, the message number is used as the exception message. Not finding the resource file can only happen on Linux and only if the `NLSPATH` environment variable is not set.

The following two examples, when called on Linux with the NLSPATH environment variable not set, show the use of the default message and the message number in the exception message.

The following is an example of using the default string: `throw`

```
FdoSchemaException::Create(NlsMsgGet1(FDORDBMS_333, "Class '%1$s' not found", value->GetText()));
```

The following is an example of not setting the default string and using the message number instead: `FdoSchemaException* pNewException =`

```
FdoSchemaException::Create(  
FdoSmError::NLSGetMessage(FDO_NLSID(FDOSM_221),  
pFeatSchema->GetName()), pException);.
```

Managing FdoPtr Behaviors

The topics in this section describe several ways that you can manage FdoPtr behavior. For more information about managing FdoPtr behavior, see the related topics “FdoPtr <T> Class Template Reference” and “FdoIDisposable Class Reference” in the *FDO Reference Help* and *The Essential FDO*.

Chain Calls

Do not chain calls. If you do, returned pointers will not be released. For example, given an `FdoClassDefinition* pclassDef`:

```
psz = pclassDef ->GetProperties()->GetItem(0)->GetName();
```

The above code would result in two memory leaks. Instead use:

```
FdoPropertyDefinitionCollection* pprops = pclassDef -> GetProperties();  
FdoPropertyDefinition* ppropDef = pprops->GetItem(0);  
psz = ppropDef->GetName();  
ppropDef->Release();  
pprops->Release();
```

or (with FdoPtr):

```
FdoPtr<FdoPropertyDefinitionCollection> pprops = pclassDef-> GetProperties();  
FdoPtr<FdoPropertyDefinition> ppropDef = pprops-> GetItem(0);  
psz = ppropDef->GetName();
```

or (also with FdoPtr):

```
psz = FdoPtr <FdoPropertyDefinition> (FdoPtr <FdoPropertyDefini
tionCollection>(pclassDef->GetProperties())-> GetItem(0))->Get
Name();
```

Assigning Return Pointer of an FDO Function Call to a Non-Smart Pointer

If you are assigning the return pointer of an FDO function call to a non-smart pointer, then you should assign that pointer to a FdoPtr. For example:

```
FdoLineString* P = gf.CreateLineString(...);
FdoPtr <FdoLineString> p2 = FDO_SAFE_ADDREF(p);
```

Establishing a Connection

5

This chapter explains how to establish a connection to an FDO provider and provides a connection example.

Connection Semantics

Data Sources and Data Stores

The FDO API uses connection semantics to implement access to feature schema data. The term data store is used to refer to a collection of zero or more objects, which instantiate class definitions belonging to one or more FDO feature schema. The connection is to a data store because that is where data objects are stored. The underlying data source technologies used to hold data stores can be relational databases, such as, a MySQL database, or a file-based solution, such as an SDF file.

The mapping of a data store to data source technology can be one-to-one or many-to-one. For example, it is

- One-to-one when the connection is made by way of the OSGeo FDO Provider for ArcSDE and the ArcSDE server is using an Oracle database.
- Many-to-one when the data source is a MySQL database and the connection is made by way of the OSGeo FDO Provider for MySQL (in this case, the data store is like a container within a container).

When many-to-one mapping is possible, a connection can be made in one or two steps. For more information, see [Establishing a Connection](#) (page 29) and *The Essential FDO*.

The underlying data source technologies differ in the connection parameters used for connecting to a particular provider. The values for these parameters

are generated during the installation and configuration of the container technologies. For more information about these values and the process of installing and configuring the associated data source technologies, see the appropriate appendix in this document and *The Essential FDO*.

Providers

You connect to a data store by way of a provider.

The FDO API contains a registry interface that you can use to register or deregister a provider. See the class `FdoProviderRegistry` in *Inc/Fdo/ClientServices/ProviderRegistry.h*.

The providers are registered during the initialization of the FDO SDK. In order to connect to a provider, you will need the name of the provider in a particular format: `<Company/Foundation/Originator>.<Provider>.<Version>`. The `<Company/Foundation/Originator>` and `<Provider>` values are invariable. For specific values, see *The Essential FDO*.

In order to connect, you will need the full name including the `<Version>` value. You can retrieve the full name from the registry and display the set of provider names in a connection menu list. If, for whatever reason, you deregister a provider, save the registry information for that provider in case you want to reregister it again. The provider object returned by the registry has a `Set()` method to allow you to change values. However, the only value you can safely change is the display name. Sample code for using the provider registry is located in [Establishing a Connection](#) (page 29).

The registry contains the following information about a provider:

- **Name.** The unique name of the feature provider. This name should be of the form `<Company/Foundation/Originator>.<Provider>.<Version>`, for example, `Autodesk.Oracle.3.0` or `OSGeo.MySQL.3.0`.
- **DisplayName.** A user-friendly display name of the feature provider. The initial values of this property for the pre-registered providers are “Autodesk FDO Provider for Oracle”, “OSGeo FDO Provider for SDF”, etc., or the equivalent in the language of the country where the application is being used.
- **Description.** A brief description of the feature provider. For example, the the OSGeo FDO Provider for SDF description is “Read/write access to Autodesk's spatial database format, a file-based personal geodatabase that supports multiple features/attributes, spatial indexing, and file-locking.”

- **Version.** The version of the feature provider. The version number string has the form <VersionMajor>.<VersionMinor>.<BuildMajor>.<BuildMinor>, for example, 3.0.0.0.
- **FDOVersion.** The version of the feature data objects specification to which the feature provider conforms. The version number string has the form <VersionMajor>.<VersionMinor>.<BuildMajor>.<BuildMinor>, for example, 3.0.1.0.
- **libraryPath.** The FULL library path including the library name of the provider, for example, <FDO SDK Install Location>/bin/FdoRdbms.dll.
- **isManaged.** A flag indicating whether the provider is a managed or unmanaged .NET provider.

Establishing a Connection

As mentioned in a previous section, [Connection Semantics](#) (page 27), the FDO API uses a provider to connect to a data store and its underlying data source technology. These data source technologies must be installed and configured. Certain values generated during data source installation and configuration are used as arguments during the connection process. Because the FDO API does not provide any methods to automate the collection and presentation of these configuration values, either the application developer must request the user to input these configuration values during the connection process, or the application developer can provide an application configuration interface, which would populate the application with the container configuration values and thus allow the user to choose them from lists.

NOTE For more information about connecting, see *The Essential FDO*.

A connection can be made in either one or two steps:

- **One-step connection.** If the user sets the required connection properties and calls the connection object's `Open()` method once, the returned state is `FdoConnectionState_Open`, no additional information is needed.
- **Two-step connection.** If the user sets the required connection properties and calls the connection object's `Open()` method, the returned state is `FdoConnectionState_Pending`, additional information is needed to complete the connection. In this case, the first call to `Open()` has resulted in the retrieval of a list of values for a property that becomes a required property for the second call to the `Open()` method. After the user has selected one

of the values in the list, the second call to `Open()` should result in `FdoConnectionState_Open`.

Connecting to a data store by way of the MySQL or the ArcSDE provider, for example, can be done in either one or two steps. In the first step, the data store parameter is not required. If the user does not give the data store parameter a value, the FDO will retrieve the list of data store values from the data source so that the user can choose from them during the second step. Otherwise the user can give the data store a value in the first step, and assuming that the value is valid, the connection will be completed in one step.

For the purpose of this example, let's assume that the user has installed MySQL on his local machine. During the installation he was prompted to assign a password to the system administrator account whose name is 'root'. He set the password to 'test'.

The following steps are preliminary to establishing a connection:

- 1 Get the list of providers.

```
FdoPtr<FdoProviderRegistry> registry = (FdoProviderRegistry
*)FdoFeatureAccessManager::GetProviderRegistry();
FdoProviderCollection * providers = registry->GetProviders();
```

- 2 Get the display names for all of the providers in the registry. An example of a display name might be OSGeo FDO Provider for MySQL.

```
FdoStringP displayName;
FdoStringP internalName;
FdoPtr<FdoProvider> provider;
int count = providers->GetCount();
for(int i = 0; i < count; i++) {
    provider = providers->GetItem(i);
    internalName = provider->GetName();
    displayName = provider->GetDisplayName();
    // add displayName to your list
}
```

- 3 Use the display names to create a menu list, from which the user will select from when making a connection.

After the user initiates a provider display name from the connection menu, do the following:

- 1 Loop through the providers in the registry until you match the display name selected by the user from the connection menu with a provider display name in the registry and retrieve the internal name for that provider. An example of an internal could be OSGeo.MySQL.3.2.

```
FdoStringP internalName = provider->GetName();
```

- 2 Get an instance of the connection manager.

```
FdoPtr<FdoConnectionManager> connectMgr = (FdoConnectionManager  
*)FdoFeatureAccessManager::GetConnectionManager();
```

- 3 Call the manager's CreateConnection() method using the provider internal name as an argument to obtain a connection object.

```
FdoPtr<FdoIConnection> fdoConnection = connectMgr->CreateConnec  
tion(L"OsGeo.MySQL.3.2");
```

- 4 Obtain a connection info object by calling the connection object's GetConnectionInfo() method.

```
FdoPtr<FdoIConnectionInfo> info = fdoConnection->GetConnection  
Info();
```

- 5 Obtain a connection property dictionary object by calling the connection info object's GetConnection Properties() method and use this dictionary to construct a dialog box requesting connection information from the user.

```
FdoPtr<FdoIConnectionPropertyDictionary> ConnDict = info->GetCon  
nectionProperties();
```

- 6 Get a list of connection property names from the dictionary and use this list to get information about the property. The following code loops through the dictionary getting all of the possible information.

NOTE An attempt to get the values of an enumerable property is made only if the property is required.

```
FdoInt32 count = 0;  
FdoString ** names = NULL;  
FdoStringP name;  
FdoStringP localname;  
FdoStringP val;  
FdoStringP defaultVal;  
bool isRequired = false;  
bool isProtected = false;  
bool isFilename = false;  
bool isFilepath = false;
```

```

bool isDatastorename = false;
bool isEnumerable = false;
FdoInt32 enumCount = 0;
FdoString ** enumNames = NULL;
FdoStringP enumName;
names = ConnDict->GetPropertyNames(count);
for(int i = 0; i < count; i++) {
    name = names[i];
    val = dict->GetProperty(name);
    defaultVal = dict->GetPropertyDefault(name);
    localname = dict->GetLocalizedName(name);
    isRequired = dict->IsPropertyRequired(name);
    isProtected = dict->IsPropertyProtected(name);
    isFilename = dict->IsPropertyFileName(name);
    isFilepath = dict->IsPropertyFilePath(name);
    isDatastorename = dict->IsPropertyDatastoreName(name);
    isEnumerable = dict->IsPropertyEnumerable(name);
    if (isEnumerable) {
        if (isRequired) {
            enumNames = dict->EnumeratePropertyValues(name, enumCount);
            for(int j = 0; j < enumCount; j++) {
                enumName = enumNames[j];
            }
        }
    }
}
}

```

- 7 Use the `GetLocalizedName` method to obtain the name of the property to present to the user. Calls to dictionary methods need the value of the internal name in the string array returned by `GetPropertyNames()`. So when the user selects the localized name in the menu, the program must map the localized name to the internal name.
- 8 Use the `IsPropertyRequired` method to determine whether to mark the line as either required or optional; the dialog box handler should not permit the user to click OK in the dialog box unless a required field has a value.
- 9 Use the `IsPropertyProtected` method to determine whether the dialog box handler should process the field value as protected data, for example, a password.
- 10 Use the `IsPropertyEnumerable` and `IsRequired` methods to determine whether to call the `EnumeratePropertyValues` method to get a list of valid values.

NOTE Call the EnumeratePropertyValues method only if both methods return true. Otherwise be prepared to catch an exception if there is no pending connection. The assumption is that a connection exists and the program is retrieving values from the data store.

As shown in the code lines above, the EnumeratePropertyValues method takes a property name and an updates integer argument and returns a string array. The updates integer will say how many values are in the returned array. Present the list of choices to the user.

If the property is not enumerable, present the values returned by either the GetProperty or GetPropertyDefault methods to the user.

Now that the user has seen the set of properties in the dictionary, s/he can set the required properties. A property is set by calling the dictionary's SetProperty method. The MySQL connection property names are Username, Password, Service, and DataStore. The dictionary tells us that Username, Password, and Service are required properties and that DataStore is not required. Let's connect to the MySQL as root.

```
ConnDict->SetProperty(L"Username", L"root");  
ConnDict->SetProperty(L"Password", L"test");  
ConnDict->SetProperty(L"Service", L"localhost");
```

NOTE fdoconnection->GetConnectionString() returns
Username=root;Password=test;Service=localhost;.
fdoconnection->SetConnectionString(L"Username=root;Password=test;Service=localhost;");
would set the connection properties to the same values as the three calls above
to the connection dictionary's SetProperty() method.

Open the connection.

```
FdoConnectionState state = fdoConnection->Open();
```

The value of state is FdoConnectionState_Pending. An examination of the connection dictionary will reveal that the DataStore property is now required.

When the user checks the command capabilities, he discovers that he can create a data store.

```

FdoPtr<FdoICommandCapabilities> commandCapabilities = fdoConnection->GetCommandCapabilities();
bool bSupportsCreateDataStore = false;
FdoInt32 numCommands;
FdoInt32 * commands = commandCapabilities->GetCommands(numCommands);
for(int i = 0; i < numCommands; i++) {
    switch(commands[i]) {
        case FdoCommandType_CreateDataStore : bSupportsCreateDataStore = true;
    }
}

```

He can use the pending connection to MySQL to create the datastore. Use the connection object to create the `FdoICreateDataStore` command object. Use the command object to create the `FdoIDataStorePropertyDictionary` object and find out from this object what properties you must define. Use the dictionary object to set the required properties and then execute the command to create the 'fdo_user' data store. The only required property is `DataStore`.

NOTE The `FdoIDataPropertyDictionary` and the `FdoIConnectionPropertyDictionary` classes are both derived from `FdoIPropertyDictionary`. The code used above to access the `FdoIConnectionPropertyDictionary` object works for the `FdoIDataPropertyDictionary`.

```

FdoPtr<FdoICreateDataStore> createDataStoreCmd = dynamic_cast<FdoICreateDataStore *> (fdoConnection->CreateCommand(FdoCommandType_CreateDataStore));
FdoPtr<FdoIDataStorePropertyDictionary> createDsDict = createDataStoreCmd->GetDataStoreProperties();
createDsDict->SetProperty(L"DataStore", L"fdo_user");
createDataStoreCmd->Execute();

```

Now use the connection property dictionary to set the `DataStore` property to 'fdo_user' and call the `Open()` method on the connection object. This method should return `FdoConnectionState_Open`.

FDO Capabilities

6

FDO Capabilities

Introduction

The FDO capabilities are statically defined. They are accessed through a connection object in the closed state. The following code illustrates how to get a connection object in the closed state.

If you are using C#, you will find the capability definitions in the `OSGeo.FDO.Connections.Capabilities` namespace. If you are using C++, you will find the headers in the `Fdo/Connections/Capabilities` directory.

In C# you access them using the `OSGeo.FDO.Connections.IConnection` object. In C++ you access them using the `IConnection` object whose header is in the `Fdo/Connections` directory.

The information in the capabilities tables reflects the state of the FDO provider capabilities as of the commercial release of Map 3D 2009.

For the most part FDO defines capabilities in three ways: booleans, enumerations or integers representing enumerated values, and collections. One schema capability is defined as a string and two other schema capabilities as integers. For example, the `Command` capability, `SupportsParameters`, is a boolean, and the `Command` capability, `Commands`, is a set of integers whose meaning is defined by the C# `CommandType` enumeration and the C++ `FdoCommandType` enumeration. Function capabilities are defined as collections of function signatures. The C# type is

`OSGeo.FDO.Connections.Capabilities.ReadOnlySignatureDefinitionCollection`.

The C++ type is `FdoReadOnlySignatureDefinitionCollection`. The latter is

defined in Fdo/Connections/Capabilities/SignatureDefinition.h. Each boolean, enumeration and signature value is shown in the tables.

In general, where a number of capabilities are supported by the same set of providers, the capabilities are aggregated into one line in the table. This rule is modified in the function category tables in the Expression section.

In some cases, for example the CLIP raster function, there is only one signature. At the other end of the spectrum is the Concat string function, which has almost 100 signatures. The same set of providers supports an enormous number of functions and so the rule for presenting aggregations of capabilities had to be modified for functions. The rule for functions is to present all of the signatures for one function on a line in the table. A handful of functions had so many signatures that their presentation had to be split over two lines.

Provider Type

A provider can be one of four types: Database Server, File, Web Server, or Unknown. The type information is available from the connection object in the closed state. The “Unknown” type indicates that the provider can connect to more than one type of data store technology. The ODBC and OGR providers can connect either to a file-based or RDBMS-based data store technology, which is not known until the connection is in the open state.

In C# the ProviderDatastoreType enumeration is in the OSGeo.FDO.Connections namespace. The ProviderDatastoreType property is accessible through the OSGeo.FDO.Connections.IConnection object (`connection.ConnectionInfo.ProviderDatastoreType`).

In C++ the FdoProviderDatastoreType enumeration is defined in the `<Fdo/Connections/ProviderDatastoreType.h>` header file. The FdoProviderDatastoreType value is accessible through the IConnection object (`connection->GetConnectionInfo()->GetProviderDatastoreType()`).

| Provider | Type |
|------------------------|-----------------|
| Autodesk.Oracle.3.3 | Database Server |
| Autodesk.Raster.3.3 | File |
| Autodesk.SqlServer.3.3 | Database Server |
| OSGeo.ArcSDE.3.3 | Database Server |

| Provider | Type |
|----------------------------|-----------------|
| OSGeo.Gdal.3.3 | File |
| OSGeo.KingOracle.3.3 | Database Server |
| OSGeo.MySQL.3.3 | Database Server |
| OSGeo.ODBC.3.3 | Unknown |
| OSGeo.OGR.3.3 | Unknown |
| OSGeo.PostGIS.3.3 | Database Server |
| OSGeo.SDF.3.3 | File |
| OSGeo.SHP.3.3 | File |
| OSGeo.SQLServerSpatial.3.3 | Database Server |
| OSGeo.WFS.3.3 | Web Server |
| OSGeo.WMS.3.3 | Web Server |

NOTE A provider name with Autodesk at the beginning indicates that the provider is available by purchasing a commercial license of an Autodesk product such as Map 3D or MapGuide. A provider name with OSGeo at the beginning indicates that the provider is available as open source from <http://fdo.osgeo.org>.

Command

In C# you access the

`OSGeo.FDO.Connections.Capabilities.ICommandCapabilities` using the `IConnection` object (`connection.CommandCapabilities`).

In C++ you access the `FdoICommandCapabilities` using the `IConnection` object (`connection->GetCommandCapabilities()`).

| Command | Description |
|-------------|---|
| AcquireLock | Locks feature instances of a given class that match the specified criteria. |

| Command | Description |
|---------------------------|--|
| ActivateLongTransaction | Activates a long transaction so that features can be versioned. |
| ActivateSpatialContext | Activates a specified spatial context. |
| ApplySchema | Creates or updates a feature schema within the data store. Optionally, a provider-specific mapping of feature schema elements to physical storage can be specified. |
| CommitLongTransaction | Commits all of the data within a leaf long transaction (one that does not have descendent long transactions) to the parent long transaction and removes the long transaction. Requires access, commit and remove privileges to the long transaction and the access privilege to the parent long transaction. |
| CreateDataStore | Creates a new provider specific datastore. |
| CreateLongTransaction | Creates a long transaction as a child of the currently active long transaction. |
| CreateSpatialContext | Creates a new spatial context. Input to the command includes the name, description, coordinate system, extent type, and extent for the new context. |
| DeactivateLongTransaction | Deactivates the active long transaction and automatically activates the root long transaction. If the root long transaction is the active one, the deactivation has no effect. |
| Delete | Deletes the instances of a given class that match the specified criteria. The instances can be at global scope or contained within an object collection. |
| DescribeSchema | Describe a single schema or all schemas available from the connection. |
| DescribeSchemaMapping | Describes the logical to physical schema mappings for one or all feature schemas available from the connection. |
| DestroyDataStore | Remove a data store. |

| Command | Description |
|----------------------------|--|
| DestroySchema | Destroys a schema definition, including all class definitions, relationship definitions, and instance data within it. If elements in other schemas refer to the schema to be destroyed, execution will fail. |
| DestroySpatialContext | Destroys an existing spatial context, which destroys all data stored in that context. |
| GetLockedObjects | Gets a list of all objects that are currently locked by a particular user. |
| GetLockInfo | Gets lock information for the feature instances of a given class that match the specified filter. |
| GetLockOwners | Gets a list of all lock owners. |
| GetLongTransactions | Gets one specific long transaction or all available long transactions. |
| GetMeasureUnits | Enumerates the existing measurement units. |
| GetSpatialContexts | Gets all spatial contexts or just the active one. |
| Insert | inserts a new instance of a given class. The instance can be at global scope or contained within an object collection. |
| ListDataStores | Gets a list of datastores at a particular server. |
| Oracle.CreateSpatialIndex | Create a spatial index for the specified spatial context. |
| Oracle.DestroySpatialIndex | Drop the specified spatial index. |
| Oracle.GetSpatialIndexes | Enumerate the existing spatial indexes. |
| ReleaseLock | Releases locks from feature instances of a given class that match the specified criteria. |
| RollbackLongTransaction | Removes all of the data within a long transaction and removes the long transaction as well. Requires access, |

| Command | Description |
|---------------------------|---|
| | commit and remove privileges to the long transaction and the access privilege to the parent long transaction. |
| SDF.CreateSDFFile | creates a new SDF+ file with a specified location and file name and a given spatial coordinate system. This command has been superceded by the CreateDataStore command. |
| SDF.Extended_Select | Supports the selection and ordering of feature instances of a given class according to the specified criteria. The results of the select are written to an SDF file. |
| Select | Queries for features of a given class that match the specified criteria. |
| SelectAggregates | Queries for features of a given class that match the specified criteria. The criteria can specify that feature values from multiple instances be aggregated, be distinct, or be grouped. |
| SQLCommand | Supports the execution of a SQL statement against an underlying RDBMS. Two execute methods are provided to distinguish between statements that return table data versus those that execute non query type operations. |
| Update | Modifies instances of a given class that match the specified criteria. The instance can be at global scope or contained within an object collection. |
| WMS.GetImageFormats | Gets the image formats supported by the connection. |
| WMS.GetFeatureClassStyles | Gets the layer styles available for the specified feature class. |

| Command | Description |
|---------------------------|---|
| WMS.FeatureClassCRS-Names | Gets the CRS names available for the specified feature class. |

| Boolean Capability | Description |
|---------------------------|---|
| SupportsSelectDistinct | Indicates whether or not a provider supports the use of distinct in SelectAggregates commands. |
| SupportsSelectExpressions | Indicates whether or not a provider supports the use of expressions for properties in Select and SelectAggregate commands. |
| SupportsSelectFunctions | Indicates whether or not a provider supports the use of functions in Select and SelectAggregates command. Only aggregate functions can be used in the SelectAggregates command. |
| SupportsSelectGrouping | Indicates whether or not a provider supports the use of grouping criteria in the SelectAggregates command. |
| SupportsSelectOrdering | Indicates whether or not a provider supports ordering in the Select and SelectAggregates command. |
| SupportsParameters | Indicates whether or not a provider supports parameter substitution. |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Select, DescribeSchema, GetSpatialContexts | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| SelectAggregates | + | + | + | + | + | + | + | + | + | + | - | + | + | + | + |
| SupportsSelectFunctions | + | + | + | + | + | + | + | + | + | + | + | - | + | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLite 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SupportsSelectExpressions | + | + | + | + | + | + | + | + | + | - | + | + | - | - | - |
| SupportsSelectDistinct, Insert, Update, Delete | + | + | + | + | + | + | + | + | + | + | + | - | - | - | - |
| DescribeSchemaMapping | - | + | + | - | + | + | + | - | + | - | - | - | + | + | + |
| CreateSpatialContext | + | + | - | - | + | + | + | + | + | + | - | - | - | - | - |
| SQLCommand | - | - | + | - | + | + | + | + | + | + | + | - | - | - | - |
| ApplySchema | + | + | - | - | + | + | + | + | + | - | + | - | - | - | - |
| SupportsSelectOrdering | - | - | + | - | + | + | + | + | + | - | + | - | - | - | - |
| CreateDataStore | + | - | - | - | + | + | + | + | + | - | + | - | - | - | - |
| ListDataStores | - | - | - | - | + | + | + | - | + | + | + | - | - | - | - |
| DestroyDataStore | + | - | - | - | + | + | + | - | + | - | + | - | - | - | - |
| SupportsSelectGrouping | - | - | + | - | + | + | + | + | + | - | - | - | - | - | - |
| DestroySchema | - | + | - | - | + | + | + | - | + | - | - | - | - | - | - |
| DestroySpatialContext | - | - | - | - | + | + | + | - | + | + | - | - | - | - | - |
| ActivateSpatialContext | - | - | - | - | + | + | + | - | - | + | - | - | - | - | - |
| SupportsParameters | - | - | - | - | - | - | - | + | - | + | + | - | - | - | - |
| Oracle.GetSpatialIndexes, Oracle.CreateSpatialIndex, Oracle.DestroySpatialIndex | - | - | - | - | - | + | + | - | - | - | - | - | - | - | - |
| SDF.CreateSDFFile | + | + | - | - | - | - | - | - | - | - | - | - | - | - | - |
| AcquireLock, CreateLongTransaction, DeactivateLongTransaction, RollbackLongTransaction, CommitLongTransaction, GetLockOwn- | - | - | - | - | - | - | + | - | - | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLite 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ers, ReleaseLock, GetLongTransactions, GetLockedObjects, ActivateLongTransaction, GetLockInfo | | | | | | | | | | | | | | | |
| SDF.Extended_Select | + | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| GetMeasureUnits | - | - | - | - | - | - | - | - | - | - | - | - | - | + | - |
| WMS.FeatureClassCRSNames, WMS.GetImageFormats, WMS.GetFeatureClassStyles | - | - | - | - | - | - | - | - | - | - | - | + | - | - | - |

Connection

In C# you access the `OSGeo.FDO.Connections.Capabilities.IConnectionCapabilities` using the `IConnection` object (`connection.ConnectionCapabilities`).

In C++ you access the `FdoIConnectionCapabilities` using the `IConnection` object (`connection->GetConnectionCapabilities()`).

| Capability | Description |
|--------------------------------------|---|
| LockType_AllLongTransactionExclusive | Indicates that only this user can modify this object in this long transaction. No user, not even the user locking the object, can modify the object in any long transaction created as a descendent of the one containing the object being locked. When not in a long transaction situation (for example, if only a root long transaction exists), the lock behaves like an Exclusive lock. |
| LockType_Exclusive | Indicates that only this user can modify this object. In a long transaction situation, any user can modify the object in any other long transaction, including the root long transaction if it is not the current long transaction. |
| LockType_LongTransactionExclusive | Indicates that only this user can modify this object in the long transaction containing the object or any long transaction created as a descendent of that one. When not in a |

| Capability | Description |
|----------------------------------|---|
| | long transaction situation (for example, if only a root long transaction exists), the lock behaves like an Exclusive lock. |
| LockType_Shared | Indicates a shared lock type. |
| LockType_Transaction | Indicates that a transaction lock should be applied to an object. |
| SupportsConfiguration | Indicates whether or not a provider can specify on a closed connection an XML source to be used to configure the data store. |
| SpatialContextExtentType_Dynamic | Indicates that the spatial extent of the context is dynamic and changes as data is added and removed from the context. |
| SpatialContextExtentType_Static | Indicates that the spatial extent of the context is static and must be specified when the context is created. |
| SupportsCSysWKTFromCSysName | Indicates whether or not a provider supports the specification of a coordinate system during the creation of a spatial context using only the coordinate system name. The provider maps the name to the WKT internally. |
| SupportsFlush | Indicates whether or not a provider supports the forced write of cached data associated with a connection to the target data store. |
| SupportsLocking | Indicates whether or not a provider supports the set of locking commands: acquire, get locked objects, get lock owners, get lock info, and release. |
| SupportsLongTransactions | Indicates whether or not a provider supports the set of long transaction commands: activate, commit, create, deactivate, and get. |
| SupportsMultipleSpatialContexts | Indicates that the data store can contain multiple spatial contexts but the provider does not necessarily support the create spatial context command. |
| SupportsSQL | Indicates whether or not a provider supports the SQL command. |

| Capability | Description |
|---|---|
| SupportsTransactions | Indicates whether or not a provider supports the creation of a transaction associated with a connection to a data store. |
| ThreadCapability_PerConnectionThreaded | Indicates that the provider supports a single thread per connection. Multiple concurrent threads cannot access the same connection object and only one command can be executing per connection. Multiple connections can be active concurrently as long as each is executing on its own thread. |
| ThreadCapability_PerCommandThreaded | Indicates that the provider supports a single thread per command. Multiple concurrent threads cannot access the same command object; however, multiple commands can be executing concurrently against a single connection. |
| ThreadCapability_SingleThreaded | Indicates that the provider is not thread safe. |
| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | |
| Capability | 1 2 3 4 5 6 7 8 9 A B C D E F |
| SpatialContextExtentType_Static | - + + - + + + + + + + + + |
| ThreadCapability_PerConnectionThreaded | + + + + + + - + + + + + - |
| SupportsMultipleSpatialContexts | - + - - + + + + + + + + + |
| SupportsSQL | - - + - + + + + + + - - - |
| SupportsConfiguration | - + + - - + + - - - + + + |
| SupportsTransactions | - - - - + + + - - + + - - - |
| LockType_Exclusive | - - - - - + + - + - - - - |
| SupportsFlush | + + - - - + - - - - - - |
| LockType_Transaction | - - - - - + + - + - - - - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SpatialContextExtentType_Dynamic | + | - | - | + | - | - | - | - | - | - | - | - | - | - | - |
| LockType_AllLongTransactionExclusive, LockType_LongTransactionExclusive, LockType_Shared | - | - | - | - | - | - | + | + | - | - | - | - | - | - | - |
| SupportsLocking, SupportsCSysWKTFromCSysName, SupportsLongTransactions | - | - | - | - | - | - | + | - | - | + | - | - | - | - | - |
| ThreadCapability_SingleThreaded | - | - | - | - | - | - | - | - | - | - | - | - | - | - | + |
| ThreadCapability_PerCommandThreaded | - | - | - | - | - | - | - | + | - | - | - | - | - | - | - |

Expression

In C# you access the

`OSGeo.FDO.Connections.Capabilities.IExpressionCapabilities` and the `OSGeo.FDO.Connections.Capabilities.FunctionDefinitionCollection` using the `IConnection` object, `connection.ExpressionCapabilities`, and `connection.ExpressionCapabilities.Functions` respectively. Each `OSGeo.FDO.Connections.Capabilities.FunctionDefinition` in the collection has has a

`OSGeo.FDO.Connections.Capabilities.ReadOnlySignatureDefinitionCollection`, which is accessed by `functionDef.Signatures`.

In C++ you access the `FdoIExpressionCapabilities` and the

`FdoFunctionDefinitionCollection` using the `IConnection` object, `connection->GetExpressionCapabilities()`,

`expression_capabilities->GetFunctions()` respectively. Each

`FdoFunctionDefinition` in the collection has an

`FdoReadOnlySignatureDefinitionCollection`, which is accessed by `functionDef->GetSignatures()`.

The functions can be broadly divided into two groups according to the mode of their implementation. The largest group contains the functions whose underlying implementation is based on the Expression Builder. The first release of the Expression Builder component is in FDO 3.3. The remaining functions

are those whose implementation was not touched by the introduction of the Expression Builder component.

The functions whose implementation is based on the Expression Builder are further categorized for UI purposes into 7 groups: aggregate, conversion, date, geometry, math, numeric, and string. These UI categories are used in this document to group the presentation of the functions.

General Capabilities

| Capability | Description |
|--------------------------|--|
| ExpressionType_Basic | Indicates whether or not the provider supports basic arithmetic expressions, for example, +, -, *, /, negate, and string concatenation. |
| ExpressionType_Function | Indicates whether or not the provider supports function evaluations. |
| ExpressionType_Parameter | Indicates whether or not the provider supports parameter substitution. An expression can be constructed using named parameters and values bound to the parameter at a later time. This allows the constructed expression to be reused. |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ExpressionType_Function | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| ExpressionType_Basic | + | + | + | + | + | + | + | + | + | + | + | + | - | - | - |
| ExpressionType_Parameter | - | - | - | - | + | + | + | - | + | - | - | - | - | - | - |

Argument and Return Types

The following tables specify the function name, arguments, and the type of the return value. An argument is optional if it is enclosed in square brackets. An argument may be a literal such as "ALL" or "DISTINCT" or a feature property name. Where the argument value is a feature property name, the type of the feature property is specified. The set of valid alternatives for an

argument value is indicated by enclosing the set of OR'd property types or literal values in parentheses. For brevity's sake, a 'numeric' property type indicates the set of numeric property types, namely {Byte | Decimal | Double | Int16 | Int32 | Int 64 | Single}. The other property types, BLOB, Boolean, CLOB, DateTime, Geometry, and String, are indicated directly in the argument specification.

Some functions have more than one return type. The general rule is that the return type matches the argument type.

Aggregate Expression Functions

These functions can be used with the SelectAggregates command.

| Function Name | Description |
|---------------|--|
| Avg | Returns a double which is the average of the values identified by the provided expression. Avg ({ALL DISTINCT},] numeric) |
| Count | Returns an Int64 which is the number of rows returned by a query identified by the provided expression. Count ({ALL DISTINCT},] {bool DateTime numeric string blob clob geometry}) |
| Max | Returns a DateTime, numeric or string in agreement with the type of the argument, which is the maximum value of the provided expression. Max ({ALL DISTINCT},] { DateTime numeric string}) |
| Median | Represents an inverse distribution function that assumes a continuous distribution model. It takes a numeric or date-time value and returns a Double, which is the middle value or an interpolated value that would be the middle value once the values are sorted. Median (numeric) |
| Min | Returns a DateTime, numeric or string in agreement with the type of the argument, which is the minimum value of the provided expression. Min ({ALL DISTINCT},] { DateTime numeric string}) |
| Stddev | Returns a Double, which is the sample standard deviation of the provided expression. Stddev ({ALL DISTINCT},] numeric) |
| Sum | Returns a Double, which is the sum of the values identified by the provided expression. Sum ({ALL DISTINCT},] numeric) |

| Function Name | Description |
|----------------|--|
| Spatial Extent | Returns a byte array, which yields the spatial extent of a set of geometries. The argument is the name of geometry property. SpatialExtent (<geometries>). For example, given these geometries, POINT(1 1), LINESTRING(01, 21), and POLYGON((01, 21, 12, 01)), it returns POLYGON((01 21, 22 02, 01)). |

NOTE The median is selected or calculated by doing the following:

- 1 Sort the sample in ascending order.
- 2 Calculate the Row of Interest (roi) using the formula: $roi = (1 + ((sample_size - 1) / 2))$.
- 3 Compare three values: roi, Ceil(roi) and Floor(roi). If they are equal, the median is the value in the Row Of Interest. If they are not equal, the median is calculated by averaging the values in the rows preceding and succeeding the Row of Interest.

NOTE The value returned by the StdDev function is known as the sample standard deviation and represents an estimate of the standard deviation of a population based on a random sample of that population. See the Wikipedia entry for further discussion and a definition of the formula for sample standard deviation:
http://en.wikipedia.org/wiki/Standard_deviation.

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Double Max(Double) | + | + | + | + | + | + | + | + | + | + | + | - | - | - | - |
| Double Min(Double) | + | + | + | + | + | + | + | + | + | + | + | - | - | - | - |
| Double Sum(Double) | + | + | + | + | + | + | + | + | + | + | + | - | - | - | - |
| BLOB SpatialExtents(geometry) | + | + | + | - | + | + | + | + | + | + | - | + | - | - | - |
| Avg ([{ALL DISTINCT},] numeric), Double Avg(Byte Decimal Int16 Int32 Int64 Single), Double Avg(ALL DISTINCT,Byte Decimal Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLite 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Count ([{ALL DISTINCT},] {bool datetime numeric string blob clob} geometry), Int64 Count(BLOB Boolean Byte CLOB DateTime Decimal Double Int16 Int32 Single String), Int64 Count(ALL DISTINCT, Boolean Byte DateTime Decimal Double Int16 Int32 Single String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Max ([{ALL DISTINCT},] { DateTime numeric string}), Byte Max(Byte), Byte Max(ALL DISTINCT, Byte), DateTime Max(DateTime), DateTime Max(ALL DISTINCT, DateTime), Decimal Max(Decimal), Decimal Max(ALL DISTINCT, Decimal), Double Max(ALL DISTINCT, Double), Int16 Max(Int16), Int16 Max(ALL DISTINCT, Int16), Int32 Max(Int32), Int32 Max(ALL DISTINCT, Int32), Int64 Max(Int64), Int64 Max(ALL DISTINCT, Int64), Single Max(Single), Single Max(ALL DISTINCT, Single), String Max(String), String Max(ALL DISTINCT, String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Median(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Min ([{ALL DISTINCT},] { DateTime numeric string}), Byte Min(Byte), Byte Min(ALL DISTINCT, Byte), DateTime Min(DateTime), DateTime Min(ALL DISTINCT, DateTime), Decimal Min(Decimal), Decimal Min(ALL DISTINCT, Decimal), Double Min(ALL DISTINCT, Double), Int16 Min(Int16), Int16 Min(ALL DISTINCT, Int16), Int32 Min(Int32), Int32 Min(ALL DISTINCT, Int32), Int64 Min(Int64), Int64 Min(ALL DISTINCT, Int64), Single Min(Single), Single Min(ALL DISTINCT, Single), String Min(String), String Min(ALL DISTINCT, String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Stddev(Byte Decimal Double Int16 Int32 Int64 Single), Double Stddev(ALL DISTINCT, Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Double Sum(Byte Decimal Double Int16 Int32 Int64 Single), Double Sum(ALL DISTINCT,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

Conversion Expression Functions

| Function Name | Description |
|---------------|---|
| NullValue | Evaluates two expressions and returns the first one if it does not evaluate to NULL, the second otherwise. The signatures without function name are Boolean(Boolean,Boolean), Byte(Byte,Byte), DateTime(DateTime,DateTime), Decimal(Decimal,Decimal), Double(Decimal,Double), Decimal(Decimal,Int16 Int32), Double(Decimal,Int64), Single(Decimal,Single), Double(Double,Decimal Double Int16 Int32 Int64 Single), Double(Int16,Decimal Double), Int16(Int16,Int16), Int32(Int16,Int32), Int64(Int16,Int64), Single(Int16,Single), Double(Int32,Decimal Double), Int32(Int32,Int16 Int32), Int64(Int32,Int64), Double(Int32,Single)Double(Int64,Decimal Double Single), Int64(Int64,Int16 Int32 Int64), Double(Single,Decimal Double Int32 Int64), Single(Single,Int16 Single), String(String,Decimal Double Int16 Int32 Int64 Single String) |
| ToDate | Converts a string with date/time information to a date. ToDate (date_string [, format_string]) The default format_string is "DD MON YYYY h24 mm ss". |
| ToDouble | Converts a numeric or string expression to a double. ToDouble ({numeric string}) |
| ToFloat | Converts a numeric or string expression to a float. ToFloat ({numeric string}) |
| ToInt32 | Converts a numeric or string expression to an int32. ToInt32 ({numeric string}) |

| Function Name | Description |
|---------------|--|
| ToInt64 | Converts a numeric or string expression to an int64. ToInt64 ({numeric string}) |
| ToString | Converts a numeric or date expression to a string. ToString(numeric), ToString (DateTime [, format_string]). The default format_string is "DD-MON-YYYY h24:mm:ss". |

The ToDate function takes a string value representing date and/or time information and converts it to a date object. The optional format specification parameter defines the format used in the string to represent the date using the abbreviations described in the following table. For example, for a string containing the date April 2, 1998 the format specification should contain Month DD, YYYY.

The ToString function takes a date value and creates a representation of it in a string. The optional format specification parameter allows the user to define the structure of the string to be created using the abbreviations described in the following table.

| Abbreviation | Description |
|--------------|---|
| YY | Indicates that a year is defined as a two digit number (example: 07). |
| YYYY | Indicates that a year is defined as a four digit number (example: 2007). |
| MONTH | Indicates that a month is defined by its name, all in uppercase letters (example: APRIL). |
| month | Indicates that a month is defined by its name, all in lowercase letters (example: april). |
| Month | Indicates that a month is defined by its name with the first letter being an uppercase letter (example: April). |
| MON | Indicates that a month is defined by its abbreviation, all in uppercase letters (example: APR). |
| mon | Indicates that a month is defined by its abbreviation, all in lowercase letters (example: apr). |
| MM | Indicates that a month is defined by its number (example: 04). |
| DD | Indicates that a day is defined by its number (example: 04). |

| Abbreviation | Description |
|---|---|
| hh24 | Indicates that a hour is defined by its number in the range [0-24] |
| hh12 | Indicates that a hour is defined by its number in the range [1-12] |
| hh | Represents a default representation of an hour. Defaults to hh24. |
| mm | Indicates a minute definition |
| ss | Indicates a second definition |
| ms | Indicates a millisecond definition |
| am pm | The meridiem. Only considered if used with the time range [1-12] (format hh12). |
| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | |
| Capability | 1 2 3 4 5 6 7 8 9 A B C D E F |
| Boolean NullValue(Boolean,Boolean), Byte NullValue(Byte,Byte), DateTime NullValue(DateTime,DateTime), Decimal NullValue(Decimal,Decimal Int16 Int32), Double NullValue(Decimal,Double Int64), Double NullValue(Double,Decimal Double Int16 Int32 Int64 Single), Double NullValue(Int16,Decimal Double), Double NullValue(Int32,Decimal Double Single), Double NullValue(Int64,Decimal Double Single), Double NullValue(Single,Decimal Double Int32 Int64), Int16 NullValue(Int16,Int16), Int32 NullValue(Int16,Int32), Int32 NullValue(Int32,Int16 Int32), Int64 NullValue(Int16,Int64), Int64 NullValue(Int32,Int64), Int64 NullValue(Int64,Int16 Int32 Int64), Single NullValue(Decimal,Single), Single NullValue(Int16,Single), Single NullValue(Single,Int16 Single), String NullValue(String,Decimal Double Int16 Int32 Int64 Single String) | + + + - + + + - + + - - - - |
| DateTime ToDate(String), DateTime ToDate(String,String) | + + + - + + + - + + - - - - |
| Double ToDouble(Byte Decimal Double Int16 Int32 Int64 Single String) | + + + - + + + - + + - - - - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Single ToFloat(Byte Decimal Double Int16 Int32 Int64 Single String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Int32 ToInt32(Byte Decimal Double Int16 Int32 Int64 Single String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Int64 ToInt64(Byte Decimal Double Int16 Int32 Int64 Single String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| ToString(numeric), ToString (DateTime [, format_string]), String ToString(Byte DateTime Decimal Double Int16 Int32 Int64 Single), String ToString(DateTime,String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

Date Expression Functions

| Function Name | Description |
|---------------|---|
| AddMonths | Adds a specified number of months to a given date expression and returns a DateTime. AddMonths (date_time, numeric) |
| CurrentDate | Returns the current date. CurrentDate () |
| Extract | Extracts a specified portion of a date and returns a DateTime. Extract ({YEAR MONTH DAY HOUR MINUTE SECOND}, date_time) |
| MonthsBetween | Calculates the number of months between two provide date expressions and returns |

| Function Name | Description |
|--|--|
| | a Double. MonthsBetween (date_time, date_time) |
| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | |
| Capability | 1 2 3 4 5 6 7 8 9 A B C D E F |
| DateTime AddMonths(DateTime,Byte Decimal Double Int16 Int32 Int64 Single) | + + + - + + + - + + - - - - |
| DateTime CurrentDate() | + + + - + + + - + + - - - - |
| Extract ((YEAR MONTH DAY HOUR MINUTE SECOND), date_time), DateTime Extract(String,DateTime) | + + + - + + + - + + - - - - |
| Double MonthsBetween(DateTime,DateTime) | + + + - + + + - + + - - - - |

Geometry Expression Functions

| Function Name | Description |
|--|--|
| Area2D | Returns a Double, which is the area of a geometry. Area2D (<geometry>) |
| Length2D | Returns a Double, which is the length of a geometry. Length2D (<geometry>) |
| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | |
| Capability | 1 2 3 4 5 6 7 8 9 A B C D E F |
| Double Area2D(BLOB) | + + + - + + + - + + - - - - |
| Double Length2D(BLOB) | + + + - + + + - + + - - - - |

Mathematical Expression Functions

| Function Name | Description |
|---------------|---|
| Abs | Returns the absolute value of a numeric expression. Abs (numeric). The return type is the same as the argument type. |
| Acos | Returns a Double, which is the arc cosine of a numeric expression. Acos (numeric) |
| Asin | Returns a Double, which is the arc sine of a numeric expression. Asin (numeric) |
| Atan | Returns a Double, which is the arc tangent of a numeric expression. Atan (numeric) |
| Atan2 | Returns a Double, which is the arc tangent based on two numeric expressions. Atan2 (numeric, numeric) |
| Cos | Returns a Double, which is the cosine of a numeric expression. Cos (numeric) |
| Exp | Returns a Double, which is e raised to the power of a numeric expression. Exp (numeric) |
| Ln | Returns a Double, which is the natural logarithm of a numeric expression. Ln (numeric) |
| Log | Returns a Double, which is the logarithm of a numeric expression using the provided base. Log (base_numeric, numeric) |
| Mod | Returns the remainder of the division of two numeric expressions. Mod (numeric, divisor_numeric). The implementation algorithm is $\text{Mod}(m, n) = (\text{sign}(m) * (\text{abs}(m) - (\text{abs}(n) * \text{Floor}(m/n))))$. mod(15,4) returns 3. mod(15,0) returns 15. mod(11.6,2) returns 1.6. mod(11.6,2.1) returns 1.1. mod(-15,4) returns -3. mod(-15,0) returns -15. The return signatures without the function name are: Byte(Byte,Byte Int16 Int32 Int64), Double(Byte,Decimal Double), Single(Byte,Single), Double(Decimal Double,Byte Decimal Double Int16 Int32 Int64 Single), Int16(Int16,Byte Int16 Int32 Int64), Double(Int16,Decimal Double), Single(Int16,Single), Int32(Int32,Byte), Double(Int32,Decimal Double), Int16(Int32,Int16), Int32(Int32,Int32 Int64), Single(Int32,Single), Int64(Int64,Byte Int64), |

| Function Name | Description |
|---------------|--|
| | Double(Int64,Decimal Double), Int16(Int64,Int16), Int32(Int64,Int32), Single(Int64,Single), Single(Single,Byte Int16 Int32 Int64 Single), Double(Single,Decimal Double) |
| Power | Returns a Double, which is the result of a numeric expression raised to the power of another numeric expression. Power (numeric, power_numeric) |
| Remainder | Returns the remainder of the division of two numeric expressions. Remainder (numeric, divisor_numeric). The implementation algorithm is $\text{Remainder}(m,n) = (\text{sign}(m) * (\text{abs}(m) - (\text{abs}(n) * \text{Round}(m/n))))$. remainder(15,6) returns 3. remainder(15,5) returns 0. remainder(15,4) returns -1. remainder(11.6,2) returns -0.4. remainder(11.6,2.1) returns -1. remainder(-15,4) returns 1. The signatures without function name are: Int16(Byte,Byte Int16), Double(Byte,Decimal Double), Int32(Byte,Int32), Int64(Byte,Int64), Single(Byte,Single), Double(Decimal Double,Byte Decimal Double Int16 Int32 Int64 Single), Int16(Int16,Byte Int16 Int32 Int64), Double(Int16,Decimal Double), Single(Int16,Single), Int32(Int32,Byte Int32 Int64), Double(Int32,Decimal Double), Int16(Int32,Int16), Single(Int32,Single)Int64(Int64,Byte Int64), Double(Int64,Decimal Double), Int16(Int64,Int16), Int32(Int64,Int32), Single(Int64,Single), Single(Single,Byte Int16 Int32 Int64 Single)) |
| Sin | Returns a Double, which is the sine of a numeric expression. Sin (numeric) |
| Sqrt | Returns a Double, which is the square root of a numeric expression. Sqrt (numeric) |

| Function Name | Description |
|---------------|--|
| Tan | Returns a Double, which is the tangent of a numeric expression. Tan (numeric) |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte Abs(Byte), Decimal Abs(Decimal), Double Abs(Double), Int16 Abs(Int16), Int32 Abs(Int32), Int64 Abs(Int64), Single Abs(Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Acos(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Asin(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Atan(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Atan2(Byte Decimal Double Int16 Int32 Int64 Single,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Cos(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Exp(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Ln(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Log (base_numeric, numeric), Double Log(Byte Decimal Double Int16 Int32 Int64 Single,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Mod (numeric, divisor_numeric), Double Mod(Byte Decimal Double Int16 Int32 Int64 Single,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Power (numeric, power_numeric), Double Power(Byte Decimal Double Int16 Int32 Int64 Single, Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Remainder (numeric, divisor_numeric), Double Remainder(Byte Decimal Double Int16 Int32 Int64 Single, Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Sin(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Sqrt(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Double Tan(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

Numeric Expression Functions

| Function Name | Description |
|---------------|---|
| Ceil | Returns the smallest integer greater than or equal to a numeric expression. The return type is the same as the argument type. Where the argument is a Decimal, Double, or Single, the returned value is an integral number. Ceil(numeric) |
| Floor | Returns the largest integer less than or equal to a numeric expression. The return type is the same as the argument type. Where the argument is a Decimal, Double, or Single, the returned value is an integral number. Floor (numeric) |
| Round | Returns the rounded value of a numeric expression. The return type is the same as the type of the first argument. Round (numeric [, number_of_decimals_numeric]). |
| Sign | Returns and Int32 which is -1 if the provided numeric expression evaluates to a value less than 0, 0 if the expression evaluates to 0 |

| Function Name | Description |
|---------------|---|
| | and 1 if the expression evaluates to a value bigger than 0. Sign (numeric). |
| Trunc | Truncates a numeric or date expression. The return type is the same as the type of the first argument. Trunc (date_time, {YEAR MONTH DAY HOUR MINUTE}). Trunc (numeric [, number_of_decimals_numeric]). |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Int64 Ceil(Int64) | + | + | + | - | + | + | + | + | + | + | - | - | - | - | - |
| Int64 Floor(Int64) | + | + | + | - | + | + | + | + | + | + | - | - | - | - | - |
| Byte Ceil(Byte), Decimal Ceil(Decimal), Double Ceil(Double), Int16 Ceil(Int16), Int32 Ceil(Int32), Single Ceil(Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Byte Floor(Byte), Decimal Floor(Decimal), Double Floor(Double), Int16 Floor(Int16), Int32 Floor(Int32), Single Floor(Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Round (numeric [, number_of_decimals_numeric]), Byte Round(Byte), Byte Round(Byte,Byte Decimal Double Int16 Int32 Int64 Single), Decimal Round(Decimal), Decimal Round(Decimal,Byte Decimal Double Int16 Int32 Int64 Single), Double Round(Double), Double Round(Double,Byte Decimal Double Int16 Int32 Int64 Single), Int16 Round(Int16), Int16 Round(Int16,Byte Decimal Double Int16 Int32 Int64 Single), Int32 Round(Int32), Int32 Round(Int32,Byte Decimal Double Int16 Int32 Int64 Single), Int64 Round(Int64), Int64 Round(Int64,Byte Decimal Double Int16 Int32 Int64 Single), Single Round(Single), Single Round(Single,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Int32 Sign(Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Trunc (date_time, {YEAR MONTH DAY HOUR MINUTE}), Trunc (numeric [, number_of_decimals_numeric]), Byte Trunc(Byte), Byte Trunc(Byte,Byte Decimal Double Int16 Int32 Int64 Single), DateTime Trunc(DateTime,String), Decimal Trunc(Decimal), Decimal Trunc(Decimal,Byte Decimal Double Int16 Int32 Int64 Single),Double Trunc(Double), Double Trunc(Double,Byte Decimal Double Int16 Int32 Int64 Single), Int16 Trunc(Int16), Int16 Trunc(Int16,Byte Decimal Double Int16 Int32 Int64 Single), Int32 Trunc(Int32), Int32 Trunc(Int32,Byte Decimal Double Int16 Int32 Int64 Single), Int64 Trunc(Int64), Int64 Trunc(Int64,Byte Decimal Double Int16 Int32 Int64 Single), Single Trunc(Single), Single Trunc(Single,Byte Decimal Double Int16 Int32 Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

String Expression Functions

All of these functions return a string.

| Function Name | Description |
|---------------|--|
| Concat | Returns the concatenation of two string expressions. Concat ({boolean date numeric string}, {boolean date numeric string}) |
| Instr | Returns an Int64, which is the position of a substring in a string expression. Instr(string,string). |
| Length | Returns an Int64, which is the length of a string expression. Length(string). |
| Lower | Converts all uppercase letters in a string expression into lowercase letters. string Lower (string) |

| Function Name | Description |
|---------------|---|
| Lpad | Pads a string expression to the left to a predefined string length. string Lpad (string, TotalLengthOfResult [, pad-string]). If used with two parameters only, the pad-string is a single space character. |
| Ltrim | Removes leading blanks from a string expression. string Ltrim (string). |
| Rpad | Pads a string expression to the right to a predefined string length. string Rpad (string, TotalLengthOfResult [, pad-string]). If used with two parameters only, the pad-string is a single space character. |
| Rtrim | Removes trailing blanks from a string expression. string Rtrim (string) |
| Soundex | Returns a string, which is the phonetic representation of a string expression. See the entry in Wikipedia for more information, http://en.wikipedia.org/wiki/Soundex . Soundex (string). Soundex('eight') returns "E230" and Soundex('expression') returns "E216". |
| Substr | Extracts a substring from a string expression. string Substr (string, start_pos [, length]) |
| Translate | Replaces a set of letters in a string. Translate (string, from_set_string, to_set_string). If string is "SQL*Plus User's Guide" and from_set_string is ' */', that is, '{space}{asterisk}{forward slash}{single quote}{single quote}' and the to_set_string is '___', that is, '{underscore}{underscore}{underscore}', then the returned string is 'SQL_PLUs_Users_Guide'. Translate looks for each character in the from_set_string in the string argument and replaces it with the corresponding character from the to_set_string. In this case two spaces are replace by underscores and an asterisk is replaced by n underscore. There is no forward slash in the target string to be replaced, and there is no replacement character in the to_set_string corresponding to the single apostrophe character in the from_set_string When there is no replacement character in the to_set_string corresponding to a character in the from_set_string, the character in the from_set_string is removed wherever it is found in the target string.. |

| Function Name | Description |
|---------------|--|
| Trim | Removes leading and/or trailing blanks from a string expression. Trim ([{BOTH LEADING TRAILING},] string). |
| Upper | Converts all lowercase letters in a string expression into uppercase letters. Upper (string). |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| String Concat(String,String) | + | + | + | - | + | + | + | + | + | + | + | - | - | - | - |
| String Concat(Boolean,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Byte,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(DateTime,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Decimal,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Double,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Int16,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Int32,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Int64,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(Single,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String), String Concat(String,Boolean Byte DateTime Decimal Double Int16 Int32 Int64 Single String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Int64 Instr(String,String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| Int64 Length(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Lower(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Lpad(<stringToBePadded>, <TotalLengthOfResult>[, <padString>]), String Lpad(String,Byte Decimal Double | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLite 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Int16 Int32 Int64 Single), String Lpad(String,Byte Decimal Double Int16 Int32 Int64 Single,String) | | | | | | | | | | | | | | | |
| String Ltrim(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Rpad(<stringToBePadded>, <TotalLengthOfResult>[, <padString>]), String Rpad(String,Byte Decimal Double Int16 Int32 Int64 Single), String Rpad(String,Byte Decimal Double Int16 Int32 Int64 Single,String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Rtrim(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Soundex(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Substr(String, <startPosition>[, <length>]), String Substr(String,Byte Decimal Double Int16 Int32, Int64 Single), Substr(String,Byte Decimal Double Int16 Int32, Int64 Single, Byte Decimal Double Int16 Int32, Int64 Single) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Translate(<stringToBeTranslated>, <charactersToBeTranslated>, <replacementCharacters>), String Translate(String,String,String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Trim(String), String Trim(BOTH LEADING TRAILING,String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Upper(String) | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| String Lower(String,String) | - | - | - | - | - | - | - | + | - | - | - | - | - | - | - |
| String Upper(String,String) | - | - | - | - | - | - | - | + | - | - | - | - | - | - | - |

Miscellaneous Expression Functions

The following functions were not affected by the introduction of the Expression Builder component in FDO 3.3.

| Function Name | Description |
|---------------|--|
| CLIP | Returns a geometry that is the subset of the geometry argument defined by the polygon specified by the coordinate arguments. CLIP(BLOB,Double,Double,Double,Double) |
| MOSAIC | Returns a geometry. MOSAIC(BLOB) |
| RESAMPLE | Returns a geometry. RES-AMPLE(BLOB,Double,Double,Double,Double,Int32,Int32) |
| SpatialExtent | Returns a geometry. SpatialExtent(BLOB). |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BLOB CLIP(BLOB,Double,Double,Double,Double) | - | - | - | - | - | - | - | - | - | - | - | - | + | + | + |
| BLOB RES-AMPLE(BLOB,Double,Double,Double,Double,Int32,Int32) | - | - | - | - | - | - | - | - | - | - | - | - | + | + | + |
| BLOB SpatialExtents(BLOB) | - | - | - | - | - | - | - | - | - | - | - | - | + | + | - |
| BLOB MOSAIC(BLOB) | - | - | - | - | - | - | - | - | - | - | - | - | - | + | + |

Filter

In C# you access the

OSGeo.FDO.Connections.Capabilities.IFilterCapabilities using the IConnection object (connection.FilterCapabilities).

In C++ you access the `FdoIFilterCapabilities` using the `IConnection` object (`connection->GetFilterCapabilities()`).

| Capability | Description |
|--------------------------------------|--|
| ConditionType_In | Tests if the value of a specified data property is within a given set of literal values. |
| ConditionType_Comparison | Tests if one expression is equal, not equal, greater than, less than, greater than or equal to, or less than or equal to another expression. |
| ConditionType_Distance | Tests whether the value of a geometric property is within or beyond a specified distance of a literal geometric value. |
| ConditionType_Like | Tests whether the value of a specified data property matches a specified pattern. |
| ConditionType_Null | Tests whether the value of a specified data property is null. |
| ConditionType_Spatial | Tests whether the value of a geometric property and a literal geometric value satisfy the spatial relationship implied by the operation. |
| DistanceOperations_Beyond | Tests whether the geometric property value lies beyond a specified distance of a literal geometric value. |
| DistanceOperations_Within | Tests whether the geometric property value lies within a specified distance of a literal geometric value. |
| SpatialOperations_Contains | Tests whether the geometric property value spatially contains the literal geometric value. |
| SpatialOperations_CoveredBy | Tests whether the geometric property value is covered by the interior and boundary of the given geometry. |
| SpatialOperations_Crosses | Tests whether the geometric property value spatially crosses the given geometry. |
| SpatialOperations_Disjoint | Tests whether the geometric property value is spatially disjoint from the given geometry. |
| SpatialOperations_EnvelopeIntersects | Tests whether the envelope of the referenced geometric property value spatially intersects the given geometry. |

| Capability | Description |
|---|---|
| SpatialOperations_Equals | Tests whether the geometric property value is spatially equal to the given geometry. |
| SpatialOperations_Inside | Tests whether the geometric property value is inside the interior of the given geometry, not touching the boundary. |
| SpatialOperations_Intersects | Tests whether the geometric property value spatially intersects the given geometry. |
| SpatialOperations_Overlaps | Tests whether the geometric property value spatially overlaps the given geometry. |
| SpatialOperations_Touches | Tests whether the geometric property value spatially touches the given geometry. |
| SpatialOperations_Within | Tests whether the geometric property value is spatially within the given geometry. |
| SupportsNonLiteralGeometricOperations | Tests whether the spatial and distance operations can be applied between two geometric properties. Returns false if spatial and distance operations can be applied only between a geometric property and a literal geometry. Returns true if spatial and distance operations can be applied |
| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | |
| Capability | 1 2 3 4 5 6 7 8 9 A B C D E F |
| SpatialOperations_EnvelopeIntersects, ConditionType_In, ConditionType_Spatial | + + + + + + + + + + - - + + |
| SpatialOperations_Intersects | + + + - + + + + + + - - + + |
| SpatialOperations_Inside | + + + - + + + + - + + - - + + |
| ConditionType_Null, ConditionType_Comparison, ConditionType_Like | + + + + + + + + + + - - - - |
| SpatialOperations_Within | + + + - - - + - + + - - + + |

| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | | | | | | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ConditionType_Distance | - | - | - | - | + | + | + | - | + | + | + | - | - | - | - |
| SpatialOperations_CoveredBy | - | - | - | - | - | - | + | + | - | + | + | - | - | - | - |
| SpatialOperations_Disjoint, SpatialOperations_Contains | + | - | - | - | - | - | - | + | - | + | + | - | - | - | - |
| DistanceOperations_Within | - | - | - | - | - | - | + | - | - | + | + | - | - | - | - |
| SpatialOperations_Crosses, SpatialOperations_Touches, SpatialOperations_Equals, SpatialOperations_Overlaps | - | - | - | - | - | - | - | + | - | + | + | - | - | - | - |
| DistanceOperations_Beyond | - | - | - | - | - | - | - | - | - | + | + | - | - | - | - |
| SupportsNonLiteralGeometricOperations | - | - | - | - | - | - | - | - | - | - | - | + | - | - | - |

Geometry

In C# you access the

`OSGeo.FDO.Connections.Capabilities.IGeometryCapabilities` using the `IConnection` object (`connection.GeometryCapabilities`).

In C++ you access the `FdoIGeometryCapabilities` using the `IConnection` object (`connection->GetGeometryCapabilities()`).

| 1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer 6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster F=OSGeo.Gdal | | | | | | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| FdoDimensionality_XY | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| GeometryType_Polygon, GeometryComponentType_Linear-Ring | + | + | - | + | + | + | + | + | + | + | + | + | + | - | - |
| GeometryType_Point | + | + | + | + | + | + | + | + | + | + | + | + | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLite 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GeometryType_MultiLineString, GeometryType_MultiPoint, GeometryType_LineString | + | + | - | + | + | + | + | + | + | + | + | + | - | - | - |
| GeometryType_MultiPolygon | + | - | - | + | + | + | + | + | + | + | + | + | - | - | - |
| GeometryComponentType_LineStringSegment | + | + | - | + | + | + | + | + | - | + | - | + | - | - | - |
| FdoDimensionality_M, FdoDimensionality_Z | + | + | - | - | + | - | + | + | - | + | + | + | - | - | - |
| GeometryComponentType_Ring | + | - | - | + | + | + | + | + | - | - | - | + | - | - | - |
| GeometryType_CurvePolygon, GeometryComponentType_CircularArcSegment, GeometryType_CurveString, GeometryType_MultiCurvePolygon, GeometryType_MultiCurveString | + | - | - | - | + | + | + | + | - | - | - | + | - | - | - |
| GeometryType_MultiGeometry | + | - | - | + | - | - | - | + | - | - | + | + | - | - | - |

Raster

In C# you access the

`OSGeo.FDO.Connections.Capabilities.IRasterCapabilities` using the `IConnection` object (`connection.RasterCapabilities`).

In C++ you access the `FdoIRasterCapabilities` using the `IConnection` object (`connection->GetRasterCapabilities()`).

| Capability | Description |
|---------------------|---|
| SupportsRaster | Indicates whether or not a provider supports the retrieval and manipulation of image data. |
| SupportsSubsampling | Indicates whether or not a provider supports the reduction of the amount of detail and data in the image. |

| Capability | Description |
|-------------------|---|
| SupportsStitching | Indicates whether or not a provider supports the composition of multiple raster images. |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SupportsRaster, SupportsSubsampling | - | - | - | - | - | - | - | - | - | - | - | - | + | + | + |
| SupportsStitching | - | - | - | - | - | - | - | - | - | - | - | - | - | + | - |

Schema

In C# you access the `OSGeo.FDO.Connections.Capabilities.ISchemaCapabilities` using the `IConnection` object (`connection.SchemaCapabilities`).

In C++ you access the `FdoISchemaCapabilities` using the `IConnection` object (`connection->GetSchemaCapabilities()`).

| Capability | Description |
|------------------------|--|
| AutoGenerated_* | Indicates whether or not a provider supports auto generation of values for the data types represented by the asterisk: Boolean, Byte, DateTime, and Decimal. |
| ClassType_Class | Indicates whether or not a provider supports the Class type, which does not have a Geometry property. |
| ClassType_FeatureClass | Indicates whether or not a provider supports the Class type, which does have a Geometry property. |
| DataType_* | Indicates whether or not a provider supports the specific data type that replaces the *: BLOB, Boolean, Byte, CLOB, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String. |

| Capability | Description |
|--|---|
| IdentityProperty_* | Indicates whether or not a provider supports the use of the property types represented by the asterisk as identity properties: BLOB, Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String. |
| MaximumDecimalPrecision | Indicates whether or not a provider supports decimals up to the specified number of digits in length. |
| MaximumDecimalScale | Indicates whether or not a provider supports decimals with up to the specified number of digits to the right of the decimal point. |
| ReservedCharactersForName | The set of characters which cannot be used in schema element names. |
| SupportsAssociationProperties | Indicates whether or not a provider supports the association property. |
| SupportsAutoIdGeneration | Indicates whether or not a provider supports the automatic generation of id property values. |
| SupportsCompositeld | Indicates whether or not a provider supports multiple identity properties per class. |
| SupportsCompositeUniqueValueConstraints | Indicates whether or not a provider supports unique value constraints on a set of columns. |
| SupportsDataStoreScopeUniqueIdGeneration | Indicates whether or not a provider supports the automatic generation of ID values that are unique for the entire datastore, rather than just for a particular class. |
| SupportsDefaultValue | Indicates whether or not a provider supports the specification of default values for property definitions. |
| SupportsExclusiveValueRangeConstraints | Indicates whether or not a provider supports the setting of minimums that the input data must greater than and maximums that the input data must be less than. |
| SupportsInclusiveValueRangeConstraints | Indicates whether or not a provider supports the setting of minimums that the input data must greater than or |

| Capability | Description |
|--------------------------------|---|
| | equal to and maximums that the input data must be less than or equal to. |
| SupportsInheritance | Indicates whether or not a provider supports class hierarchies in the feature schema. |
| SupportsMultipleSchemas | Indicates whether or not a provider supports the definition of more than one feature schema in the data store. |
| SupportsNullValueConstraints | Indicates whether or not a provider allows property values to be null. |
| SupportsObjectProperties | Indicates whether or not a provider supports object properties. A class instance can contain an instance of another class. |
| SupportsSchemaModification | Indicates whether or not a provider supports the modification of a schema after its initial creation. |
| SupportsSchemaOverrides | Indicates whether or not a provider supports the overriding of the default rules for mapping feature schemas to provider-specific physical schemas. |
| SupportsUniqueValueConstraints | Indicates whether or not a provider supports requiring that a column contain unique values. |
| SupportsValueConstraintsList | Indicates whether or not a provider supports requiring that a column contain values that belong to a value constraints list. |

Expressible as Boolean

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ClassType_FeatureClass, DataType_String | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SqlServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DataType_Int32, DataType_DateTime, ClassType_Class | + | + | + | + | + | + | + | + | + | + | + | + | - | - | - |
| IdentityProperty_String | + | - | - | - | + | + | + | + | + | + | + | + | + | + | + |
| IdentityProperty_DateTime, DataType_Double | + | - | + | + | + | + | + | + | + | + | + | + | - | - | - |
| SupportsSchemaOverrides | - | + | + | - | + | + | + | + | + | - | + | - | + | + | + |
| DataType_Int16, DataType_Single | + | - | + | - | + | + | + | + | + | + | + | + | - | - | - |
| SupportsAutoIdGeneration, SupportsNullValueConstraints | + | + | + | - | + | + | + | + | + | + | + | - | - | - | - |
| SupportsInheritance | + | - | + | - | + | + | + | - | + | - | - | + | + | + | + |
| DataType_Decimal, DataType_Boolean | + | + | + | - | + | + | + | + | + | - | + | + | - | - | - |
| SupportsMultipleSchemas | - | - | + | - | + | + | + | + | + | + | - | + | - | + | + |
| DataType_Byte, IdentityProperty_Boolean, DataType_Int64, IdentityProperty_Byte | + | - | + | - | + | + | + | + | + | - | + | + | - | - | - |
| IdentityProperty_Decimal | + | + | - | - | + | + | + | + | + | - | + | + | - | - | - |
| IdentityProperty_Int32, IdentityProperty_Double, IdentityProperty_Single, IdentityProperty_Int16 | + | - | - | - | + | + | + | + | + | + | + | + | - | - | - |
| ReservedCharactersForName | + | + | + | - | + | + | + | - | + | + | - | - | - | - | - |
| IdentityProperty_Int64 | + | - | - | - | + | + | + | + | + | - | + | + | - | - | - |
| MaximumDecimalPrecision, MaximumDecimalScale | - | + | + | - | + | + | + | - | + | + | + | - | - | - | - |
| SupportsSchemaModification | + | + | - | - | + | + | + | + | + | - | + | - | - | - | - |
| SupportsCompositeld | + | - | + | - | + | + | + | - | + | - | + | + | - | - | - |
| AutoGenerated_Int32 | + | + | + | + | - | - | - | + | - | + | + | - | - | - | - |

1=OSGeo.SDF 2=OSGeo.SHP 3=OSGeo.ODBC 4=OSGeo.OGR 5=Autodesk.SqlServer
6=OSGeo.SQLServerSpatial 7=Autodesk.Oracle 8=OSGeo.KingOracle 9=OSGeo.MySQL
A=OSGeo.ArcSDE B=OSGeo.PostGIS C=OSGeo.WFS D=OSGeo.WMS E=Autodesk.Raster
F=OSGeo.Gdal

| Capability | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AutoGenerated_Int64, SupportsDefaultValue | - | - | + | - | + | + | + | - | + | - | + | - | - | - | - |
| SupportsCompositeUniqueValueConstraints, SupportsUnique-ValueConstraints | - | - | - | - | + | + | + | - | + | + | + | - | - | - | - |
| SupportsAssociationProperties | + | - | - | - | + | + | + | - | + | - | - | + | - | - | - |
| SupportsObjectProperties | - | - | - | - | + | + | + | - | + | - | - | + | - | - | - |
| DataType_BLOB | - | - | - | - | - | - | + | - | - | + | - | - | + | + | + |
| SupportsValueConstraintsList, SupportsExclusiveValueRange-Constraints, SupportsInclusiveValueRangeConstraints | + | - | - | - | + | + | + | - | - | - | - | - | - | - | - |
| SupportsDataStoreScopeUniqueIdGeneration | - | - | - | - | + | + | + | - | + | - | - | - | - | - | - |
| IdentityProperty_BLOB | - | - | - | - | - | - | - | - | - | + | - | - | - | - | - |
| DataType_CLOB | - | - | - | - | - | - | + | - | - | - | - | - | - | - | - |
| AutoGenerated_Int16 | - | - | + | - | - | - | - | - | - | - | - | - | - | - | - |

Not Expressible as a Boolean

| Provider | Reserved Characters For Name | Maximum Decimal Precision | Maximum Decimal Scale |
|-----------|------------------------------|---------------------------|-----------------------|
| SHP | :: | 255 | 255 |
| SDF | :: | null | null |
| SqlServer | :: | 38 | 38 |
| Oracle | :: | 38 | 38 |
| ODBC | :: | 28 | 28 |

| Provider | Reserved Characters For Name | Maximum Decimal Precision | Maximum Decimal Scale |
|------------------|------------------------------|---------------------------|-----------------------|
| MySQL | :: | 65 | 65 |
| ArcSDE | :: | 38 | 38 |
| PostGIS | null | 1000 | 1000 |
| SQLServerSpatial | :: | 38 | 38 |

Schema Management

7

This chapter describes how to create and work with schemas and explains some issues related to schema management. For example, you can use the FDO feature schema to specify how to represent geospatial features.

Schema Package

The FDO feature schema provides a logical mechanism for specifying how to represent geospatial features. FDO providers are responsible for mapping the feature schema to some underlying physical data store. The FDO feature schema is based somewhat on a subset of the OpenGIS and ISO feature models. It supports both non-spatial features and spatial features.

The Schema package contains a collection of classes that define the logical feature schema. These classes can be used to set up a feature schema and to interrogate the metadata from a provider using an object-oriented structure. The logical feature schema provides a logical view of geospatial feature data that is fully independent from the underlying storage schema. All data operations in FDO are performed against the classes and relationships defined by the logical feature schema. For example, different class types in the feature schema are used to describe different types of geospatial objects and spatial features.

Base Properties

All classes in the feature schema support the concept of base properties, which are properties that are pre-defined either by the FDO API or by a specific FDO feature provider. For example, all classes in the schema have two base properties: `ClassName` and `SchemaName`. These properties can be used to query across an inheritance hierarchy or to process the results of heterogeneous queries. FDO

feature providers can also predefine base properties. The following base properties are predefined by the FDO API:

| Property Name | Required | Description |
|----------------|----------|---|
| SchemaName | Y | Name of the schema to which objects of the class belong; read-only string. |
| ClassName | Y | Name of the class that defines the object; read-only string. |
| RevisionNumber | N | Revision number of the object; read-only 64-bit integer. NOTE Some providers may use this property to support optimistic locking. |

Cross-Schema References

Some FDO feature providers may support multiple schemas. For these providers, the feature schema supports the concept of cross-schema references for classes. This means that a class in one schema may derive from a class in another schema, relate to a class in another schema, or contain an object property definition that is based on a class in another schema.

Parenting in the Schema Classes

The feature schema object model defined in the FDO API supports full navigation through parenting. That is, once a schema element is added to an `FdoFeatureSchema` class, it can navigate the object hierarchy upward to the root `FdoFeatureSchema` and, from there, to any other element in the feature schema. This parenting support is fully defined in the `FdoSchemaElement` abstract base class.

When inserting features that have object collections, the parent object instance must be identified when inserting the child objects (for example, a parent class “Road” has an object property called “sidewalks” of type “Sidewalk”). For more information, see [Data Maintenance](#) (page 109).

Physical Mappings

Each feature provider maps the logical feature schema to an underlying physical data store. Some feature providers may provide some level of control over how the logical schema gets mapped to the underlying physical storage. For example, an RDBMS-based feature provider may allow table and column

names to be specified for classes and properties. Since this is entirely provider-dependent, the FDO API simply provides abstract classes for passing physical schema and class mappings to the provider (FdoPhysicalSchemaMapping, FdoPhysicalClassMapping, FdoPhysicalPropertyMapping, and FdoPhysicalElementMapping, respectively). The implementation of these abstract classes is up to each feature provider.

Schema Mappings

When a provider connects to a data source that it did not create and does a describe schema command, it will map FDO data types to the physical entities contained in the data source. This mapping is called the default schema mapping or alternatively, the physical to logical schema mapping.

When reverse engineering an existing schema, FDO uses the following rules to select the column(s) to use as the identity property:

- 1 Use the primary key.
- 2 Use a unique index.
- 3 Treat the data as read-only

NOTE When reverse engineering a view, FDO will apply rules 1 and 2 to the view and then, if necessary, to the base tables.

NOTE If FDO detects more than one unique index, it calculates a weight for each index and selects the lightest one. The weight of each index is the sum of the weights of its columns. Column weight depends on column type as follows. Indexes with Blobs or Clobs are not chosen. If FDO picks a multi-column index as the unique index, it creates an identity property for each column in the index.

- 1 for Boolean or Byte
 - 2 for Int16
 - 4 for Int32
 - 8 for Int64, Decimal, Double, or Float
 - 50 for DateTime
 - length for String
-

After you use FDO to create a feature schema, you can use external means to access the data store to inspect what native data types FDO uses for its logical types. This mapping is called the logical to physical schema mapping.

The following mappings have been documented in the provider appendices.

- physical to logical schema mappings for ODBC connections to a Microsoft Access database, an Excel spreadsheet, and a text file
- physical to logical and logical to physical schema mappings for Oracle, MySQL, Sql Server, and Sql Server Spatial

The mechanism for overriding the default schema mapping is described in the next topic. Sample code is located in the FDO Cookbook chapter of this document.

Schema Overrides

Using schema overrides, FDO applications can customize the mappings between Feature (logical) Schemas and the Physical Schema of the provider data store.

Schema overrides are provider-specific because different FDO providers support FDO data stores with widely different physical formats. Therefore, the types of schema mappings in these overrides also vary between providers. For example, an RDBMS-type provider might provide a mapping to index a set of columns in a class table. However, other providers would not necessarily be able to work with the concept of an index. For information about schema overrides support by a specific provider, see the appropriate appendix in this document and *The Essential FDO*.

NOTE Some providers support only default schema mappings.

Working with Schemas

There are three primary operations involved with schema management:

- Creating a schema
- Describing a schema
- Modifying a schema

Creating a Schema

The following basic steps are required to create a schema (some steps are optional; some may be done in an alternate order to achieve the same result):

- Use the `FdoFeatureSchema::Create("SchemaName", "FeatureSchema Description")` method to create a schema.
- Use the `FdoFeatureSchema::GetClasses()` method to return a class collection.
- Use the `FdoClass::Create("className", "classDescription")` or `FdoFeatureClass::Create("className", "classDescription")` method to create `FdoClass` or `FdoFeatureClass` type objects.
- Use the `FdoClassCollection::Add(class)` method to add `FdoClass` or `FdoFeatureClass` objects to the class collection.
- Use the `FdoGeometricPropertyDefinition::Create("name", "Description")` method to create `FdoGeometryProperty`.
- Use the `FdoDataPropertyDefinition::Create("name", "Description")` method to create `FdoDataProperty`.
- Use the `FdoObjectPropertyDefinition::Create("name", "Description")` method to create `FdoObjectProperty`.
- Use the `FdoClassDefinition::GetProperties()` and `Add(property)` methods to add property to class definition.
- Use the `FdoIApplySchemaCommand::SetFeatureSchema(feature schema)` method to set the schema object for the `IFdoApplySchemaCommand`.
- Use the `FdoAssociationPropertydefinition` class to represent the association between two classes. The class of the associated class must already be defined in the feature schema and cannot be abstract.
- Use the `FdoIApplySchemaCommand::Execute()` method to execute changes to the feature schema.

For an example of schema creation, see [Example: Creating a Feature Schema](#) (page 103).

Use the `FdoClassDefinition::GetIdentityProperties()` and `Add(Property Object)` methods to set the property as `FdoClass` or `FdoFeatureClass` Identifier. FDO allows multiple Identifiers for both types of classes, although Identifiers have slight differences in both cases.

FDOFeatureClass

FdoFeatureClass is a class that defines features. In the case of GIS, they would often be spatial features, having some sort of geometry associated with them. In most providers, FdoFeatureClass requires a unique identifier to distinguish the features.

However, there are identifiers only if no base class exists. If the base class has an identifier, the child class does not have one. You cannot set an identifier to the child class. Any class definition that has a base class cannot also have any identity properties because it inherits from the base class.

Therefore, you cannot send an identifier when a feature class is a child since it always inherits the identifier from the base class.

FDOClass

This class is used for non-spatial data. It can act as a stand-alone class, where it would have no association with any other class, or if the FdoClass is being used as an ObjectProperty, it can be used to define properties of some other FdoClass or FdoFeatureClass.

ObjectProperty Types

ObjectProperties have the following types:

- Value
- Collection
- OrderedCollection

The Value ObjectProperty type has a relationship of one-to-one, providing a single value for each property.

The Collection and OrderedCollection ObjectProperty types have a one-to-many relationship, where many ObjectProperties may be associated with one property. Ordered Collections can be stored in an ascending or descending order

At least one Identifier will be required if the FdoClass is to be used as a stand-alone Class.

- All Identifiers for FdoDataType_Int64 must not be Read-Only, since none of these will be an auto-generated property value.

- If creating multiple Identifiers, all Identifiers must be set to NOT NULL.

Non-Feature Class Issues

A non-feature class in FDO can be created as a stand-alone class, a contained class, or both. As a contained class, it defines a property of another class or feature class (see `FdoFeatureClass` and `FdoClassType` Global Enum). How this non-feature class is created affects the way the data is inserted, queried, and updated.

Stand-alone Class

This type of class stores non-feature data (for example, manufacturers). The `FdoClassType_Class` must be created with one or more identity properties (see `FdoObjectPropertyDefinition`), which is required in order that the class has a physical container (that is, a table in the RDBMS) associated with it. If the class is created without specifying an `IdentityProperty`, only the definition is stored in the metadata, which prevents any direct data inserts.

Contained Class

This type of class stores non-feature data that defines a property of another class or feature class (for example, Sidewalk could be a property of a Road feature class; the Sidewalk class defines the `Road.Sidewalk` property). In this case, the `FdoClassType_Class` does not need to be created with one or more identity properties, although it can be.

Class With IdentityProperty Used as ObjectProperty

This type of class reacts like a stand-alone class; however, with this type, it is possible to do direct data inserts. It can also be populated through a container class (for example, `Road.Sidewalk`) since it defines an object property (see `FdoObjectPropertyDefinition`). If this class is queried directly, only the data inserted into the class as a stand-alone is returned. The data associated with the `ObjectProperty` can only be queried through the container class (for example, `Road.Sidewalk`).

Class Without a Defined IdentityProperty Used as ObjectProperty

Because this class has no defined `IdentityProperty`, it can only be populated through the container class (for example, `Road.Sidewalk`) since it defines `ObjectProperty`. This class cannot be queried directly. The data associated with the object property can only be queried through the container class (for

example, Road.Sidewalk). As an object property, it is defined as one of the following:

- **Value type property.** Does not need any identifier since it has a one-to-one relationship with the container class.
- **Collection type property.** Requires a local identifier, which is an identifier defined when creating the ObjectProperty object.
- **Ordered Collection type property.** Requires a local identifier, which is an identifier defined when creating the ObjectProperty object.

When defining either a Collection or Ordered Collection type ObjectProperty, you must set an IdentityProperty attribute for that object property. This ObjectClass.IdentityProperty acts only as a local identifier compared to the IdentityProperty set at the class level. As a local identifier, it acts to uniquely identify each item within each collection (for example, if the local identifier for Road.Sidewalk is Side, there can be multiple sidewalks with Side="Left" but only one per Road).

Describing a Schema

Use the FdoDescribeSchema::Execute function to retrieve an FdoFeatureSchemaCollection in order to obtain any information about existing schema(s). The FdoFeatureSchemaCollection consists of all FdoFeatureSchemas in the data store and can be used to obtain information about any schema object, including FdoFeatureSchema, FdoClass, FdoFeatureClass, and their respective properties. The following functions return the main collections required to obtain information about all schema objects:

- FdoFeatureSchema::GetClasses method obtains FdoClass and FdoFeatureClasses.
- FdoClassDefinition::GetProperties method obtains a FdoPropertyDefinitionCollection.
- FdoClassDefinition::GetBaseProperties method obtains a FdoPropertyDefinitionCollection of the properties inherited from the base classes.

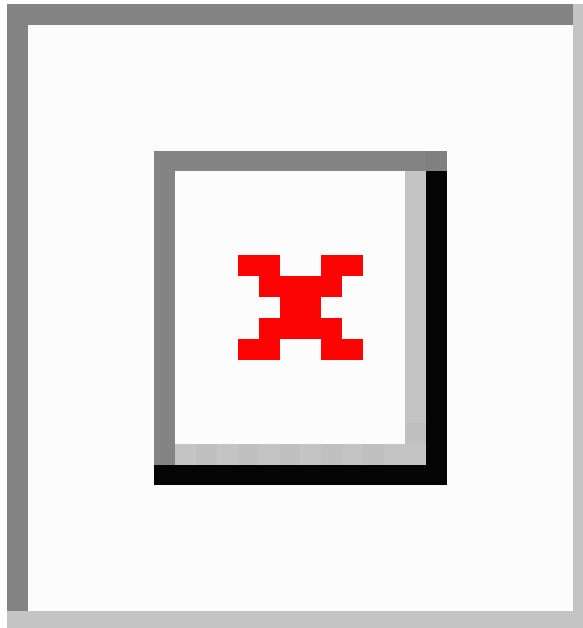
NOTE Even if your schema has no base classes (inheritance), all classes will inherit some properties from system classes.

Use these functions throughout the application to obtain any information about schema objects. For example, in order to insert data into a class, you

must use these functions to determine what data type is required. Description of the data is separate from actions.

The example in the following link is a simple function that shows how to use `FdoIDDescribeSchema` and loop through the schema and class containers to search for duplicate class names. It searches all schemas to ensure that the class name does not exist in any schema in the data store. Class names must be unique across the entire FDO database.

For a schema description example, see [Example: Describing a Schema and Writing It to an XML File](#) (page 105) NO LABEL .



FDO Schema Element Class Diagram

Modifying Models

Add schema elements to a model by inserting them into the appropriate collection.

Elements are removed from the model by using either of the following methods:

- Call the `FdoSchemaElement::Delete()` method. This flags the element for deletion when the changes are accepted (generally through `FdoIApplySchema`), but the element remains a member of all collections until that time.
- Remove the element from the appropriate collection via the `FdoSchemaCollection::Remove()` or `FdoSchemaCollection::RemoveAt()` methods. This immediately disassociates the element from the collection.

Schema Element States

All elements within the model maintain a state flag. This flag can be retrieved by calling `FdoSchemaElement::GetElementState()`, but it cannot be directly set. Instead, its state changes in reaction to the changes made to the model:

- **Unchanged.** When a schema model is retrieved via `FdoIDescribeSchema`, all elements are initially marked Unchanged.
- **Detached.** Removing an element from an owning collection sets its state to Detached.
- **Deleted.** Calling the `Delete()` method on an element sets its state to Deleted.
- **Added.** Placing an element within a collection marks the element as Added.
- **Modified.** When adding or removing a sub-element, such as a property element from a class, the class element state will be changed to Modified.

Additionally, when an element that is contained by another element is changed in any way, the containing element is also marked as Modified. So, for example, if a new value is added to the `SchemaAttributeDictionary` of the “Class3” element in our model, both the “Class3” `FdoClass` object and the `FdoFeatureSchema` object would be marked as Modified.

The state flags are maintained until the changes are accepted, that is, when `IApplySchema` is executed. At that time, all elements marked Deleted are released and all other elements are set to Unchanged.

NOTE When you remove an element from an owning collection, its state is marked as Detached. All collections currently in FDO are owning collections, except for one, the collections `FdoClassDefinition::GetIdentityProperties()`.

Rollback Mechanism

The `FdoFeatureSchema` contains a mechanism that allows you to “roll back” model changes to the last accepted state. For example, a model retrieved via `FdoIDescribeSchema` can have classes added, attributes deleted, or names and default values changed. All of these changes are thrown out and the model returned to its unmodified state by calling `FdoFeatureSchema::RejectChanges()`.

The converse of this operation is the `FdoFeatureSchema::AcceptChanges()` method, which removes all of the elements with a status of Deleted and sets the state flag of all other elements to Unchanged. Generally, this method is only invoked by FDO provider code after it has processed an `FdoIApplySchema::Execute()` command. Normal FDO clients should not call this method directly.

FDO XML Format

FDO feature schemas can be written to an XML file. The `FdoFeatureSchema` and `FdoFeatureSchemaCollection` classes support the `FdoXmlSerializable` interface. The sample code shows an `FdoFeatureSchema` object calling the `WriteXml()` method to generate an XML file containing the feature schema created by the sample code.

FDO feature schemas can also be read from an XML file. The `FdoFeatureSchemaCollection` class supports the `FdoXmlDeserializable` interface. The sample code shows an `FdoFeatureSchemaCollection` object calling the `ReadXml()` method to read a set of feature schemas into memory from an XML file. The code shows the desired schema being retrieved from the collection and applied to the data store.

The XML format used by FDO is a subset of the Geography Markup Language (GML) standardized by the Open GIS Consortium (OGC). One thing shown in the sample code is a round-trip conversion from FDO feature schema to GML schema back to FDO feature schema. To accomplish this round-trip, the `ReadXml()` method supports a superset of the GML that is written by the `WriteXml()` method.

The following table specifies the mapping of FDO feature schema elements to GML elements and attributes. This mapping is sufficient to understand the XML file generated from the schema defined by the sample code. It also provides a guide for writing a GML schema file by hand. This file can then be read in and applied to a data store. For more information, see [Example: Creating a Schema Read In from an XML File](#) (page 106).

Another form of round-trip translation would be from a GML schema produced by another vendor's tool to an FDO feature schema, and then back to a GML schema. However, the resemblance the of resulting GML schema to the original GML schema might vary from only roughly equivalent to being exactly the same.

Map FDO Element to GML Schema Fragment

| FDO Element | GML Schema Fragment |
|---|--|
| FeatureSchema | <pre> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://<customer_url>/<FeatureSchemaName>" xmlns:fdo="http://fdo.osgeo.org/isd/schema" xmlns:gml="http://www.opengis.net/gml" xmlns:<FeatureSchemaName>="http://<customer_url>/<FeatureSchemaName>" elementFormDefault="qualified" attributeFormDefault="unqualified" > { see <MetaData> } { optional xs:import element to enable schema validation <xs:import namespace="http://fdo.osgeo.org/schema" schemaLocation="<FDO SDK Install Location>/docs/XmlSchema/FdoDocument.xsd"/> } { <one xs:element and/or xs:complexType per class> } </xs:schema> </pre> |
| ClassDefinition (with identity properties) | <pre> <xs:element name="<className>" type="<className>Type" abstract="<true false>" substitutionGroup="gml:_Feature" > <xs:key name="<className>Key"> <xs:selector xpath="."//<className>"/> <xs:field xpath="<identityProperty1Name>"/> <xs:field xpath="..." /> <xs:field xpath="<identityProperty<n>Name>" /> </xs:key> </xs:element> </pre> |

| FDO Element | GML Schema Fragment |
|-------------------------------------|---|
| FeatureClass | <pre> <xs:element ...see ClassDefinition (with identity proper ties)...</xs:element> <xs:complexType name="<className>Type" abstract="<true false>"/> { see FeatureClass.GeometryProperty } > { see <MetaData> } <xs:complexContent> <xs:extension base="{baseClass} ? {baseClass.schema.name}:{baseClass.name} : 'gml:AbstractFeatureType' " > <xs:sequence> { list of properties; see DataProperty, Geometric Property } </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType> </pre> |
| FeatureClass. Geo- metryProperty | <pre> <!-- these attributes belong to the xs:complexType element --> fdo:geometryName="<geometryPropertyName>" fdo:geometricTypes="<list of FdoGeometricTypes>" fdo:geometryReadOnly="<true false>" fdo:hasMeasure="<true false>" fdo:hasElevation="<true false>" fdo:srsName="<spatialContextName>"/> </pre> |

| FDO Element | GML Schema Fragment |
|-------------------------------------|--|
| DataProperty (decimal or string) | <pre> <!-- minOccurs attribute generated only if value is 1 default attribute generated only if a default value exists fdo:readOnly attribute generated only if value is true --> <xs:element name="<propertyName>" minOccurs="{isNullable ? 0 : 1}" default="<defaultValue>" fdo:readOnly="<true false>" > { see <MetaData> } <xs:simpleType> { see DataType String or DataType Decimal } </xs:simpleType> </xs:element> </pre> |
| DataProperty (other type) | <pre> <xs:element name="<propertyName>" type="<datatype>" minOccurs="{isNullable ? 0 : 1}" default="<defaultValue>" fdo:readOnly="<true false>" > { see <MetaData> } </xs:element> </pre> |
| DataType String | <pre> <xs:restriction base="xs:string"> <xs:maxLength value="<length>" /> </xs:restriction> </pre> |
| DataType Decimal | <pre> <xs:restriction base="xs:decimal"> <xs:totalDigits value="<precision>" /> <xs:fractionDigits value="<scale>" /> </xs:restriction> </pre> |

| FDO Element | GML Schema Fragment |
|---|--|
| GeometricProperty (not a defining FeatureClass GeometryProperty) | <pre> <xs:element name="<propertyName>" type="gml:AbstractGeometryType" fdo:geometryName="<propertyName>" fdo:geometricTypes="<list of FdoGeometricTypes>" fdo:geometryReadOnly="<true false>" fdo:hasMeasure="<true false>" fdo:hasElevation="<true false>" fdo:srsName="<spatialContextName>"/> > { see <MetaData> } </xs:element> </pre> |

| FDO Element | GML Schema Fragment |
|-------------|--|
| MetaData | <pre> <!-- the pattern referenced in the xs:schema element for FeatureSchema--> <xs:annotation> <xs:documentation>{description arg to static FdoFeaturesS chema::Create()}</xs:documentation> </xs:annotation> <!-- the pattern referenced in the xs:element element for DataProperty --> <xs:annotation> <xs:documentation>{description arg to static FdoDataProp ertyDefinition::Create()}</xs:documentation> </xs:annotation> <!-- the pattern referenced in the xs:element element for a non-feature-defining GeometricProperty --> <xs:annotation> <xs:documentation>{description arg to static FdoGeomet ricPropertyDefinition::Create()}</xs:documentation> </xs:annotation> <!-- the pattern referenced in the xs:complexType element for FeatureClass --> <xs:annotation> <xs:documentation>{description arg to static FdoFeature Class::Create()}</xs:documentation> <xs:appinfo source="<uri>"/> <xs:documentation>{description arg to static FdoGeomet ricPropertyDefinition::Create()}</xs:documentation> </xs:annotation> </pre> |

| Map FDO Datatype to GML Type | |
|------------------------------|-------------|
| FDO Datatype | GML Type |
| Boolean | xs:boolean |
| Byte | fdo:Byte |
| DateTime | xs:dateTime |
| Double | xs:double |

| FDO Datatype | GML Type |
|--------------|-----------------|
| Int16 | fdo:Int16 |
| Int32 | fdo:Int32 |
| Int64 | fdo:Int64 |
| Single | xs:float |
| BLOB | xs:base64Binary |
| CLOB | xs:string |

Creating and Editing a GML Schema File

The sample in this section illustrates the creation of a GML schema file containing the definition of an FDO feature schema that contains one feature. The name of this file will have the standard XML schema extension name, *.xsd*. This means that it contains only one schema and that the root element is `xs:schema`. The `ReadXml()` method will take a filename argument whose extension is either *.xsd* or *.xml*. In the latter case, the file could contain many schema definitions. If it does, each schema is contained in an `xs:schema` element, and all `xs:schema` elements are contained in the `fdo:DataStore` element. If there is only one schema in the *.xml* file, then the `fdo:DataStore` element is not used, and the root element is `xs:schema`.

You may want to validate the schema that you create. To do so, you must include the optional `xs:import` line specified in the GML schema fragment for `FeatureSchema`.

The sample feature implements a table definition for the Buildings feature in the Open GIS Consortium document 98-046r1. This table definition is expressed in an XML format on page 14 of the document and is reproduced as follows:

```

<ogc-sfsql-table>
  <table-definition>
    <name>buildings</name>
    <column-definition>
      <name>fid</name>
      <type>INTEGER</type>
      <constraint>NOT NULL</constraint>
      <constraint>PRIMARY KEY</constraint>
    </column-definition>
    <column-definition>
      <name>address</name>
      <type>VARCHAR(64)</type>
    </column-definition>
    <column-definition>
      <name>position</name>
      <type>POINT</type>
    </column-definition>
    <column-definition>
      <name>footprint</name>
      <type>POLYGON</type>
    </column-definition>
  </table-definition>

```

Add GML for the FDO Feature Schema

Start with the skeleton GML for an FDO Feature Schema with the <MetaData> reference replaced by the valid pattern:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://<customer_url>/<FeatureSchemaName>"
  xmlns:fdo="http://fdo.osgeo.org/schema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:<FeatureSchemaName>="http://<customer_url>/<FeatureSchemaName>"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>
  <xs:annotation>
    <xs:documentation>
      {description arg to static FdoFeatureSchema::Create()}
    </xs:documentation>
  </xs:annotation>
  { <one xs:element and/or xs:complexType per class> }
</xs:schema>

```


For <customer_url> substitute “fdo_customer”. For <FeatureSchemaName> substitute “OGC980461FS”, and for {description arg ... } substitute “OGC Simple Features Specification for SQL.”

Add GML for an FDO Feature Class

Start with the GML that is already written and add the skeleton for an FDO Feature Class, which includes the skeleton for a class definition with identity properties. The <MetaData> is replaced with the valid pattern.

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://fdo_customer/OGC980461FS"
  xmlns:fdo="http://fdo.osgeo.org/schema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:OGC980461FS="http://fdo_customer/OGC980461FS"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>
  <xs:annotation>
    <xs:documentation>OGC Simple Features Specification for
      SQL</xs:documentation>
  </xs:annotation>
  <xs:element name="<className>"
    type="<className>Type"
    abstract="<true | false>"
    substitutionGroup="gml:_Feature"
  >
    <xs:key name="<className>Key">
      <xs:selector xpath="."/.<className>"/>
      <xs:field xpath="<identityPropertyName>"/>
    </xs:key>
  </xs:element>
  <xs:complexType name="<className>Type"
    abstract="<true | false>"/>
    fdo:geometryName="<geometryPropertyName>"
    fdo:geometricTypes="<list of FdoGeometricTypes>"
    fdo:geometryReadOnly="<true | false>"
    fdo:hasMeasure="<true | false>"
    fdo:hasElevation="<true | false>"
    fdo:srsName="<spatialContextName>"/>
  >
    <xs:annotation>
      <xs:documentation>{description arg to static
        FdoFeatureClass::Create()}</xs:documentation>
      <xs:appinfo source="<uri>"/>
      <xs:documentation>{description arg to static
        FdoGeometricPropertyDefinition::Create()}
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="{baseClass} ?
        {baseClass.schema.name}:{baseClass.name} :
        'gml:AbstractFeatureType' "

```

```

    >
    <xs:sequence>
      { list of properties; see DataProperty, GeometricProperty
    }
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>

```

You can make the following changes:

- For <className> substitute “buildings”.
- Set the value of the xs:element abstract attribute to false.
- For <identityPropertyName> substitute “fid”. A data property whose name is “fid” will be added.
- Set the value of the xs:complexType abstract attribute to false.
- For <geometryPropertyName> substitute “footprint”.
- For <list of FdoGeometricTypes> substitute “surface”.
- Set the values of fdo:geometryReadOnly, fdo:hasMeasure, and fdo:hasElevation to false.
- For <spatialContextName> substitute “SC_0”.
- For {description arg to FdoFeatureClass::Create()} substitute “OGC 98-046r1 buildings”.
- For <uri> substitute “http://fdo.osgeo.org/schema”.
- For {description arg to FdoGeometricPropertyDefinition::Create()} substitute “a polygon defines a building perimeter”.
- This class has no base class so set the value of the xs:extension base attribute to ‘gml:AbstractFeatureType’.

Add GML for Property Definitions

An integer data property whose name is “fid” will be added. This property is already identified as an identity property in the xs:key element. A string data property whose name is “name” and a geometry property whose name is “position” will also be added.

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://fdo_customer/OGC980461FS"
  xmlns:fdo="http://fdo.osgeo.org/schema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:OGC980461FS="http://fdo_customer/OGC980461FS"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>
  <xs:annotation>
    <xs:documentation>OGC Simple Features Specification for
      SQL</xs:documentation>
  </xs:annotation>
  <xs:element name="buildings"
    type="buildingsType"
    abstract="false"
    substitutionGroup="gml:_Feature"
  >
    <xs:key name="buildingsKey">
      <xs:selector xpath="//buildings"/>
      <xs:field xpath="fid"/>
    </xs:key>
  </xs:element>
  <xs:complexType name="buildingsType"
    abstract="false">
    footprint
    fdo:geometryName="footprint"
    fdo:geometricTypes="surface"
    fdo:geometryReadOnly="false"
    fdo:hasMeasure="false"
    fdo:hasElevation="false"
    fdo:srsName="SC_0"/>
  >
    <xs:annotation>
      <xs:documentation>OGC 98-046r1 buildings
      </xs:documentation>
      <xs:appinfo source="http://fdo.osgeo.org/schema"/>
      <xs:documentation>a polygon defines the perimeter of a
        building</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType"
      >
        <xs:sequence>
          <xs:element name="<propertyName>"

```

```

type="<datatype>"
minOccurs="{isNullable ? 0 : 1}"
default="<defaultValue>"
fdo:readOnly="<true | false>"
>
<xs:annotation>
  <xs:documentation>{description arg to static
    FdoDataPropertyDefinition::Create() }
  </xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="<propertyName>"
  minOccurs="{isNullable ? 0 : 1}"
  default="<defaultValue>"
  fdo:readOnly="<true | false>"
>
  <xs:annotation>
    <xs:documentation>{description arg to static
      FdoDataPropertyDefinition::Create() }
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="<length>" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="<propertyName>"
  ref="gml:_Geometry"
  fdo:geometryName="<propertyName>"
  fdo:geometricTypes="<list of FdoGeometricTypes>"
  fdo:geometryReadOnly="<true | false>"
  fdo:hasMeasure="<true | false>"
  fdo:hasElevation="<true | false>"
  fdo:srsName="<spatialContextName>" />
>
  <xs:annotation>
    <xs:documentation>{description arg to static
      FdoGeometricPropertyDefinition::Create() }
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>

```

```

        </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:schema>

```

You can make the following changes:

- For the first data property <propertyName> substitute “fid”.
- For the first data property <dataType> substitute “fdo:int32”.
- Do not include the minOccurs or default attributes because the value of minOccurs is 0, which is the default, and there is no <defaultValue>.
- Set the fdo:readOnly attribute for “fid” to false.
- Set the content for xs:documentation for “fid” to “feature id”.
- For the second data property <propertyName> substitute “address”.
- Do not include the minOccurs or default attributes because the value of minOccurs is 0, which is the default, and there is no <defaultValue>.
- Set the fdo:readOnly attribute for “name” to false.
- Set the content for xs:documentation for “address” to “address of the building”.
- For <length> substitute “64”.
- For the geometry property <propertyName> substitute “position”.
- For <list of FdoGeometricTypes> substitute “point”.
- Set the values of fdo:geometryReadOnly, fdo:hasMeasure, and fdo:hasElevation to false.
- For <spatialContextName> substitute “SC_0”.
- For {description arg to FdoGeometricPropertyDefinition::Create()} substitute “position of the building”.

The Final Result

After all the required substitutions, the GML for the schema containing the Buildings feature is as follows:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://fdo_customer/OGC980461FS"
  xmlns:fdo="http://fdo.osgeo.org/schema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:OGC980461FS="http://fdo_customer/OGC980461FS"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>
  <xs:annotation>
    <xs:documentation>OGC Simple Features Specification for
      SQL</xs:documentation>
  </xs:annotation>
  <xs:element name="buildings"
    type="buildingsType"
    abstract="false"
    substitutionGroup="gml:_Feature"
  >
    <xs:key name="buildingsKey">
      <xs:selector xpath="."/></buildings"/>
      <xs:field xpath="fid"/>
    </xs:key>
  </xs:element>
  <xs:complexType name="buildingsType"
    abstract="false">
    fdo:geometryName="footprint"
    fdo:geometricTypes="surface"
    fdo:geometryReadOnly="false"
    fdo:hasMeasure="false"
    fdo:hasElevation="false"
    fdo:srsName="SC_0"/>
  >
    <xs:annotation>
      <xs:documentation>OGC 98-046r1 buildings
      </xs:documentation>
      <xs:appinfo source="http://fdo.osgeo.org/schema"/>
      <xs:documentation>a polygon defines the perimeter of a
        building</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType"
      >
        <xs:sequence>
          <xs:element name="fid"

```

```

        type="fdo:int32"
        fdo:readOnly="false"
    >
        <xs:annotation>
            <xs:documentation>feature id
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="address"
        fdo:readOnly="false"
    >
        <xs:annotation>
            <xs:documentation>address of the building
            </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="64"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="position"
        ref="gml:_Geometry"
        fdo:geometryName="position"
        fdo:geometricTypes="point"
        fdo:geometryReadOnly="false"
        fdo:hasMeasure="false"
        fdo:hasElevation="false"
        fdo:srsName="SC_0"/>
    >
        <xs:annotation>
            <xs:documentation>position of the building</xs:docu
mentation>
        </xs:annotation>
    </xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>

```


Schema Management Examples

Example: Creating a Feature Schema

The following sample code creates an `FdoFeatureSchema` object called “SampleFeatureSchema.” The schema contains one class, which has three properties. The class and its properties conform to the table definition for the Lakes feature in the Open GIS Consortium document 98-046r1. This table definition is expressed in an XML format on page 10 of the document and is reproduced as follows:

```
<ogc-sfsql-table>
  <table-definition>
    <name>lakes</name>
    <column-definition>
      <name>fid</name>
      <type>INTEGER</type>
      <constraint>NOT NULL</constraint>
      <constraint>PRIMARY KEY</constraint>
    </column-definition>
    <column-definition>
      <name>name</name>
      <type>VARCHAR(64)</type>
    </column-definition>
    <column-definition>
      <name>shore</name>
      <type>POLYGON</type>
    </column-definition>
  </table-definition>
```

The table definition whose name is “lakes” is mapped to an `FdoFeatureClass` object called “SampleFeatureClass.” The column definition whose name is “fid” is mapped to an `FdoDataPropertyDefinition` object called “SampleIdentityDataProperty.” The column definition whose name is “name” is mapped to an `FdoDataPropertyDefinition` object called “SampleNameDataProperty.” The column definition whose name is “shore” is mapped to an `FdoGeometricPropertyDefinition` object called “SampleGeometricProperty.”

```

// Create the ApplySchema command
FdoPtr<FdoIApplySchema> sampleApplySchema;
sampleApplySchema = (FdoIApplySchema *)
    connection->CreateCommand(FdoCommandType_ApplySchema);
// Create the feature schema
FdoPtr<FdoFeatureSchema> sampleFeatureSchema;
sampleFeatureSchema = FdoFeatureSchema::Create(L"SampleFeaturesS
chema", L"Sample Feature Schema Description");
// get a pointer to the feature schema's class collection
// this object is used to add classes to the schema
FdoPtr<FdoClassCollection> sampleClassCollection;
sampleClassCollection = sampleFeatureSchema->GetClasses();
// create a feature class, i.e., a class containing a geometric
// property set some class level properties
FdoPtr<FdoFeatureClass> sampleFeatureClass;
sampleFeatureClass = FdoFeatureClass::Create(L"SampleFeatureClass",
    L"Sample Feature Class Description");
sampleFeatureClass->SetIsAbstract(false);
// get a pointer to the feature class's property collection
// this pointer is used to add data and other properties to the
class
FdoPtr<FdoPropertyDefinitionCollection> sampleFeatureClassProper
ties;
sampleFeatureClassProperties = sampleFeatureClass->GetProperties();
// get a pointer to the feature schema's class collection
// this object is used to add classes to the schema
FdoPtr<FdoClassCollection> sampleClassCollection;
sampleClassCollection = sampleFeatureSchema->GetClasses();
// get a pointer to the feature class's identity property collec
tion
// this property is used to add identity properties to the feature
// class
FdoPtr<FdoDataPropertyDefinitionCollection> sampleFeatureClassId
entityProperties;
sampleFeatureClassIdentityProperties = sampleFeatureClass->
GetIdentityProperties();
// create a data property that is of type Int32 and identifies
// the feature uniquely
FdoPtr<FdoDataPropertyDefinition> sampleIdentityDataProperty;
sampleIdentityDataProperty = FdoDataPropertyDefinition::Cre
ate(L"SampleIdentityDataProperty", L"Sample Identity Data Property
Description");
sampleIdentityDataProperty->SetDataType(FdoDataType_Int32);

```

```

sampleIdentityDataProperty->SetReadOnly(false);
sampleIdentityDataProperty->SetNullable(false);
sampleIdentityDataProperty->SetIsAutoGenerated(false);
// add the identity property to the sampleFeatureClass
sampleFeatureClassProperties->Add(sampleIdentityDataProperty);
sampleFeatureClassIdentityProperties->Add(sampleIdentityDataProperty);
// create a data property that is of type String and names the
// feature
FdoPtr<FdoDataPropertyDefinition> sampleNameDataProperty;
sampleNameDataProperty = FdoDataPropertyDefinition::Create(L"SampleNameDataProperty", L"Sample Name Data Property Description");
sampleNameDataProperty->SetDataType(FdoDataType_String);
sampleNameDataProperty->SetLength(64);
sampleNameDataProperty->SetReadOnly(false);
sampleNameDataProperty->SetNullable(false);
sampleNameDataProperty->SetIsAutoGenerated(false);
// add the name property to the sampleFeatureClass
sampleFeatureClassProperties->Add(sampleNameDataProperty);
// create a geometric property
FdoPtr<FdoGeometricPropertyDefinition> sampleGeometricProperty;
sampleGeometricProperty = FdoGeometricPropertyDefinition::Create(L"SampleGeometricProperty", L"Sample Geometric Property Description");
sampleGeometricProperty->SetGeometryTypes(FdoGeometricType_Surface);
sampleGeometricProperty->SetReadOnly(false);
sampleGeometricProperty->SetHasMeasure(false);
sampleGeometricProperty->SetHasElevation(false);
// add the geometric property to the sampleFeatureClass
sampleFeatureClassProperties->Add(sampleGeometricProperty);
// identify it as a geometry property
sampleFeatureClass->SetGeometryProperty(sampleGeometricProperty);
// add the feature class to the schema
sampleClassCollection->Add(sampleFeatureClass);
// point the ApplySchema command at the newly created feature
// schema and execute
sampleApplySchema->SetFeatureSchema(sampleFeatureSchema);
sampleApplySchema->Execute();

```

Example: Describing a Schema and Writing It to an XML File

The following sample code demonstrates describing a schema and writing it to an XML file:

```

// create the DescribeSchema command
FdoPtr<FdoIDescribeSchema> sampleDescribeSchema;
sampleDescribeSchema = (FdoIDescribeSchema *)
    connection->CreateCommand(FdoCommandType_DescribeSchema);
// executing the DescribeSchema command returns a feature
// schema collection that is, the set of feature schema which
// reside in the DataStore
FdoPtr<FdoFeatureSchemaCollection> sampleFeatureSchemaCollection;
sampleFeatureSchemaCollection = sampleDescribeSchema->Execute();
// find the target feature schema in the collection, write it
// to an xml file, and clear the collection
sampleFeatureSchema = sampleFeatureSchemaCollection->Find
Item(L"SampleFeatureSchema");
sampleFeatureSchema->WriteXml(L"SampleFeatureSchema.xml");
sampleFeatureSchemaCollection->Clear();

```

Example: Destroying a Schema

The following sample code demonstrates destroying a schema:

```

// create the DestroySchema command
FdoPtr<FdoIDestroySchema> sampleDestroySchema;
sampleDestroySchema = (FdoIDestroySchema *)
    connection->CreateCommand(FdoCommandType_DestroySchema);
// destroy the schema
sampleDestroySchema->SetSchemaName(L"SampleFeatureSchema");
sampleDestroySchema->Execute();

```

Example: Creating a Schema Read In from an XML File

The following sample code demonstrates creating a schema read in from an XML file:

```

sampleFeatureSchemaCollection->ReadXml(L"SampleFeatureSchema.xml");
sampleFeatureSchema = sampleFeatureSchemaCollection->Find
Item(L"SampleFeatureSchema");
sampleApplySchema->SetFeatureSchema(sampleFeatureSchema);
sampleApplySchema->Execute();
sampleFeatureSchemaCollection->Clear();

```

SampleFeatureSchema.xml

The following sample XML schema is the contents of the file written out by the WriteXml method belonging to the FdoFeatureSchema class object that was created in the preceding sample code:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://fdo_customer/SampleFeatureSchema"
  xmlns:fdo="http://fdo.osgeo.org/schema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:SampleFeatureSchema="http://fdo_customer/
    SampleFeatureSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
<xs:annotation>
  <xs:documentation>Sample Feature Schema Description
  </xs:documentation>
  <xs:appinfo source="http://fdo.osgeo.org/schema" />
</xs:annotation>
<xs:element name="SampleFeatureClass"
  type="SampleFeatureSchema:SampleFeatureClassType"
  abstract="false" substitutionGroup="gml:_Feature">
  <xs:key name="SampleFeatureClassKey">
    <xs:selector xpath="."/SampleFeatureClass" />
    <xs:field xpath="SampleIdentityDataProperty" />
  </xs:key>
</xs:element>
<xs:complexType name="SampleFeatureClassType"
  abstract="false"
  fdo:geometryName="SampleGeometricProperty"
  fdo:hasMeasure="false"
  fdo:hasElevation="false"
  fdo:srsName="SC_0"
  fdo:geometricTypes="surface">
  <xs:annotation>
    <xs:documentation>Sample Feature Class Description
    </xs:documentation>
    <xs:appinfo source="http://fdo.osgeo.org/schema" />
    <xs:documentation>Sample Geometric Property Descrip
tion</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element name="SampleIdentityDataProperty"
          default=""
          type="fdo:int32">
          <xs:annotation>

```

```

        <xs:documentation>
            Sample Identity Data Property Description
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="SampleNameDataProperty"
    default="">
    <xs:annotation>
        <xs:documentation>
            Sample Name Data Property Description
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:maxLength value="64" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>

```

Data Maintenance

8

This chapter provides information about using the FDO API to maintain data.

Data Maintenance Operations

The primary operations associated with data maintenance are:

- Inserting
- Updating
- Deleting
- Transactions
- Locking

NOTE Discussion of Transactions and Locking is deferred to a future release of this document.

Inserting Values

Preconditions

In a previous chapter, we created a feature schema and added a feature class to it. The feature class had three properties: an integer data property, a string data property, and a geometric property. We applied this feature schema to the data store. We are now ready to create feature data objects, which are instances of the feature class, and insert them into the data store.

Property Values in General

We can now create feature data objects, which are instances of the feature class, by defining a set of property values corresponding to the properties defined for the class and then inserting them into the data store.

An FDO class corresponds roughly to a table definition in a relational database and a property of a class corresponds roughly to a column definition in a table. Adding the property values corresponds roughly to adding a row in the table.

The main distinction between a data value or geometry value and a property value is the order in which they are created. A data value or geometry value object is created first and is then used to create a property value object. The property value object is then added to the value collection object belonging to the Insert command object. Then, the command is executed.

An insert operation consists of the following steps:

- 1 Create the insert command object (type `FdoInsert`); this object can be reused for multiple insert operations.
- 2 Point the insert command object at the feature class to which you are adding values (call the `SetFeatureClassName(<className>)` method).
- 3 From the insert command object, obtain a pointer using the `GetPropertyValues()` method to a value collection object (type `FdoPropertyValueCollection`). You will add property values to the insert command object by adding values to the collection object.
- 4 Create a data value (type `FdoDataValue`) or geometry value (type `FdoGeometryValue`) object. Creating the data value is straightforward; you pass the string or integer value to a static `Create()` method. Creating the geometry value is described in [Geometry Property Values](#) (page 111).
- 5 Create a property value (type `FdoPropertyValue`) object, which involves passing the data value or geometry value object as an argument to a static `Create()` method.
- 6 Add the property value object to the value collection object.
- 7 Execute the Insert command.

Data Property Values

A data value object contains data whose type is one of the following:

- Boolean

- Byte
- DateTime
- Decimal
- Double
- Int16
- Int32
- Int64
- Single (another floating point type)
- String
- Binary large object (BLOB)
- Character large object (CLOB)

The data value object is added to the data property value object. The data property value object is added to the property value collection belonging to the Insert command.

Geometry Property Values

A geometry property value object contains a geometry in the form of a byte array. A geometry can be relatively simple, for example, a point (a single pair of ordinates), or quite complex, for example, a polygon (one or more arrays of ordinates). In the latter case, a number of geometry objects are created and then combined together to form the target geometry. Finally, the target geometry is converted to a byte array and incorporated into the geometry property value object.

Creating a geometry value object consists of the following steps:

- 1 Create a geometry value object (type `FdoGeometryValue`) using a static `Create()` method.
- 2 Create a geometry factory object (type `FdoAgfGeometryFactory`) using a static `GetInstance()` method. This object is used to create the geometry object or objects which comprise the target geometry.
- 3 Create the required geometry object or objects using the appropriate `Create<geometry>` method() belonging to the geometry factory object.

- 4 Use the geometry factory object to convert the target geometry object to a byte array.
- 5 Incorporate the byte array into the geometry property value object.

Example: Inserting an Integer, a String, and a Geometry Value

The following sample code shows how to insert an integer, a string, and a geometry value:

```

// create the insert command
FdoPtr<FdoIInsert> sampleInsert;
sampleInsert = (FdoIInsert *)
    connection->CreateCommand(FdoCommandType_Insert);
// index returned by the operation which adds a value to the value
// collection
FdoInt32 valueCollectionIndex = 0;
// point the Insert command to the target class
// use a fully qualified class name
// whose format is <schemaName>:<className>
sampleInsert-> SetFeatureClassName(L"SampleFeatureSchema:SampleFea
tureClass");
// get the pointer to the value collection used to add properties
// to the Insert command
FdoPtr<FdoPropertyValueCollection> samplePropertyValues;
samplePropertyValues = sampleInsert->GetPropertyValues();
// create an FdoDataValue for the identity property value
FdoPtr<FdoDataValue> sampleIdentityDataValue;
sampleIdentityDataValue = FdoDataValue::Create(101);
// add the FdoDataValue to the identity property value
FdoPtr<FdoPropertyValue> sampleIdentityPropertyValue;
sampleIdentityPropertyValue =
    FdoPropertyValue::Create(L"SampleIdentityDataProperty",
        sampleIdentityDataValue);
// add the identity property value to the value collection
valueCollectionIndex =
    samplePropertyValues->Add(sampleIdentityPropertyValue);
// create an FdoDataValue for the name property value
FdoPtr<FdoDataValue> sampleNameDataValue;
sampleNameDataValue = FdoDataValue::Create(L"Blue Lake");
// add the FdoDataValue to the name property value
FdoPtr<FdoPropertyValue> sampleNamePropertyValue;
sampleNamePropertyValue =
    FdoPropertyValue::Create(L"SampleNameDataProperty",
        sampleNameDataValue);
// add the name property value to the value collection
valueCollectionIndex =
    samplePropertyValues->Add(sampleNamePropertyValue);
// create an FdoGeometryValue for the geometry property value
// this polygon represents a lake which has an island
// the outer shoreline of the lake is defined as a linear ring
// the shoreline of the island is defined as a linear ring
// the outer shoreline is the external boundary of the polygon

```

```

// the island shoreline is an internal linear ring
// a polygon geometry can have zero or more internal rings
FdoPtr<FdoGeometryValue> sampleGeometryValue;
sampleGeometryValue = FdoGeometryValue::Create();
// create an instance of a geometry factory used to create the
// geometry objects
FdoPtr<FdoFgfGeometryFactory> sampleGeometryFactory;
sampleGeometryFactory = FdoFgfGeometryFactory::GetInstance();
// define the external boundary of the polygon, the shoreline of
// Blue Lake
FdoPtr<FdoILinearRing> exteriorRingBlueLake;
FdoInt32 numBlueLakeShorelineOrdinates = 10;
double blueLakeExteriorRingOrdinates[] = {52.0, 18.0, 66.0, 23.0,
    73.0, 9.0, 48.0, 6.0, 52.0, 18.0};
exteriorRingBlueLake = sampleGeometryFactory->CreateLinearRing(
    FdoDimensionality_XY, numBlueLakeShorelineOrdinates,
    blueLakeExteriorRingOrdinates);
// define the shoreline of Goose Island which is on Blue Lake
// this is the sole member of the list of interior rings
FdoPtr<FdoILinearRing> linearRingGooseIsland;
FdoInt32 numGooseIslandShorelineOrdinates = 10;
double gooseIslandLinearRingOrdinates[] = {59.0, 18.0, 67.0, 18.0,
    67.0, 13.0, 59.0, 13.0, 59.0, 18.0};
linearRingGooseIsland = sampleGeometryFactory->CreateLinearRing(
    FdoDimensionality_XY, numGooseIslandShorelineOrdinates,
    gooseIslandLinearRingOrdinates);
// add the Goose Island linear ring to the list of interior rings
FdoPtr<FdoLinearRingCollection> interiorRingsBlueLake;
interiorRingsBlueLake = FdoLinearRingCollection::Create();
interiorRingsBlueLake->Add(linearRingGooseIsland);
// create the Blue Lake polygon
FdoPtr<FdoIPolygon> blueLake;
blueLake =
    sampleGeometryFactory->CreatePolygon(exteriorRingBlueLake,
    interiorRingsBlueLake);
// convert the Blue Lake polygon into a byte array
// and set the geometry value to this byte array
FdoByteArray * geometryByteArray =
    sampleGeometryFactory->GetAgf(blueLake);
sampleGeometryValue->SetGeometry(geometryByteArray);
// add the Blue Lake FdoGeometryValue to the geometry property
value
FdoPtr<FdoPropertyValue> sampleGeometryPropertyValue;

```

```

sampleGeometryPropertyValue =
    FdoPropertyValue::Create (L"SampleGeometryProperty",
        sampleGeometryValue);
// add the geometry property value to the value collection
valueCollectionIndex =
    samplePropertyValues->Add(sampleGeometryPropertyValue);
// do the insertion
// the command returns an FdoIFeatureReader
FdoPtr<FdoIFeatureReader> sampleFeatureReader;
sampleFeatureReader = sampleInsert->Execute();

```

Updating Values

After inserting (see [Inserting Values](#) (page 109)), you can update the values. The update operation involves identifying a feature class (“table”), a feature class object (“row”), and an object property (“column in a row”) to be changed, and supplying a new value for the object property to replace the old.

First, create an `FdoUpdate` command object and use the command object’s `SetFeatureClassName()` method to identify the feature class. Then, create a filter to identify the feature class object whose properties we want to update, and use the command object’s `SetFilter()` method to attach the command to it. Filters are discussed in [Filter and Expression Languages](#) (page 129).

One of the data properties in the example `SampleFeatureClass` class definition is an identity property, whose name is “`SampleIdentityDataProperty`” and whose type is `fdo:Int32`. This means that its value uniquely identifies the feature class object, that is, the “row”. Use the name of the identity property in the filter. In the Insert operation, the value of the identity property was set to be ‘101’. The value of the filter that is needed is “(`SampleIdentityDataProperty` = 101)”.

Finally, create a property value, which contains the new value, attach it to the command object, and then execute the command.

Example: Updating Property Values

The following is an example of updating property values:

```

FdoPtr<FdoIUpdate> sampleUpdate;
sampleUpdate =
    (FdoIUpdate *)connection->CreateCommand(FdoCommandType_Update);
FdoInt32 numUpdated = 0;
// point the Update command at the target feature class
// use a fully qualified class name
// whose format is <schemaName>:<className>
sampleUpdate->SetFeatureClassName(L"SampleFeatureSchema:SampleFeatureClass");
// set the filter to identify which set of properties to update
sampleUpdate->SetFilter(L"( SampleIdentityDataProperty = 101 )");
// get the pointer to the value collection used to add properties
// to the Update command
// we are reusing the samplePropertyValues object that we used
// for the insert operation
samplePropertyValues = sampleUpdate->GetPropertyValues();
// create an FdoDataValue for the name property value
FdoPtr<FdoDataValue> sampleNameDataValue;
sampleNameDataValue = FdoDataValue::Create(L"Green Lake");
// set the name and value of the property value
sampleNamePropertyValue->SetName(L"SampleNameDataProperty");
sampleNamePropertyValue->SetValue(sampleNameDataValue);
// add the name property value to the property value collection
// owned by the Update command
samplePropertyValues->Add(sampleNamePropertyValue);
// execute the command
numUpdated = sampleUpdate->Execute();

```

Deleting Values

In addition to inserting (see [Inserting Values](#) (page 109)) and updating (see [Updating Values](#) (page 115)) values, you can delete the values. The deletion operation involves identifying a feature class (“table”) whose feature class objects (“rows”) are to be deleted.

First, create an `FdoIDelete` command object and use the command object’s `SetFeatureClassName()` method to identify the feature class. Then, create a filter to identify the feature class objects that you want to delete, and use the command object’s `SetFilter()` method to attach the filter to it. You can use the same filter that was specified in the preceding section, [Updating Values](#) (page 115). Finally, execute the command.

Example: Deleting Property Values

```
FdoPtr<FdoIDelete> sampleDelete;  
sampleDelete =  
    (FdoIDelete *)connection->CreateCommand(FdoCommandType_Delete);  
FdoInt32 numDeleted = 0;  
sampleDelete->  
    SetFeatureClassName(L"SampleFeatureSchema:SampleFeatureClass");  
sampleDelete->SetFilter(L"( SampleIdentityDataProperty = 101 )");  
numDeleted = sampleDelete->Execute();
```

Related Class Topics

The following classes are used in the preceding Data Maintenance examples:

- FdoInsert
- FdoPropertyValueCollection
- FdoDataValue
- FdoPropertyValue
- FdoGeometryValue
- FdoFgfGeometryFactory
- FdoILinearRing
- FdoLinearRingCollection
- FdoIPolygon
- FdoByteArray
- FdoIDelete
- FdoIUpdate

For more information, see *FDO API Reference Help*.

Performing Queries

9

This chapter describes how to create and perform queries. In the FDO API, you can use queries to retrieve specific features from a data store.

Creating a Query

You create and perform queries using the `FdoISelect` class, which is a member of the Feature sub-package of the Commands package. Queries are used to retrieve features from the data store, and are executed against one class at a time. The class is specified using the `SetFeatureClassName()` method in `FdoIFeatureCommand`. The `SetFeatureClassName` can be used with feature and non-feature classes.

`FdoISelect` supports the use of filters to limit the scope of features returned by the command. This is done through one of the `SetFilter` methods available in the `FdoIFeatureCommand` class. The filter is similar to the SQL WHERE clause, which specifies the search conditions that are applied to one or more class properties.

Search conditions include spatial and non-spatial conditions. Non-spatial queries create a condition against a data property, such as an integer or string. Basic comparisons (`=`, `<`, `>`, `>=`, `<=`, `!=`), pattern matching (like), and 'In' comparisons can be specified. Spatial queries create a spatial condition against a geometry property. Spatial conditions are enumerated in `FdoSpatialCondition` and `FdoDistanceCondition`.

The feature reader (`FdoIFeatureReader`) is used to retrieve the results of a query for feature and non-feature classes. To retrieve the features from the reader, iterate through the reader using the `FdoIFeatureReader.ReadNext()` method().

Query Example

In the Data Maintenance chapter, we created an instance of the `FdoFeatureClass SampleFeatureClass` and assigned values to its integer, string, and geometry properties (see [Example: Inserting an Integer, a String, and a Geometry Value](#) (page 112)). The sample code in the following query example selects this instance and retrieves the values of its properties. Specifically, the sample code does the following:

- 1 Creates the select command, and
- 2 Points the select command at the target `FdoFeatureClass SampleFeatureClass`, and
- 3 Creates a filter to identify which instance of `SampleFeatureClass` to select, and
- 4 Points the select command at the filter, and
- 5 Executes the command, which returns an `FdoIFeatureReader` object, and
- 6 Loops through the feature reader object, which contains one or more query results depending on the filter arguments. In the sample code provided, there is only one result.
- 7 Finally, the code extracts the property values from each query result.

```

// we have one FdoFeatureClass object in the DataStore
// create a query that returns this object
// create the select command
FdoPtr<FdoISelect> sampleSelect;
sampleSelect = (FdoISelect *)
    connection->CreateCommand(FdoCommandType_Select);
// point the select command at the target FdoFeatureClass
// SampleFeatureClass
sampleSelect->SetFeatureClassName(L"SampleFeatureClass");
// create the filter by
// 1. creating an FdoIdentifier object containing the name of
//    the identity property
FdoPtr<FdoIdentifier> queryPropertyName;
queryPropertyName =
    FdoIdentifier::Create(L"SampleIdentityDataProperty");
// 2. creating an FdoDataValue object containing the value of the
//    identity property
FdoPtr<FdoDataValue> queryPropertyValue;
queryPropertyValue = FdoDataValue::Create(101);
// 3. calling FdoComparisonCondition::Create() passing in the
//    the queryPropertyName, an enumeration constant signifying an
//    equals comparison operation, and the queryPropertyValue
FdoPtr<FdoFilter> filter;
filter = FdoComparisonCondition::Create(queryPropertyName,
    FdoComparisonOperations_EqualTo, queryPropertyValue);
// point the select command at the filter
sampleSelect->SetFilter(filter);
// execute the select command
FdoPtr<FdoIFeatureReader> queryResults;
queryResults = sampleSelect->Execute();
// declare variables needed to capture query results
FdoPtr<FdoClassDefinition> classDef;
FdoPtr<FdoPropertyDefinitionCollection> properties;
FdoInt32 numProperties = 0;
FdoPropertyDefinition * propertyDef;
FdoPropertyType propertyType;
FdoDataType dataType;
FdoDataPropertyDefinition * dataPropertyDef;
FdoString * propertyName = NULL;
FdoPtr<FdoByteArray> byteArray;
FdoIGeometry * geometry = NULL;
FdoGeometryType geometryType = FdoGeometryType_None;
FdoIPolygon * polygon = NULL;

```

```

FdoILinearRing * exteriorRing = NULL;
FdoILinearRing * interiorRing = NULL;
FdoIDirectPosition * position = NULL;
FdoInt32 dimensionality = FdoDimensionality_XY;
FdoInt32 numPositions = 0;
FdoInt32 numInteriorRings = 0;
// loop through the query results
while (queryResults->ReadNext()) {
    // get the feature class object and its properties
    classDef = queryResults->GetClassDefinition();
    properties = classDef->GetProperties();
    // loop through the properties
    numProperties = properties->GetCount();
    for(int i = 0; i < numProperties; i++) {
        propertyDef = properties->GetItem(i);
        // get the property name and property type
        propertyName = propertyDef->GetName();
        propertyType = propertyDef->GetPropertyType();
        switch (propertyType) {
            // it's a data property
            case FdoPropertyType_DataProperty:
                dataPropertyDef =
                    dynamic_cast<FdoDataPropertyDefinition *>
                    (propertyDef);
                dataType = dataPropertyDef->GetDataType();
                switch (dataType) {
                    case FdoDataType_Boolean:
                        break;
                    case FdoDataType_Int32:
                        break;
                    case FdoDataType_String:
                        break;
                    default:
                }
                break;
            // it's a geometric property
            // convert the byte array to a geometry
            // and determine the derived type of the geometry
            case FdoPropertyType_GeometricProperty:
                byteArray = queryResults->GetGeometry(propertyName);
                geometry =
                    sampleGeometryFactory->CreateGeometryFromAgf
                    (byteArray);

```

```

geometryType = geometry->GetDerivedType();
// resolve the derived type into a list of ordinates
switch (geometryType) {
    case FdoGeometryType_None:
        break;
    case FdoGeometryType_Point:
        break;
    case FdoGeometryType_LineString:
        break;
    case FdoGeometryType_Polygon:
        polygon = dynamic_cast<FdoIPolygon *>(geometry);
        exteriorRing = polygon->GetExteriorRing();
        dimensionality = exteriorRing-
            >GetDimensionality();
        numPositions = exteriorRing->GetCount();
        double X, Y, Z, M;
        for(int i=0; i<numPositions; i++) {
            position = exteriorRing->GetItem(i);
            if (dimensionality & FdoDimensionality_Z &&
                dimensionality & FdoDimensionality_M) {
                X = position->GetX();
                Y = position->GetY();
                Z = position->GetZ();
                M = position->GetM();
            }
            else if (dimensionality & FdoDimensionality_Z
                && !(dimensionality & FdoDimensionality_M)) {
                X = position->GetX();
                Y = position->GetY();
                Z = position->GetZ();
            }
            else {
                X = position->GetX();
                Y = position->GetY();
            }
        }
        numInteriorRings = polygon-
            >GetInteriorRingCount();
        for(int i=0; i<numInteriorRings; i++) {
            interiorRing = polygon->GetInteriorRing(i);
            // do same for interior ring as exterior ring
        }
        break;
    case FdoGeometryType_MultiPoint:
        break;
}

```

```

        case FdoGeometryType_MultiLineString:
            break;
        case FdoGeometryType_MultiPolygon:
            break;
        case FdoGeometryType_MultiGeometry:
            break;
        case FdoGeometryType_CurveString:
            break;
        case FdoGeometryType_CurvePolygon:
            break;
        case FdoGeometryType_MultiCurveString:
            break;
        case FdoGeometryType_MultiCurvePolygon:
            break;
        default:
    }
    break;
default:
}
}
}

```

Long Transaction Processing

10

This chapter defines long transactions (LT) and long transaction interfaces, and explains how to implement LT processing in your application.

NOTE For this release, the providers that support long transaction processing are Autodesk FDO Provider for Oracle and OSGeo FDO Provider for ArcSDE.

What Is Long Transaction Processing?

A long transaction (LT) is an administration unit that is used to group conditional changes to objects. Depending on the situation, such a unit can contain conditional changes to one or to many objects. Long transactions are used to modify as-built data in the database without permanently changing the as-built data. Long transactions can be used to apply revisions or alternates to an object.

A root long transaction is a long transaction that represents permanent data and that has descendents. Any long transaction has a root long transaction as an ancestor in its long transaction dependency graph. A leaf long transaction does not have descendents.

For more information about Oracle-specific long transaction versions and locking, see [Locking and Long Transactions](#) (page 216).

Supported Interfaces

In the current release of FDO, the following long transaction interfaces are supported:

- `FDOIActivateLongTransaction`
- `FDOIDeactivateLongTransaction`
- `FDOIRollbackLongTransaction`
- `FDOICommitLongTransaction`
- `FDOICreateLongTransaction`
- `FDOIGetLongTransaction`

These interfaces are summarized below. For more information about their usage, supported methods, associated enumerations and readers, see the *FDO API Reference Help*.

FDOIActivateLongTransaction

The `FdoIActivateLongTransaction` interface defines the `ActivateLongTransaction` command, which activates a long transaction where feature manipulation and locking commands operate on it. Input to the activate long transaction command is the long transaction name. The `Execute` operation activates the identified long transaction.

FDOIDeactivateLongTransaction

The `FdoIDeactivateLongTransaction` interface defines the `DeactivateLongTransaction` command, which deactivates the active long transaction where feature manipulation and locking commands operate on it. If the active long transaction is the root long transaction, then no long transaction will be deactivated.

FDOIRollbackLongTransaction

The `FdoIRollbackLongTransaction` interface defines the `RollbackLongTransaction` command, which allows a user to execute rollback operations on a long transaction. Two different rollback operations are available: Full and Partial.

The operation is executed on all data within a long transaction and on all its descendents. The data is removed from the database and all versions involved in the process deleted.

NOTE If the currently active long transaction is the same as the one being committed or rolled back, then, if the commit or rollback succeeds, the provider resets the current active long transaction to be the root long transaction. If it does not succeed, the active long transaction is left alone and current. If the currently active long transaction is not the same as the one being committed or rolled back, then it is not affected.

FDOICCommitLongTransaction

The `FdoICommitLongTransaction` interface defines the `CommitLongTransaction` command, which allows a user to execute commit operations on a long transaction. Two different commit operations are available: Full and Partial.

The commit operation can be performed on a leaf long transaction only. A long transaction is a leaf long transaction if it does not have descendents.

FDOICreateLongTransaction

The `FdoICreateLongTransaction` interface defines the `CreateLongTransaction` command, which creates a long transaction that is based on the currently active long transaction. There is always an active long transaction. If the user has not activated a user-defined long transaction, then the root long transaction is active.

Input to the `CreateLongTransaction` command includes a name and description for the new long transaction. The long transaction name submitted to the command has to be unique. If it is not unique, an exception is thrown.

FDOIGetLongTransactions

The `FdoIGetLongTransactions` interface defines the `GetLongTransactions` command, which allows the user to retrieve long transaction information. If a long transaction name is submitted, the command returns the information for the named long transaction only. If no long transaction name is given, the command retrieves the names of all available long transactions.

For each returned long transaction, the user has the option to retrieve a list of descendents and/or ancestors.

Filter and Expression Languages

11

This chapter discusses the use of filters and filter expressions. You can use filters and expressions to specify to an FDO provider how to identify a subset of the objects in a data store.

For more information and implementation details about the expression functions signatures, the RDBMS-specific built-in support for some of the functions, and the provider-specific support, see the appendix [Expression Functions](#) (page 323).

Filters

FDO uses filters through its commands (including provider-specific commands) to select certain features and exclude others.

A filter is a construct that an application specifies to an FDO provider to identify a subset of objects of an FDO data store. For example, a filter may be used to identify all Road type features that have 2 lanes and that are within 200 metres of a particular location. Many FDO commands use filter parameters to specify the objects to which the command applies. For example, a select command takes a filter to identify the objects that the application wants to retrieve or a delete command takes a filter to identify the objects that the application wants to delete from the data store.

When a command executes, the filter is evaluated for each feature instance and that instance is included in the scope of the command only if the filter evaluates to True. Filters may be specified either as text or as an expression tree. Feature providers declare their level of support for filters through the filter capabilities metadata. Query builders should configure themselves based on the filter capabilities metadata in order to provide users with a robust user interface. For more information, see [What Is an Expression?](#) (page 19).

Expressions

FDO uses expressions through its commands (including provider-specific commands) to specify input values in order to filter features. In general, commands in FDO do not support the SQL command language (the one exception is the optional `SQLCommand`). However, to facilitate ease of use for application developers, expressions in FDO can be specified using a textual notation that is based syntactically on expressions and SQL `WHERE` clauses. In FDO, expressions are not intended to work against tables and columns, but against feature classes, properties, and relationships. For example, an expression to select roads with four or more lanes might look like this:

```
Lanes >= 4
```

An expression is a construct that an application can use to build up a filter. In other words, an expression is a clause of a filter or larger expression. For example, “`Lanes >=4 and PavementType = 'Asphalt'`” takes two expressions and combines them to create a filter.

Filter and Expression Text

In general, commands in FDO do not support the SQL command language (the one exception is the optional `SQLCommand`). However, to facilitate ease of use for application developers, expressions and filters in FDO can be specified using a textual notation that is based syntactically on expressions and SQL `WHERE` clauses. The biggest difference between this approach and SQL is that these clauses are not intended to work against tables and columns, but against feature classes, properties, and relationships. For example, a filter to select roads with four or more lanes might look like:

```
Lanes >= 4
```

Similarly, a filter to select all `PipeNetworks` that have at least one `Pipe` in the proposed state might look like:

```
Pipes.state = "proposed"
```

Furthermore, a filter to select all existing parcels whose owner contains the text “Smith” might look like:

```
state = "existing" and owner like "%Smith%"
```

Finally, a filter to select all parcels that are either affected or encroached upon by some change might look like:

```
state in ("affected", "encroached")
```

Language Issues

There are a number of language issues to be considered when working with classes in the Filter, Expression, and Geometry packages:

- Provider-specific constraints on text
- Filter grammar
- Expression grammar
- Filter and Expression keywords
- Data types
- Operators
- Special characters
- Geometry value

Provider-Specific Constraints on Filter and Expression Text

Some providers may have reserved words that require special rules when used with filters and expressions. For more information, see [Oracle Reserved Words Used with Filter and Expression Text](#) (page 215).

Filter Grammar

The rules for entering filter expressions are described in the following sections using BNF notation. For more information about BNF notation, see <http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html>.

The `FdoFilter::Parse()` method supports the following filter grammar:

```

<Filter> ::= '(' Filter ')'
| <LogicalOperator>
| <SearchCondition>
<LogicalOperator> ::= <BinaryLogicalOperator>
| <UnaryLogicalOperator>
<BinaryLogicalOperator> ::=
<Filter> <BinaryLogicalOperations> <Filter>
<SearchCondition> ::= <InCondition>
| <ComparisonCondition>
| <GeometricCondition>
| <NullCondition>
<InCondition> ::= <Identifier> IN '(' ValueExpressionCollection
')'
<ValueExpressionCollection> ::= <ValueExpression>
| <ValueExpressionCollection> ',' <ValueExpression>
<ComparisonCondition> ::=
<Expression> <ComparisonOperations> <Expression>
<GeometricCondition> ::= <DistanceCondition>
| <SpatialCondition>
<DistanceCondition> ::=
<Identifier> <DistanceOperations> <Expression> <distance>
<NullCondition> ::= <Identifier> NULL
<SpatialCondition> ::=
<Identifier> <SpatialOperations> <Expression>
<UnaryLogicalOperator> ::= NOT <Filter>
<BinaryLogicalOperations> ::= AND | OR
<ComparisonOperations> ::=
= // EqualTo (EQ)
<> // NotEqualTo (NE)
> // GreaterThan (GT)
>= // GreaterThanOrEqualTo (GE)
< // LessThan (LT)
<= // LessThanOrEqualTo (LE)
LIKE // Like
<DistanceOperations> ::= BEYOND | WITHINDISTANCE
<distance> ::= DOUBLE | INTEGER
<SpatialOperations> ::= CONTAINS | CROSSES | DISJOINT
| EQUALS | INTERSECTS | OVERLAPS | TOUCHES | WITHIN | COVEREDBY |
INSIDE

```

Expression Grammar

The `FdoExpression::Parse()` method supports the following expression grammar:

```
<Expression> ::= '(' Expression ')'  
| <UnaryExpression>  
| <BinaryExpression>  
| <Function>  
| <Identifier>  
| <ValueExpression>  
<BinaryExpression> ::=  
<Expression> '+' <Expression>  
| <Expression> '-' <Expression>  
| <Expression> '*' <Expression>  
| <Expression> '/' <Expression>  
<DataValue> ::=  
TRUE  
| FALSE  
| DATETIME  
| DOUBLE  
| INTEGER  
| STRING  
| BLOB  
| CLOB  
| NULL  
<Function> ::= <Identifier> '(' <ExpressionCollection> ')'  
<ExpressionCollection> ::=  
| <Expression>  
| <ExpressionCollection> ',' <Expression>  
<GeometryValue> ::= GEOMFROMTEXT '(' STRING ')'  
<Identifier> ::= IDENTIFIER  
<ValueExpression> ::= <LiteralValue> | <Parameter>;  
<LiteralValue> ::= <GeometryValue> | <DataValue>  
<Parameter> ::= PARAMETER | ':' STRING  
<UnaryExpression> ::= '-' <Expression>
```

Expression Operator Precedence

The precedence is shown in YACC notation, that is, the highest precedence operators are at the bottom.

```
%left Add Subtract  
%left Multiply Divide  
%left Negate
```

Filter and Expression Keywords

The following case-insensitive keywords are reserved in the language, that is, they cannot be used as identifier or function names:

```
AND BEYOND COMPARE CONTAINS COVEREDBY CROSSES DATE  
DISJOINT DISTANCE EQUALS FALSE GeomFromText IN INSIDE  
INTERSECTS LIKE NOT NULL OR OVERLAPS RELATE SPATIAL TIME  
TIMESTAMP TOUCHES TRUE WITHIN WITHINDISTANCE
```

Data Types

The available data types are described in this section.

Identifier

An identifier can be any alphanumeric sequence of characters other than a keyword. Identifiers can be enclosed in double quotes to allow special characters and white space. If you need to include a double quote character inside an identifier, double the character, for example "abc""def".

Parameter

Parameters are defined by a colon followed by alphanumeric characters. The FDO filter language extends SQL to allow for a literal string to follow the colon to allow blanks (and other possibilities), for example, :'Enter Name'.

Determine whether parameters are supported by the FDO Provider you are using by checking SupportParameters on the Connection interface.

String

Strings are literal constants enclosed in single quotes. The FDO filter language also supports the special characters (left and right single quotes) that Microsoft Word uses to automatically replace the single quote character typed from the keyboard. If you need to include a single quote character inside a string you can double the character, for example 'aaa"bbb'.

Integer

Integers allow only decimal characters with an optional unary minus sign. Unary plus is not supported.

(-)[[0-9]]

Double

Floating point numbers have a decimal point, can be signed (-), and include an optional exponent (e{[0-9]}).

NOTE If an integer is out of the 32-bit precision range, it is converted to floating point.

Examples:

```
-3.4
12345678901234567
1.2e13
```

DateTime

Date and time are parsed using the standard SQL literal strings:

```
DATE 'YYYY-MM-DD'
TIME 'HH:MM:SS[.sss]'
TIMESTAMP 'YYYY-MM-DD HH:MM:SS[.sss]'
```

For example:

```
DATE '1971-12-24'
TIMESTAMP '2003-10-23 11:00:02'
```

NOTE The BLOB and CLOB strings are currently not supported. If you need to support binary input, use parameters.

Operators

The following operators are special characters common to SQL and most programming languages:

BinaryOperations

These binary operations are available:

+ Add (for compatibility with SQL string concatenation may also be defined using "||")

- Subtract

* Multiply

/ Divide

UnaryOperations

These unary operation are available:

- Negate

Comparison Operations

These comparison operations are available:

= EqualTo (EQ)

<> NotEqualTo (NE)

> GreaterThan (GT)

>= GreaterThanOrEqualTo (GE)

< LessThan (LT)

<= LessThanOrEqualTo (LE)

Operator Precedence

The following precedence is shown from highest to lowest:

Negate NOT

Multiply Divide

Add Subtract

EQ NE GT GE LT LE

AND

OR

Special Character

The following special characters are used in ExpressionCollections and ValueExpressions to define function arguments and IN conditions:

- (Left Parenthesis
- , Comma
-) Right Parenthesis

The Colon (:) is used in defining parameters and the Dot (.) can be included in real numbers and identifiers.

Geometry Value

Geometry values are handled using a function call `GeomFromText('FGF Text string')`, as is typical in an SQL query.

The Autodesk extension to WKT, referred to as FGF Text, is a superset of WKT (that is, you can enter WKT as valid FGF Text strings). Dimensionality is optional. It can be XY, XYM, XYZ, or XYZM. If it is not specified, it is assumed to be XY. For more information about FGF Text, see [FGF Text](#) (page 148).

NOTE Extra ordinates are ignored, rather than generating an error during FGF text parsing. For example, in the string "POINT (10 11 12)", the '12' is ignored because the dimensionality is assumed to be XY.

The following is the grammar definition for FGF Text:

```
<FGF Text> ::= POINT <Dimensionality> <PointEntity>
| LINESTRING <Dimensionality> <LineString>
| POLYGON <Dimensionality> <Polygon>
| CURVESTRING <Dimensionality> <CurveString>
| CURVEPOLYGON <Dimensionality> <CurvePolygon>
| MULTIPOINT <Dimensionality> <MultiPoint>
| MULTILINESTRING <Dimensionality> <MultiLineString>
| MULTIPOLYGON <Dimensionality> <MultiPolygon>
| MULTICURVESTRING <Dimensionality> <MultiCurveString>
| MULTICURVEPOLYGON <Dimensionality> <MultiCurvePolygon>
```

```

| GEOMETRYCOLLECTION <GeometryCollection>
<PointEntity> ::= '(' <Point> ')'
<LineString> ::= '(' <PointCollection> ')'
<Polygon> ::= '(' <LineStringCollection> ')'
<MultiPoint> ::= '(' <PointCollection> ')'
<MultiLineString> ::= '(' <LineStringCollection> ')'
<MultiPolygon> ::= '(' <PolygonCollection> ')'
<GeometryCollection : '(' <FGF Collection Text> ')'
<CurveString> ::= '(' <Point> '(' <CurveSegmentCollection> ')' ')'
<CurvePolygon> ::= '(' <CurveStringCollection> ')'
<MultiCurveString> ::= '(' <CurveStringCollection> ')'
<MultiCurvePolygon> ::= '(' <CurvePolygonCollection> ')'
<Dimensionality> ::= // default to XY
| XY
| XYZ
| XYM
| XYZM
<Point> ::= DOUBLE DOUBLE
| DOUBLE DOUBLE DOUBLE
| DOUBLE DOUBLE DOUBLE DOUBLE
<PointCollection> ::= <Point>
| <PointCollection ',' <Point>
<LineStringCollection> ::= <LineString>
| <LineStringCollection ',' <LineString>
<PolygonCollection> ::= <Polygon>
| <PolygonCollection ',' <Polygon>
<FGF Collection Text> ::= <FGF Text>
| <FGF Collection Text> ',' <FGF Text>
<CurveSegment> ::= CIRCULARARCSEGMENT '(' <Point> ',' <Point> ')'

```

```

| LINESTRINGSEGMENT '(' <PointCollection> ')'
<CurveSegmentCollection> ::= <CurveSegment>
| <CurveSegmentCollection> ',' <CurveSegment>
<CurveStringCollection> ::= <CurveString>
| <CurveStringCollection> ',' <CurveString>
<CurvePolygonCollection> ::= <CurvePolygon>
| <CurvePolygonCollection> ',' <CurvePolygon>

```

The only other token type is DOUBLE, representing a double precision floating point values. Integer (non-decimal point) input is converted to DOUBLE in the lexical analyzer.

Examples of the Autodesk extensions include:

```
POINT XY (10 11) // equivalent to POINT (10 11)
```

```
POINT XYZ (10 11 12)
```

```
POINT XYM (10 11 1.2)
```

```
POINT XYZM (10 11 12 1.2)
```

```
GEOMETRYCOLLECTION (POINT xyz (10 11 12),POINT XYM (30 20 1.8),
LINESTRING XYZM(1 2 3 4, 3 5 15, 3 20 20))
```

```
CURVESTRING (0 0 (LINESTRINGSEGMENT (10 10, 20 20, 30 40))))
```

```
CURVESTRING (0 0 (CIRCULARARCSEGMENT (11 11, 12 12),
LINESTRINGSEGMENT (10 10, 20 20, 30 40)))
```

```
CURVESTRING (0 0 (ARC (11 11, 12 12), LINESTRINGSEGMENT (10 10, 20
20, 30 40)))
```

```
CURVESTRING XYZ (0 0 0 (LINESTRINGSEGMENT (10 10 1, 20 20 1, 30 40
1)))
```

```
MULTICURVESTRING ((0 0 (LINESTRINGSEGMENT (10 10, 20 20, 30 40))), (0
0 (ARC (11 11, 12 12), LINESTRINGSEGMENT (10 10, 20 20, 30 40))))
```

```
CURVEPOLYGON (((0 0 (LINESTRINGSEGMENT (10 10, 10 20, 20 20), ARC
(20 15, 10 10))), (0 0 (ARC (11 11, 12 12), LINESTRINGSEGMENT (10 10, 20
20, 40 40, 90 90))))
```

```
MULTICURVEPOLYGON (((0 0 (LINESTRINGSEGMENT (10 10, 10 20, 20 20),
ARC (20 15, 10 10))), (0 0 (ARC (11 11, 12 12), LINESTRINGSEGMENT (10 10,
20 20, 40 40, 90 90))), (0 0 (LINESTRINGSEGMENT (10 10, 10 20, 20 20),
```

ARC (20 15, 10 10))), (0 0 (ARC (11 11, 12 12), LINESTRINGSEGMENT (10 10,
20 20, 40 40, 90 90))))))

The Geometry API

12

This chapter describes the FdoGeometry API (hereafter called the “Geometry API”) and explains the various geometry types and formats.

Introduction

The Geometry API supports specific Autodesk applications and APIs, including FDO (Feature Data Objects). This API consists of the following components:

- a Geometry Type package (all through fully encapsulated interfaces)
- an Abstract Geometry Factory
- a Concrete Geometry Factory for FGF

You can work with the Geometry API in several different ways:

- FGF (Feature Geometry Format)
- FGF Text
- Abstract Geometry Factory

FGF and WKB

WKB is a memory layout used to store geometry features. This format was created by the OpenGIS organization to allow the efficient exchange of geometry data between different components in an FDO system. Most pieces of the original specification defining the WKB format are in the document, *99-050.pdf*, the OpenGIS Simple feature specification for OLE/COM that can be found at www.opengis.org.

FGF is an extended version of the Well Known Binary (WKB) format. The two formats differ in the following ways:

- WKB defines a byte order of the data in every piece of geometry. This is stored as a byte field, which may change the memory alignment from word to byte. In FGF, only one memory alignment type is supported, which is the same alignment type used by the .NET framework and Windows; the encoding uses the little-endian byte order format. As a result, the byte flag does not need to be stored.
- WKB is defined as a 2D format only. This is insufficient to represent 3D points, polylines and polygons. In FGF, the dimension flag has been added. In particular, a flag is included for each geometry piece to indicate whether the geometry is 2D, 3D, or even 4D (storing a measure value as used by dynamic segmentation).
- FGF includes geometry types that are not yet covered by any WKB specification.

FGF Binary Specification

In this section, the memory layout of each simple geometry type is described. The format is based on the OGC specification, which is built on the memory layout of a C++ struct. All arrays have a computable size and are inline; they do not point to a different location in memory. This format allows streaming of geometry data.

First, the different data types, their size, and memory layout are discussed.


```

// double == 8byte IEEE double number in little endian encoding.
// int == 4 byte integer in little endian encoding
// the type of the geometry
enum GeometryType : int
{
    None = 0,
    Point = 1,
    LineString = 2,
    Polygon = 3,
    MultiPoint = 4,
    MultiLineString = 5,
    MultiPolygon = 6,
    MultiGeometry = 7,
    CurveString = 10,
    CurvePolygon = 11,
    MultiCurveString = 12,
    MultiCurvePolygon = 13
}

```

Coordinate Types

This is a bit field, for example, `xym == coordinateDimensionality.XY | coordinateDimensionality.M`. The following sequence defines the type of coordinates used for this object:

```

enum CoordinateDimensionality : FdoInt32
{
    XY = 0,
    Z = 1,
    M = 2
}

```

Basic Geometry

The following sequence establishes the basic pure geometry:

```

struct Geometry
{
    int geomType;
    CoordinateDimensionality type;
}

```

Notation Definition

The following sequence defines a notation used to specify geometries within a byte stream.

```

// Define a notation within this specification
// int PositionSize(geometry)
// {
//   if (geometry.type == CoordinateDimensionality.XY |
//       // CoordinateDimensionality.M ||
//       geometry.type == CoordinateDimensionality.XY |
//       // CoordinateDimensionality.Z)
//     return 3;
//   if (geometry.type == CoordinateDimensionality.XY |
//       // CoordinateDimensionality.M | CoordinateDimensionality.Z)
//     return 4
//   return 2;
// }
struct Point // : Geometry
{
  int geomType; // == GeometryType.Point;
  CoordinateDimensionality type; // all types allowed
  double[] coords; // size = PositionSize(this)
}
struct LineString
{
  int geomType;
  CoordinateDimensionality type;
  int numPts; // >0
  double[] coords; // size = numPts* PositionSize(this)
}
struct MultiPoint
{
  int geomType;
  int numPoints; // > 0
  Point[] points; // size = numPoints
}
struct MultiLineString
{
  int geomType;
  int numLineStrings; // >= 0
  LineString[] lineStrings; // size = numLineStrings
}
// building block for polygons, not geometry by itself
struct LinearRing
{
  int numPts; // >0
  double[] coords; // size = numPts* PositionSize(polygon)
}

```

```

}
struct Polygon
{
    int geomType;
    CoordinateDimensionality type;
    int numRings; // >= 1 as there has to be at least one ring
    LinearRing[] lineStrings; // size = numRings
}
struct MultiPolygon
{
    int geomType;
    int numPolygons; // >= 0
    Polygon[] polygons; // size = numPolygons
}
struct MultiGeometry
{
    int geomType;
    int numGeom; // >= 0
    Geometry[] geometry; // size = numGeom
}
enum CurveElementType : int
{
    LineString = 1,
    CircularArc = 2
}
struct CurveStringElement
{
    int CurveElementType;
}
struct LinearCurveStringElement
{
    int CurveElementType;
    int length;
    double[] coords; // size = this.length * PositionSize (this)
}
struct CircularArcCurveStringElement
{
    int CurveElementType; // == CurveElmentType.Arc
    double[] coords; // size = 2 * PositionSize(this)
}
struct CurveString
{
    int geomType;

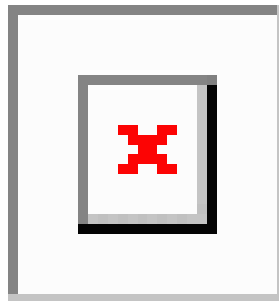
```

```

CoordinateDimensionality type; // all types allowed
double[] startPoint; // size = PositionSize(this)
int numElements; // >=0
CurveStringElement[] elements; // size = numElements
}
struct Ring
{
double[] startPoint; // size = PositionSize(this)
int numElements; // >=0
CurveStringElement[] elements; // size = numElements
}
struct MultiCurveString
{
int geomType;
int numCurveStrings; // >= 0
CurveString[] curveStrings; // size = numCurveStrings
}
struct CurvePolygon
{
int geomType; ;
CoordinateDimensionality type;
int numRings; // >=1 as there has to be at least one ring
Ring[] rings; // size = numRings
}
struct MultiCurvePolygon
{
int geomType;
int numPolygons; // >=0
CurvePolygon[] polygons; // size = numElements
}

```

In the following example a polygon is formatted within a byte array representing the stream according to the FGF specification.



T = 3 stands for `GeometryType == GeometryType.Polygon`

CT = 0 stands for `CoordinateDimensionality == CoordinateDimensionality.XY`

NR = 2 stands for number of rings = 2

NP = 3 stands for number of points = 3

FGF Text

FGF Text is the textual analogue to the binary FGF format. It is a superset of the OGC WKT format. XY dimensionality is the default, and is optional. FGF Text can be used to represent any geometry value in the Geometry API, whether or not it originates from the FGF geometry factory. Conversions are done with the following methods:

- `FdoGeometryFactoryAbstract:: CreateGeometry(FdoString* text);`
- `FdoIGeometry:: GetText();`

A BNF for the FGF textual specification is contained in the topic [Geometry Value](#) (page 137).

Abstract and Concrete Classes

The Geometry API is almost completely abstract. It provides an object-oriented interface to geometry values. All objects in the Geometry API have factory methods in the `FdoGeometryFactorytAbstract` class. One default implementation is provided, based on FGF in-memory binary storage. It is accessible via the concrete class `FdoFgfGeometryFactory`.

NOTE The `FdoFgfGeometryFactory` employs object pooling for many of the data types in the API. While many methods appear to be executing “Create” or “Get” actions, they are, in fact, accessing object pools, thus avoiding costly operations on the memory heap.

All of the other classes in the Geometry API with the exception of two relate to the main abstract type, `FdoIGeometry`. They either derive from it or are components of it.

The two exception concrete classes are:

- `FdoDirectPositionImpl`, a small helper class implementing `FdoIDirectPosition`.

- `FdoEnvelopeImpl`, a small helper class implementing `FdoIEnvelope`.

Geometries in FGF format can be exchanged between software components without depending on the Geometry API itself, because they are not genuine geometry “objects.” FGF content is based on byte arrays. It is handled through a simple `FdoByteArray` class that is not specific to geometry.

Geometry Types

The Geometry types comprise the Global Enum `FdoGeometryType`. The following are Geometry types:

- 0 `FdoGeometryType_None` Indicates no specific type; used for “unknown”, “do not care” or an incompletely constructed Geometry object.

NOTE `FdoGeometryType_None` does not represent an instantiable type. An FDO client should not expect an FDO provider to list support for it in its capabilities.

- 1 `FdoGeometryType_Point` Point type (`FdoIPoint`).
- 2 `FdoGeometryType_LineString` LineString type (`FdoILineString`).
- 3 `FdoGeometryType_Polygon` Polygon type (`FdoIPolygon`).
- 4 `FdoGeometryType_MultiPoint` MultiPoint type (`FdoIMultiPoint`).
- 5 `FdoGeometryType_MultiLineString` MultiLineString type (`FdoIMultiLineString`).
- 6 `FdoGeometryType_MultiPolygon` MultiPolygon type (`FdoIMultiPolygon`).
- 7 `FdoGeometryType_MultiGeometry` MultiGeometry type (`FdoIMultiGeometry`).
- 10 `FdoGeometryType_CurveString` CurveString type (`FdoICurveString`).
- 11 `FdoGeometryType_CurvePolygon` CurvePolygon type (`FdoICurvePolygon`).
- 12 `FdoGeometryType_MultiCurveString` MultiCurveString type (`FdoIMultiCurveString`).
- 13 `FdoGeometryType_MultiCurvePolygon` MultiCurvePolygon type (`FdoIMultiCurvePolygon`).

Mapping Between Geometry and Geometric Types

The FDO API `GeometricType` enumeration of `GeometricProperty` gives the client application some knowledge of which geometry types comprise the geometric property so that it can present the user with an intelligent editor for selecting styles for rendering the geometry. In particular, `GeometricType` relates to shape dimensionality of geometries allowed in FDO geometric properties. The nearest analogues in the Geometry API are:

- `FdoDimensionality`, which pertains to ordinate (not shape) dimensionality of geometry values.
- `FdoGeometryType`, which has types whose abstract base types map to `GeometricType`

The `GeometricType` enumeration is as follows:

- `Point = 0x01`, // Point Type Geometry
- `Curve = 0x02`, // Line and Curve Type Geometry
- `Surface = 0x04`, // Surface (or Area) Type Geometry
- `Solid = 0x08`, // Solid Type Geometry

NOTE The enumeration defines a bit mask and the `GetGeometricTypes` and `SetGeometricTypes` methods take and return an integer. This is to allow a geometry property to be of more than one type. For example, the call:

```
geometricProperty.SetGeometricTypes(Point | Surface);
```

would allow the geometric property to represent either point type geometry or surface type geometry (polygons).

Spatial Context

Spatial Context is a coordinate system with an identity. Any geometries that are to be spatially related must be in a common spatial context.

Providing an identify for each coordinate system supports separate workspaces, such as schematic diagrams, which are non-georeferenced. However, there are also georeferenced cases. In general, two users may create drawings using the same default spatial parameters (for example, rectangular and 10,000x10,000) that have nothing to do with each other. If their drawings are to be put into a common database, the spatial context capability of FDO preserves the container aspect of the data along with the spatial parameters.

The FDO Spatial Context Commands are part of the FDO API. They support control over Spatial Contexts in the following ways:

- **Metadata control.** Creates and deletes Spatial Contexts.
- **Active Spatial Context.** A session setting to specify which Spatial Context to use by default while storing/retrieving geometries and performing spatial queries.

There is a default Spatial Context for each database. Its attributes (such as coordinate system) are specified when the database is created. This Spatial Context is the active one in any FDO session until a Spatial Context Command is used to change this state. The default Spatial Context's identifier number is 0 (zero).

Spatial contexts have two tolerance attributes: XYTolerance and ZTolerance. The tolerances are in distance units that depend on the coordinate system in use. Geodetic coordinate systems typically have "on the ground" linear distance units instead of the angular (that is, degrees, minutes or seconds) units used for positional ordinates. The meter is the most common unit. Most non-geodetic systems are rectilinear and use the same unit for positional ordinates and distances, for example, meters or feet.

Specify Dimensionality When Creating Geometries Using String Specifications

When creating a 3D geometry from string specifications, you must specify the Dimensionality argument `xyz` explicitly, because the default dimensionality is XY, and the geometry factory code will only process the first two ordinates.

The following code successfully creates `pointOne` as a 3D point, whereas `pointTwo` is created as a 2D point.

```
FdoFgfGeometryFactory * geometryFactory = FdoFgfGeometryFactory::GetInstance();
FdoIPoint * pointOne;
pointOne = dynamic_cast<FdoIPoint>(geometryFactory->CreateGeometry(L"GeomFromText('POINT XYZ (1 2 3)')"));
FdoPoint * pointTwo;
pointTwo = dynamic_cast<FdoPoint>(geometryFactory->CreateGeometry(L"GeomFromText('POINT (1 2 3)')"));

xmlFeatureFlags = FdoXmlFeatureFlags.Create( None, FdoXmlFlags.ErrorLevel_Normal, True, FdoXmlFeatureFlags.ConflictOption_Add )
```

Inserting Geometry Values

For information about geometry property values, see [Geometry Property Values](#) (page 111).

See [Example: Inserting an Integer, a String, and a Geometry Value](#) (page 112) for a code example that shows how to insert a Geometry value.

Introduction

This chapter contains code samples of common operations.

Some operations like connection must discuss differences among providers. References are made to both Autodesk proprietary providers and open source providers. Providers whose name begins with “Autodesk” are available only in commercial releases of Autodesk products. Providers whose name begins with “OSGeo” are available in releases of open source software at <http://fdo.osgeo.org>. Some of the open source providers are also available in commercial releases of Autodesk products.

Recommendations

- After iterating through readers such IFeatureReader, IDataReader, ISQLDataReader or DbUserReader, call the Close() or Dispose() method. Not doing so may cause unhandled exceptions at some future time. Be sure to enclose the statement that returns a reader in a try block so that you can close the reader in the catch block.

Registry

The registry contains the list of providers that are available in a particular FDO installation. FDO initializes the registry using the contents of the providers.xml file that is co-located with the FDO binaries.

The /FeatureProviderRegistry/FeatureProvider/Name elements in the providers.xml file contain fully qualified provider names. A fully qualified

provider name has four parts which are separated by dot (‘.’) characters. The parts are
<domain>.<shortName>.<majorReleaseNumber>.<minorReleaseNumber>.
There are two domains: Autodesk and OSGeo. The former indicates proprietary, and the latter open source, software.

Use Cases

- Get a fully qualified provider name in order to create a connection. The fully qualified name contains version numbers so having a dictionary that maps generic provider names to the currently loaded fully qualified name is useful.

C#

The following namespaces are used:

- OSGeo.FDO.ClientServices for FeatureAccessManager and ProviderCollection
- OSGeo.FDO for IProviderRegistry

```
IProviderRegistry registry = FeatureAccessManager.GetProviderRegistry();
ProviderCollection providers = registry.GetProviders();
// Map provider short name to fully qualified name
Dictionary<string, string> providerAbbrToFullName = new Dictionary<string, string>();
string shortName = null;
foreach (Provider provider in providers)
{
    shortName = provider.Name.Split('.')[1];
    providerAbbrToFullName[shortName] = provider.Name;
}
```

Connection

The connection object contains a connection string property that contains the connection parameter names and values. The connection string property can be set by using a connection property dictionary or by assigning a connection string directly to it. The property dictionary’s methods tell you the names of the connection parameters, their current values, and their optionality.

Some parameters must always be set for a connection to succeed, for example, the OSGeo.SDF File parameter, and the Autodesk.Oracle Username, Password, and Service parameters. Some parameters need not be set if you accept the default value. For example, the SDF ReadOnly parameter has a default setting of false. The Oracle DataStore parameter need not be set if your objective is to obtain the names of the available data stores within the data source. In this case connecting with Username, Password, and Service parameters results in a pending connection. You can then query the DataStore parameter to get the list of available data stores, set the DataStore parameter to the value of one of the list members and open the connection again, but this time to a fully operational state. If you already know the name of the data store that you want, you can set its value at the outset and open the connection fully without the intervening step.

The following table sets out the names of the connection parameters for each provider.

| Provider | Connection Parameters |
|--------------------|---|
| Autodesk.Oracle | Username, Password, Service, DataStore |
| Autodesk.Raster | DefaultRasterFileLocation |
| Autodesk.SqlServer | Username, Password, Service, DataStore |
| OSGeo.ArcSDE | Server, Instance, Username, Password, Datastore |
| OSGeo.Gdal | DefaultRasterFileLocation |
| OSGeo.KingOracle | Username, Password, Service, OracleSchema, KingFdoClass |
| OSGeo.MySQL | Username, Password, Service, DataStore |
| OSGeo.ODBC | UserId, Password, DataSourceName, ConnectionString, GenerateDefaultGeometryProperty |
| OSGeo.OGR | DataSource, ReadOnly |
| OSGeo.PostGIS | Username, Password, Service, DataStore |
| OSGeo.SDF | File, ReadOnly |
| OSGeo.SHP | DefaultFileLocation |

| Provider | Connection Parameters |
|---------------------|---|
| OSGeo.SQLiteSpatial | Username, Password, Service, DataStore |
| OSGeo.WFS | FeatureServer, Username, Password |
| OSGeo.WMS | FeatureServer, Username, Password, DefaultImageHeight |

Use Cases

- Create a connection to get the provider capabilities. See the [Capabilities](#) (page 157) topic.
- Get a list of the connection parameters.
- Determine the optionality of a parameter.
- Set the value of a parameter.
- Open and close connections.

C#

The following namespaces are used:

- OSGeo.FDO.ClientServices for FeatureAccessManager
- OSGeo.FDO for IConnectionManager
- OSGeo.FDO.Connections for IConnectionImp, IConnectionInfo, IConnectionPropertyDictionary, ConnectionState

Create a Connection

The connection is created in the closed state.

```

IConnectionManager connMgr = FeatureAccessManager.GetConnectionManager();
string fullName = providerAbbrToFullName["Oracle"]
IConnectionImp conn = connMgr.CreateConnection(fullName) as IConnectionImp;

```

Determine the Names of the Parameters

Before a connection can be opened, the connection parameters must be identified and given values. You can query a connection to identify these

parameters. The code block shows the retrieval of the connection parameter names into a string array.

```
IConnectionInfo connInfo = connection.ConnectionInfo;  
IConnectionPropertyDictionary properties = connInfo.ConnectionProperties;  
string[] names = properties.PropertyNames;
```

Determine the Optionality of a Parameter

The code block shows determining the optionality of the first connection parameter.

```
Boolean isRequired = properties.IsPropertyRequired(names[0]);
```

Set a Parameter

The code block shows the setting of the Username parameter.

```
properties.SetProperty("Username", "EXISTINGUSER");
```

Get the Connection String

The code block shows getting the connection string value from the connection.

```
string connectionString = connection.ConnectionString;
```

Open and Close a Connection

The code block shows opening and closing a connection. Valid connection states are Busy, Closed, Open, and Pending.

```
ConnectionState connState = connection.Open();  
ConnectionState connState = connection.Close();
```

Capabilities

The following code samples are a representative listing of the capability methods in each category. For a complete listing consult the API reference documentation.

The capabilities for a provider are statically defined. You create a connection object for the provider and use this object to query the capabilities. The connection does not have to be open.

Some capabilities are defined as booleans, for example, support for the DISTINCT operator in a SQL SELECT statement. Sample code for this is shown

in the Command section. Some capabilities are defined by arrays of enumerated type values, for example, the feature commands. The presence of the capability is determined by testing each value in the array for its equality with one of the values in the enumerated type range; sample code that tests for the presence of the create data store command is shown in the Command section.

The schema capabilities include a string type that defines the characters that may not appear in a schema name and two integer types that define the maximum decimal precision and scale.

Use Cases

- Get the capabilities of a provider.
- Write code that executes if the provider currently in use supports it.

C#

The following namespaces and types are used:

- OSGeo.FDO.Commands for CommandType
- OSGeo.FDO.Commands.Locking for LockType
- OSGeo.FDO.Commands.SpatialContext for SpatialContextExtentType
- OSGeo.FDO.Common for GeometryComponentType, GeometryType
- OSGeo.FDO.Connections.Capabilities for ICommandCapabilities, IConnectionCapabilities, IExpressionCapabilities, IFilterCapabilities, IGeometryCapabilities, IRasterCapabilities, ISchemaCapabilities, FunctionDefinitionCollection, FunctionDefinition, ReadOnlySignatureDefinitionCollection, SignatureDefinition, ReadOnlyArgumentDefinitionCollection, ArgumentDefinition, ThreadCapability
- OSGeo.FDO.Expression for ExpressionType
- OSGeo.FDO.Filter for ConditionType, DistanceOperations, SpatialOperations
- OSGeo.FDO.Schema for ClassType, DataType, PropertyType

Command

```
ICommandCapabilities cmdCapabilities = conn.CommandCapabilities;
int[] commands = cmdCapabilities.Commands;
Boolean supportsSelectDistinct = cmdCapabilities.SupportsSelectDistinct();
CommandType cmdType = (CommandType)commands[0];
Boolean supportsCreateDataStore = false;
for (int i = 0; i < commands.GetLength(0); i++) {
    if (commands[i] == (int)CommandType.CommandType_CreateDataStore)
        supportsCreateDataStore = true;
}
```

Connection

```
IConnectionCapabilities connCapabilities = conn.ConnectionCapabilities;
LockType[] lockTypes = connCapabilities.LockTypes;
SpatialContextExtentType[] spatialContextExtentTypes = connCapabilities.SpatialContextTypes;
ThreadCapability threadCapability = connCapabilities.ThreadCapability;
Boolean supportsConfiguration = connCapabilities.SupportsConfiguration();
```

Expression

```
IExpressionCapabilities exprCapabilities = connection.Expression
Capabilities;
ExpressionType[] expressionTypes = exprCapabilities.Expression
Types;
FunctionDefinitionCollection functions = exprCapabilities.Func
tions;
FunctionDefinition function = functions[0];
ReadOnlySignatureDefinitionCollection signatureDefs = function.Sig
natures;
SignatureDefinition signatureDef = signatureDefs[0];
PropertyType returnPropertyType = signatureDef.ReturnPropertyType;
if (returnPropertyType == PropertyType.PropertyType_DataProperty)
{
    DataType returnDataType = signatureDef.ReturnType;
}
ReadOnlyArgumentDefinitionCollection arguments = signatureDef.Ar
guments;
ArgumentDefinition argDef = arguments[0];
PropertyType argPropertyType = argDef.PropertyType;
if (argPropertyType == PropertyType.PropertyType_DataProperty)
{
    DataType argDataType = argDef.DataType;
}
```

Filter

```
IFilterCapabilities filterCapabilities = conn.FilterCapabilities;
Boolean supportsGeodesicDistance = filterCapabilities.Supports
GeodesicDistance();
ConditionType[] conditionTypes = filterCapabilities.ConditionTypes;
DistanceOperations[] distanceOperations = filterCapabilities.Dis
tanceOperations;
SpatialOperations[] spatialOperations = filterCapabilities.Spatia
lOperations;
```

Geometry

```
IGeometryCapabilities geometryCapabilities = conn.GeometryCapabilities;  
GeometryType[] geometryTypes = geometryCapabilities.GeometryTypes;  
GeometryComponentType[] geometryComponentTypes = geometryCapabilities.GeometryComponentTypes;  
int dimensionalities = geometryCapabilities.Dimensionalities;  
Boolean HasZ = (dimensionalities & 1) == 1;  
Boolean HasM = (dimensionalities & 2) == 2;
```

NOTE The dimensionalities integer contains bit-field definitions. XY is 0 because all spatial data has at least an X and Y coordinate. Z is 1 and M is 2. Use bitwise AND (&) to combine and bitwise OR (|) to separate.

Raster

```
IRasterCapabilities rasterCapabilities = conn.RasterCapabilities;  
Boolean supportsRaster = rasterCapabilities.SupportsRaster();
```

Schema

```
ISchemaCapabilities schemaCapabilities = conn.SchemaCapabilities;  
DataType[] dataTypes = schemaCapabilities.DataTypes;  
ClassType[] classTypes = schemaCapabilities.ClassTypes;  
string reservedCharactersForName = schemaCapabilities.ReservedCharactersForName;  
int maximumDecimalPrecision = schemaCapabilities.MaximumDecimalPrecision;  
DataType[] supportedAutoGeneratedTypes = schemaCapabilities.SupportedAutoGeneratedTypes;  
DataType[] supportedIdentityPropertyTypes = schemaCapabilities.SupportedIdentityPropertyTypes;  
Boolean supportsAutoIdGeneration = schemaCapabilities.SupportsAutoIdGeneration;
```

Data Store

The Sql Server, SQL Server Spatial, Oracle, MySQL, PostGIS, and SDF providers support the following data store commands, create, destroy, and, with the exception of SDF, list. SDF is a file-based provider, and the rest are RDBMS-based.

By default the Sql Server, Oracle, MySQL, and SDF providers create data stores that contain FDO metadata. By default the SQL Server Spatial provider creates a data store that does not contain the FDO metadata. To make this provider create a data store that does contain the FDO metadata, you must set the `IDataStorePropertyDictionary` property `IsFdoEnabled` to true. The PostGIS provider creates a data store without the FDO metadata.

A provider uses the FDO metadata to:

- Assign a default spatial context to a Geometric Property Definition during schema creation.

When connected to the SDF Provider, the `CreateDataStore` command creates an SDF file, and the `DestroyDataStore` command deletes it. There cannot be multiple data stores in an SDF file and so there is no need for the list command. Unlike the RDBMS-based providers, you do not have to open a pending connection in order to create and execute the `CreateDataStore` command. Before executing the `DestroyDataStore` command close the connection.

For the RDBMS-based providers creating a data store means creating a container within a database. There can be many data stores within a database, and the list command lists them.

To create a data store in one of these RDBMS-based providers you open a pending connection using an administrator account name and password and then execute the `CreateDataStore` command. Once you have created the data store, you close the administrator connection and open a normal user connection. This time you give a value to the `DataStore` connection parameter and open the connection to a fully operational state. Before executing the `DestroyDataStore` command, close the connection, unset the `DataStore` connection parameter and reopen it using the administrator account name and password in the pending state.

C#

The following namespaces are used:

- `OSGeo.FDO.Commands.DataStore`
- `OSGeo.FDO.Commands`

```

    ICreateDataStore createDS = conn.CreateCommand(CommandType.Com
mandType_CreateDataStore) as ICreateDataStore;
IDataStorePropertyDictionary properties = createDS.DataStoreProp
erties;
properties.SetProperty(paramName, paramValue);
createDS.Execute();
    IListDataStores listDS = conn.CreateCommand(CommandType.Command
Type_ListDataStores) as IListDataStores;
IDataStoreReader reader = listDS.Execute();
    IDestroyDataStore destroyDS = conn.CreateCommand(CommandType.Com
mandType_DestroyDataStore) as IDestroyDataStore;
IDataStorePropertyDictionary properties = destroyDS.DataStoreProp
erties;
properties.SetProperty(paramName, paramValue);
destroyDS.Execute();

```

User Management

The providers may be categorized into five groups with respect to user management. The grouping criteria is the method used to create the username.

The first group consists of the Autodesk.Oracle provider, which has access to an FDO API for user management.

The second group consists of MySQL, SqlServer, and SQLServerSpatial. These providers support the SQLCommand, which can be used to create users.

The third group consists of ODBC. A username and password may be required to connect depending on the back-end data store. On XP you use the ODBC Data Source Administrator tool to configure the username.

The third group consists of the web server providers: WFS and WMS. A username and password may be required depending on the site.

The fourth group consists of the file-based providers, which do not require a username and password.

The FDO User Management API

The Autodesk.Oracle provider has access to an FDO API for user management. Before you can add a user, you must have added the `f_user_role` to the database as described in the Preparations topic at the beginning of this chapter. During the add user operation, the `f_user_role` is assigned to the new user. You must open a pending connection using an administrator account name and password to add or drop a user.

C#

The Autodesk.Fdo.Fdo.Utilities.UserMgr namespace is used.

```
IUserManagerImp userMgr = new IUserManagerImp(connection);
userMgr.AddUser("AUserName", "APassword");
DbUserReader reader = userMgr.GetDbUsers();
userMgr.DropUser("AUserName");
```

The SqlCommand Method for User Management

| Provider | Add a user | Drop a user | List users |
|--|--|--|--|
| MySQL | CREATE USER 'AUserName' IDENTIFIED BY 'APassword'; GRANT ALL ON *.* TO 'AUs- erName'; | DROP USER 'AUser- Name'; | SELECT user FROM mysql.user WHERE user = 'AUserName'; |
| SqlServer and SqlServerSpa- tial | CREATE LOGIN AUserName WITH password = 'APass- word'; EXEC sp_addsrvrole- member @loginame = 'AUs- erName', @rolename = 'sysadmin'; | EXEC sp_droplogin @loginame = 'AUser- Name'; | SELECT name FROM sys.serv- er_principals WHERE name = 'AUserName'; |

C#

The code samples use the SQL syntax supported by MySQL in the strings passed to the FDO API command calls.

The following namespaces are used:

- OSGeo.FDO.Commands.SQL
- OSGeo.FDO.Commands

```
// create a user
ISQLCommand sqlCommand = conn.CreateCommand(CommandType.Command
Type_SQLCommand) as ISQLCommand;
sqlCommand.SQLStatement = "create user 'AUserName' identified by
'APassword'";
int returnVal = sqlCommand.ExecuteNonQuery();
sqlCommand.SQLStatement = "grant all on *.* to 'AUserName'";
returnVal = sqlCommand.ExecuteNonQuery();
```

```
// drop a user
ISQLCommand sqlCommand = conn.CreateCommand(CommandType.Command
Type_SQLCommand) as ISQLCommand;
sqlCommand.SQLStatement = "drop user 'AUserName'";
int returnVal = sqlCommand.ExecuteNonQuery();

// list users
ISQLCommand sqlCommand = conn.CreateCommand(CommandType.Command
Type_SQLCommand) as ISQLCommand;
sqlCommand.SQLStatement = "select user from mysql.user where user
= 'AUserName'";
ISQLDataReader reader = sqlCommand.ExecuteReader();
reader.Close();
```

Spatial Context

A spatial context specifies a coordinate system and an extent for a set of geometries. The spatial context object has a Name property. The value of the Name property is assigned to the SpatialContextAssociation property of the GeometricPropertyDefinition object that defines a feature or non-feature geometry belonging to a feature class. In this way a coordinate system and extent is associated with geometry values inserted into the data store as part of instances of the class. Example code showing this association is contained in the [Basic Schema Operations](#) (page 168) topic.

NOTE It is worth emphasizing that each geometric property definition in a feature class definition can have a different spatial context. This mechanism replaces that of the active spatial context.

The data stores created by the Oracle, Sql Server, MySQL, and optionally, the SQL Server Spatial providers contain metadata that supports the assignment of a default spatial context to a Geometric Property Definition during schema creation in the event that the user has not explicitly assigned one. The attribute values of the default spatial context are set out in the following table.

| Attribute | Value |
|---------------------|------------|
| Name | Default |
| CoordinateSystem | <no value> |
| CoordinateSystemWkt | <no value> |

| Attribute | Value |
|------------|--|
| ExtentType | SpatialContextExtentType_Static |
| Extent | POLYGON((-2000000 -2000000, 2000000 -2000000, 2000000 2000000, -2000000 2000000, -2000000 -2000000)) |

C#

The code samples show how to create a spatial context and how to get the spatial contexts from a data store.

The following namespaces are used:

- OSGeo.FDO.Commands.SpatialContext
- OSGeo.FDO.Geometry
- OSGeo.FDO.Connections


```

// get the spatial contexts from a data store
IGetSpatialContexts getSpatialContexts = connection.CreateCommand(
    CommandType.CommandType_GetSpatialContexts) as IGetSpatialContexts;
ISpatialContextReader reader = getSpatialContexts.Execute();
string name = null;
string coordSys = null;
string wellKnownText = null;
string desc = null;
FgfGeometryFactory geomFactory = new FgfGeometryFactory();
IGeometry geom = null;
string extent = null;
SpatialContextExtentType extentType;
double xyTolerance;
double zTolerance;
while (reader.ReadNext())
{
    name = reader.GetName();
    coordSys = reader.GetCoordinateSystem();
    wellKnownText = reader.GetCoordinateSystemWkt();
    desc = reader.GetDescription();
    geom = geomFactory.CreateGeometryFromFgf(reader.GetExtent());
    extent = geom.Text;
    extentType = reader.GetExtentType();
    xyTolerance = reader.GetXYTolerance();
    zTolerance = reader.GetZTolerance();
}
reader.Dispose();

```

```

// create a spatial context whose coordinate system is WGS84 and
// whose extent is the maximum
// create the extent
DirectPositionImpl lowerLeft = new DirectPositionImpl();
lowerLeft.X = -180.0;
lowerLeft.Y = -90.0;
DirectPositionImpl upperRight = new DirectPositionImpl();
upperRight.X = 180.0;
upperRight.Y = 90.0;
IEnvelope envelope = geomFactory.CreateEnvelope(lowerLeft, upper
Right);
IGeometry geom = geomFactory.CreateGeometry(envelope);
byte[] extent = geomFactory.GetFgf(geom);
ICreateSpatialContext createSpatialContext = connection.CreateCom
mand(CommandType.CommandType_CreateSpatialContext) as ICreateSpa
tialContext;
createSpatialContext.CoordinateSystem = "";
createSpatialContext.CoordinateSystemWkt = wgs84Wkt;
createSpatialContext.Description = "This Coordinate System is used
for GPS.";
// a static extent is defined once and never changed
// a dynamic extent grows to accommodate the geometries added to
the data store
createSpatialContext.ExtentType = SpatialContextExtentType.Spatial
ContextExtentType_Static;
createSpatialContext.Extent = extent;
// the value of Name is assigned to the SpatialContextAssociation
attribute of a GeometricPropertyDefinition object during the
creation of the logical feature schema
// in this way a coordinate system and extent is associated with
the geometries
createSpatialContext.Name = "WGS84 Spatial Context";
createSpatialContext.Execute();

```

Basic Schema Operations

All providers allow you to describe a schema, but only SDE, SHP, MySQL, PostGIS, SqlServer, SqlServerSpatial, KingOracle, and Oracle allow you to apply and destroy a schema.

C#

The code samples show how to describe, apply and destroy a schema.

The following namespaces are used:

- OSGeo.FDO.Commands.Schema
- OSGeo.FDO.Commands
- OSGeo.FDO.Schema

```
// describe a schema
IDescribeSchema descSchema = conn.CreateCommand(CommandType.CommandType_DescribeSchema) as IDescribeSchema;
FeatureSchemaCollection schemas = descSchema.Execute();
FeatureSchema schema = schemas[0];
ClassCollection classes = schema.Classes;
Class featureClass = classes[0];
PropertyDefinitionCollection properties = featureClass.Properties;
PropertyDefinition property = properties[0];
PropertyType propertyType = property.PropertyType;
if (propertyType == PropertyType.PropertyType_DataProperty) {
    DataPropertyDefinition dataPropDef = property as DataPropertyDefinition;
    DataType dataType = dataPropDef.DataType;
}
```

```

// create and apply a schema
FeatureSchema schema = new FeatureSchema("AnApplicationSchema",
"This schema contains one feature class.");
FeatureClass featClass = new FeatureClass("AFeatureClass", "This
feature class contains one identity property one data property and
a feature geometry.");
PropertyDefinitionCollection properties = featClass.Properties;
DataPropertyDefinitionCollection idProperties = featClass.Identi
tyProperties;
DataPropertyDefinition idProp = new DataPropertyDefinition("ID",
"This is the identity property");
idProp.DataType = DataType.DataType_Int64;
idProp.IsAutoGenerated = true;
idProp.Nullable = false;
idProp.ReadOnly = true;
properties.Add(idProp);
idProperties.Add(idProp);
DataPropertyDefinition int32Prop = new DataPropertyDefini
tion("Int32Prop", "This is an Int32 property");
int32Prop.DataType = DataType.DataType_Int32;
properties.Add(int32Prop);
// add the feature geometry
GeometricPropertyDefinition featGeomProp = new GeometricProperty
Definition("FeatGeomProp", "This is the feature geometry.");
// associate this geometric property with a spatial context
// you must have already added the named context to the data store
// if this property is not set, it is assigned the default spatial
context
// see the Spatial Context topic to read a description of the de
fault spatial context
featGeomProp.SpatialContextAssociation = "XY-M Spatial Context";
// by default this geometric property can contain geometries that
may be classified as point, curve, surface or solid
// also by default this geometric property cannot contain geomet
ries that have a Z ordinate or a measure attribute
properties.Add(featGeomProp);
// without the following line you would be adding a non-feature
geometry
featClass.GeometryProperty = featGeomProp;
ClassCollection classes = schema.Classes;
classes.Add(featClass);
// apply the schema

```

```

IApplySchema applySchema = conn.CreateCommand(CommandType.Command
Type_ApplySchema) as IApplySchema;
applySchema.FeatureSchema = schema;
applySchema.Execute();

// destroy the schema
IDestroySchema destroySchema = conn.CreateCommand(CommandType.Com
mandType_DestroySchema) as IDestroySchema;
destroySchema.SchemaName = "AnApplicationSchema";
destroySchema.Execute();

```

Insert Data

All providers except Gdal, Raster, WFS, and WMS allow you to insert data.

C#

The code sample shows how to insert a feature consisting of a data value and a geometry value.

The following namespaces are used:

- OSGeo.FDO.Commands
- OSGeo.FDO.Commands.Feature
- OSGeo.FDO.Expression
- OSGeo.FDO.Geometry

```

Identifier className = new Identifier("FeatureClass");
IInsert insert = conn.CreateCommand(CommandType.CommandType_Insert)
    as IInsert;
insert.FeatureClassName = className;
    PropertyValueCollection values = insert.PropertyValues;
// add the Int32 value to the insert command
int32Value = new Int32Value(5);
int32PropVal = new PropertyValue("Int32Prop", int32Value);
values.Add(int32PropVal);
// add the feature geometry to the insert command
FgfGeometryFactory geomFactory = new FgfGeometryFactory();
DirectPositionImpl position1 = new DirectPositionImpl(1.0, 1.0);
IPoint point = geomFactory.CreatePoint(position1);
GeometryValue geomVal = new GeometryValue(geomFactory.Get
Fgf(point));
PropertyValue geomPropVal = new PropertyValue("FeatGeomProp",
geomVal);
values.Add(geomPropVal);
// insert the feature
IFeatureReader reader = insert.Execute();
reader.Close();

```

Select Data

All providers allow you to select data.

C#

In the following code sample the filter is set to retrieve values for each feature in the database. Without adding a computed identifier to the select command's property collection, the select command would return all of the property values for each feature in the database. In this example, the computed identifier contains a function expression that converts the `DateTime` value in the `DateTimeProp` property to a string and returns the string. So the select returns a set of `DateTime` strings.

The following namespaces are used:

- `OSGeo.FDO.Commands`
- `OSGeo.FDO.Commands.Feature`
- `OSGeo.FDO.Expression`
- `OSGeo.FDO.Filter`

```

Identifier className = new Identifier("FeatureClass");
ISelect select = conn.CreateCommand(CommandType.CommandType_Select)
    as ISelect;
select.FeatureClassName = className;
Filter filter = Filter.Parse("ID >= 0");
select.Filter = filter;
ComputedIdentifier computedId = Expression.Parse("(ToString(Date
TimeProp) as computedId)") as ComputedIdentifier;
select.PropertyNames.Add(computedId);
IFeatureReader reader = select.Execute();
reader.Close();

```

Select Aggregate Data

All providers except the PostGIS provider, allow you to select aggregate data.

C#

In the following code sample the filter is set to retrieve values for each feature in the database. The computed identifier contains a function expression that takes the average of all of the values in the property whose name is Int32Prop and returns a double.

The following namespaces are used:

- OSGeo.FDO.Commands
- OSGeo.FDO.Commands.Feature
- OSGeo.FDO.Expression
- OSGeo.FDO.Filter

```

Identifier className = new Identifier("FeatureClass");
ISelectAggregates selectAggregates = conn.CreateCommand(Command
Type.CommandType_SelectAggregates) as ISelectAggregates;
selectAggregates.FeatureClassName = className;
Filter filter = Filter.Parse("ID >= 0");
selectAggregates.Filter = filter;
ComputedIdentifier computedId = Expression.Parse("(Avg(Int32Prop)
as computedId)") as ComputedIdentifier;
selectAggregates.PropertyNames.Add(computedId);
IDataReader reader = selectAggregates.Execute();
reader.Close();

```

Delete Data

All providers except Gdal, Raster, WFS, and WMS allow you to delete data.

C#

The following namespaces are used:

- OSGeo.FDO.Commands
- OSGeo.FDO.Commands.Feature
- OSGeo.FDO.Expression
- OSGeo.FDO.Filter

```
// delete all of the data
// ID is the name of the identity property
Filter filter = Filter.Parse("ID >= 0");
Identifier className = new Identifier("FeatureClass");
IDelete delete = conn.CreateCommand(CommandType.CommandType_Delete)
    as IDelete;
delete.FeatureClassName = className;
delete.Filter = filter;
int numDeleted = delete.Execute();
```

Schema Overrides

When FDO connects to a datastore with an existing schema and describes the schema, it maps the native data types to FDO data types to create a default physical to logical schema mapping. You can create a schema mapping that overrides the default FDO mapping. This schema override must be applied every time that you connect to the data store. Use the provider's Schema Override API to write a program that defines the overrides and writes them to an XML configuration file. Thereafter, before opening the connection to the provider, set the connection objects's configuration property to point to the XML configuration file.

The key elements to a schema override are an FDO feature schema and a mapping of column names in the existing table to property names in the FDO feature schema. The ODBC provider supports the mapping of two column names in an existing table to a 2D FDO point geometry property and optionally supports the mapping of three columns to a 3D point geometry property. In

the event that the mapping includes an FDO geometry property, a spatial context definition should be provided.

The description given here is based on sample code written to define a schema override for a table in a Microsoft Access database. The following table shows the name of the column in the Access database, which is the same as the default FDO property name, the FDO property name assigned by the schema override, a description of the contents of the column, the FDO data type assigned by the default schema mapping, and the FDO data type assigned by the schema override.

The default schema mapping of the CITY_ID and POPULATION columns are overridden to assign them different data types than the default ones and to give them different names. A change in case counts as a change in name. The default schema mapping of the LATITUDE and LONGITUDE columns are overridden because they are combined into a single property that has a different name and a different type than the defaults. The default schema mapping of the NAME, COUNTRY, and CAPITAL columns are overridden to assign them different names than the defaults. The default schema mapping of the URL column is not overridden and retains its default type and name.

| Column Name (Default Property Name) | Override Property Name | Column Contents | Default FDO Data Type | Override FDO Data Type |
|--|------------------------|---|-----------------------|------------------------|
| CITY_ID | Id | consecutive sequence of integers from 1 to < 1000 | Int16 | Int64 |
| NAME | Name | Names | String | String |
| COUNTRY | Country | Names | String | String |
| CAPITAL | Capital | 'N' or 'Y' | String | String |
| URL | Not over-riden | null | String | Not over-riden |
| POPULATION | Population | positive integers < 17,000,000 | Double | Int32 |
| LATITUDE | Geometry | positive and negative decimals | Double | Geometry |
| LONGITUDE | Geometry | positive and negative decimals | Double | Geometry |

First a conceptual view of how to create the XML configuration file for a schema override is presented and then a code view is presented. The override includes a geometry property and so a spatial context is defined as well.

- 1 Open the XML file
- 2 Create a spatial context for the geometry and write it to the XML file
- 3 Create the FDO feature schema and write it to the XML file
- 4 Create a connection to the provider and get a reference to the connection's physical schema mapping.
- 5 Use the reference to the connection's physical schema mapping and the Schema Override API to create the schema override whereby the logical fields in the feature schema are associated with columns in the database table
- 6 Write the schema override to the XML file
- 7 Close the XML file.

C#

The following namespaces are used.

- OSGeo.FDO.Common..Io
- OSGeo.FDO.Common.Xml
- OSGeoFDO.Geometry
- OSGeo.FDO.Schema

Create the XML Schema Override Configuration File

Open the XML File

```
IOFileStream writeFileStream = new IOFileStream("Cities.xml", "w");  
XmlWriter writer = new XmlWriter(writeFileStream);
```

Create a Spatial Context and Write It to the XML File

```
XmlSpatialContextFlags flags = new XmlSpatialContextFlags();
XmlSpatialContextWriter spatialContextWriter = new XmlSpatialContextWriter(writer, flags);
DirectPositionImpl lowerLeft = new DirectPositionImpl();
lowerLeft.X = -180.0;
lowerLeft.Y = -90.0;
DirectPositionImpl upperRight = new DirectPositionImpl();
upperRight.X = 180.0;
upperRight.Y = 90.0;
FgfGeometryFactory geomFactory = new FgfGeometryFactory();
IEnvelope envelope = geomFactory.CreateEnvelope(lowerLeft, upperRight);
IGeometry geom = geomFactory.CreateGeometry(envelope);
byte[] extent = geomFactory.GetFgf(geom);
spatialContextWriter.CoordinateSystem = "WGS 84";
spatialContextWriter.CoordinateSystemWkt = "GEOGCS [ \"Longitude / Latitude (WGS 84)\", DATUM [\"WGS 84\", SPHEROID [\"WGS 84\", 6378137, 298.257223563]], PRIMEM [ \"Greenwich\", 0.000000 ], UNIT \"Decimal Degree\", 0.01745329251994330]]";
spatialContextWriter.Description = "This Coordinate System is used for GPS.";
spatialContextWriter.ExtentType = SpatialContextExtentType.SpatialContextExtentType_Static;
spatialContextWriter.Extent = extent;
spatialContextWriter.Name = "WGS84";
spatialContextWriter.XYTolerance = 0.001;
// the next line writes the XML declaration, the opening tag for the fdo:DataStore element
// and the gml:DerivedCRS element to the XML file
spatialContextWriter.WriteSpatialContext();
```

Create the Feature Schema and Write It to the XML File

```

FeatureSchema schema = new FeatureSchema("World", "Logical feature
schema");
FeatureClass featClass = new FeatureClass("Cities", "This feature
class contains one identity property, many data properties, and
a feature geometry.");
PropertyDefinitionCollection properties = featClass.Properties;
DataPropertyDefinitionCollection idProperties = featClass.Identi
tyProperties;
DataPropertyDefinition idProp = new DataPropertyDefinition("Id",
"This is the unique id number for the city.");
idProp.DataType = DataType.DataType_Int64;
idProp.IsAutoGenerated = false;
idProp.Nullable = false;
idProp.ReadOnly = true;
properties.Add(idProp);
idProperties.Add(idProp);
DataPropertyDefinition nameProp = new DataPropertyDefini
tion("Name", "This is the name of the City.");
nameProp.DataType = DataType.DataType_String;
nameProp.Length = 64;
properties.Add(nameProp);
DataPropertyDefinition countryProp = new DataPropertyDefini
tion("Country", "This is country that contains the city.");
countryProp.DataType = DataType.DataType_String;
countryProp.Length = 64;
properties.Add(countryProp);
DataPropertyDefinition populationProp = new DataPropertyDefini
tion("Population", "This is the population of the city.");
populationProp.DataType = DataType.DataType_Int32;
properties.Add(populationProp);
DataPropertyDefinition capitalProp = new DataPropertyDefini
tion("Capital", "This is 'Y' or 'N' to say whether the city is the
capital of the country.");
capitalProp.DataType = DataType.DataType_String;
capitalProp.Length = 1;
properties.Add(capitalProp);
GeometricPropertyDefinition featGeomProp = new GeometricProperty
Definition("Geometry", "This is the feature geometry.");
featGeomProp.GeometryTypes = (int)GeometricType.Geometric
Type_Point;
featGeomProp.HasElevation = false;
featGeomProp.HasMeasure = false;
properties.Add(featGeomProp);

```

```
featClass.GeometryProperty = featGeomProp;  
ClassCollection classes = schema.Classes;  
classes.Add(featClass);  
// the next line writes the xs:schema element to the XML file  
schema.WriteXml(writer);
```

Create the Physical Schema Mapping and Write It to the XML File

```

// set the connection to point
// at the Microsoft Access database file
//whose Data Source Name (DSN) has been
// defined using the Data Sources (ODBC)
// Windows XP administrative tool
connection.ConnectionString = "DataSourceName=" + dataSourceName;
// get a reference to the PhysicalSchemaMapping object contained
in the connection
PhysicalSchemaMapping connectionSchemaMapping = connection.CreateS
chemaMapping();
// use that reference to create an override physical schema mapping
// the override definition associates columns in the columns of
the Access database with property names in the FDO feature schema
OvPhysicalSchemaMapping mapping = new OvPhysicalSchemaMapping(con
nectionSchemaMapping, false);
mapping.Name = "World";
OvClassCollection overrideClasses = mapping.Classes;
OvClassDefinition overrideClassDef = new OvClassDefinition("Cit
ies");
OvPropertyDefinitionCollection overrideProperties = override
ClassDef.Properties;
// associate the table in the Access database with the override
class definition
OvTable table = new OvTable("Cities");
overrideClassDef.Table = table;
// associate the columns in the table in the Access database with
the properties in the FDO feature schema
OvColumn column = null;
OvGeometricPropertyDefinition overridGeomProp = new OvGeometric
PropertyDefinition("Geometry");
overridGeomProp.GeometricColumnType = OSGeo.FDO.Providers.Rd
bms.Override.OvGeometricColumnType.OvGeometricColumnType_Double;
overridGeomProp.GeometricContentType = OSGeo.FDO.Providers.Rd
bms.Override.OvGeometricContentType.OvGeometricContentType_Ordin
ates;
overridGeomProp.XColumnName = "LONGITUDE";
overridGeomProp.YColumnName = "LATITUDE";
overrideProperties.Add(overridGeomProp);
OvDataPropertyDefinition overrideIdProp = new OvDataPropertyDefin
ition("Id");
column = new OvColumn("CITY_ID"); overrideIdProp.Column = column;
overrideProperties.Add(overrideIdProp);

```



```

OvDataPropertyDefinition overrideNameProp = new OvDataPropertyDefinition("Name");
column = new OvColumn("NAME");
overrideNameProp.Column = column;
overrideProperties.Add(overrideNameProp);
OvDataPropertyDefinition overrideCountryProp = new OvDataPropertyDefinition("Country");
column = new OvColumn("COUNTRY");
overrideCountryProp.Column = column;
overrideProperties.Add(overrideCountryProp);
OvDataPropertyDefinition overridePopulationProp = new OvDataPropertyDefinition("Population");
column = new OvColumn("POPULATION");
overridePopulationProp.Column = column;
overrideProperties.Add(overridePopulationProp);
OvDataPropertyDefinition overrideCapitalProp = new OvDataPropertyDefinition("Capital");
column = new OvColumn("CAPITAL");
overrideCapitalProp.Column = column;
overrideProperties.Add(overrideCapitalProp);
    overrideClasses.Add(overrideClassDef);
// write the override definition to the XML file
XmlFlags flags = new XmlFlags();
// the next line writes the SchemaMapping element to the XML file
mapping.WriteXml(writer, flags);
mapping.Dispose();

```

Write the Closing Tag and Close the XML File

```

// the next line writes the closing tag of the fdo:DataStore element to the XML file
writer.WriteEndElement();
writer.Close();
writer.Dispose();
writeFileStream.Close();
writeFileStream.Dispose();

```

Configure a Connection to Use the Schema Override

The connection object contains a property whose value points to the XML configuration file.

```
IConnection connection = connMgr.CreateConnection(providerName);
IOFileStream configurationFileStream = new IOFileStream("Cit
ies.xml", "r");
connection.Configuration = configurationFileStream;
connection.ConnectionString = "DataSourceName=" + dataSourceName;
connection.Open();
```

Xml Serialize/Deserialize

This code sample serializes the spatial context and feature schema of a data store to an xml file and then deserializes it in another data store.

C#

This is the C# code.

Namespaces

The following namespaces are used:

- OSGeo.FDO.Commands.Schema
- OSGeo.FDO.Common.IO
- OSGeo.FDO.Common.Xml
- OSGeo.FDO.Schema
- OSGeo.FDO.Xml

Serialize

```
// open the xml file
XmlWriter xmlWriter = new XmlWriter(xmlFilename);
// serial the spatial contexts
XmlSpatialContextWriter spatialContextWriter = new XmlSpatialContextWriter(xmlWriter);
XmlSpatialContextSerializer.XmlSerialize(connection, spatialContextWriter);
// serialize the feature schemas
IDescribeSchema descSchema = connection.CreateCommand(CommandType.CommandType_DescribeSchema) as IDescribeSchema;
FeatureSchemaCollection schemas = descSchema.Execute();
schemas.WriteXml(xmlWriter);
// clean up
spatialContextWriter.Dispose();
xmlWriter.Dispose();
```

Deserialize

```
// deserialize the spatial contexts
IOFileStream fileStream = new IOFileStream(xmlFilename, "r");
XmlReader xmlReader = new XmlReader(fileStream);
XmlSpatialContextReader xScReader = new XmlSpatialContextReader(xmlReader);
XmlSpatialContextSerializer.XmlDeserialize(connection, xScReader);
fileStream.Reset();
// deserialize the schema
FeatureSchemaCollection schemas = new FeatureSchemaCollection(null);
schemas.ReadXml(fileStream);
IApplySchema applySchema = connection.CreateCommand(CommandType.CommandType_ApplySchema) as IApplySchema;
FeatureSchema schema = null;
for (int i = 0; i < schemas.Count; i++)
{
    schema = schemas.get_Item(i);
    applySchema.FeatureSchema = schema;
    applySchema.Execute();
}
fileStream.Reset();
// clean up
xmlReader.Dispose();
fileStream.Close();
```

Geometry

The geometry code samples are divided into two parts. The first is constructing geometries. The second is deconstructing them, that is, extracting information from the geometry objects.

Construction

Geometries can be constructed using a set of scaffolding classes, which includes a geometry factory class, or they can be constructed using the geometry factory class from text specifications. The syntax for the text specifications is described in the Geometry Value section in the Filter and Expression Languages chapter. What follows here is a description of geometry construction using the scaffolding classes.

Some geometries are constructed using a two step procedure. First, create coordinates and then create the geometry. Some geometries require an intermediate step. Use the coordinates to create subcomponents and then use the subcomponents to create the geometry. The construction of some geometries requires the use of helper collection classes. Some of these classes are collections of geometries and some are collections of geometry subcomponents.

Simple geometries like point and line are constructed directly from coordinates. Complex geometries like curved lines and polygons are constructed from geometry subcomponents like arc and line segments and rings. In general subcomponents are constructed from coordinates. The exception is the ring, which is constructed from arc and line segments. The glue is the helper collection classes for coordinates, segments, and rings.

A geometry object may consist of homogeneous or heterogeneous aggregations of points, lines, and polygons. The glue is helper collection classes for points, lines, and polygons.

C# Namespaces

The following namespaces are used:

- `OSGeo.FDO.Common`
- `OSGeo.FDO.Geometry`

C# Geometry Scaffolding Classes

The scaffolding classes used to construct the simple, complex, and aggregate geometries are the following:

- the `IDirectionPosition` class used to construct positions (coordinates),
- the `FgfGeometryFactory` class used to construct the geometry subcomponents and geometries,
- the geometry subcomponent classes: `ILineStringSegment`, `ICircularArcSegment`, `ILinearRing`, and `IRing`,
- the helper collection classes: `PointCollection`, `LineStringCollection`, `CurveSegmentCollection`, `LinearRingCollection`, `RingCollection`, `PolygonCollection`, `CurvePolygonCollection`, and `GeometryCollection`.

IDirectPositionImpl

The `IDirectPosition` is the basic building block of geometries. Use its constructor to define a coordinate. Use instances of it to define the `IPoint` and `ILineString` geometries or to define the geometry subcomponents `ILinearRing`, `ICircularArcSegment`, and `ILineStringSegment`.

```
DirectPositionImpl pos00 = new DirectPositionImpl(0.0, 0.0);
```

FgfGeometryFactory

Use an instance of this class to do the following:

- create geometry subcomponents used to construct `IGeometry`-based objects
- create `IGeometry`-based objects
- convert `IGeometry`-based objects into binary-formatted geometries used for data store inserts
- convert binary-formatted geometries retrieved during data store queries into `IGeometry`-based objects

```
FgfGeometryFactory geomFactory = new FgfGeometryFactory();
```

Geometry Subcomponents

The following subcomponents are used in the construction of geometries and geometry subcomponents:

- The `ICircularArcSegment` and `ILineStringSegment` subcomponents are used to construct the `ICurveString` geometry and the `IRing` geometry subcomponent.
- The `ILinearRing` subcomponent is used to construct the `IPolygon` geometry.
- The `IRing` subcomponent is used to construct the `ICurvePolygon` geometry.

ICircularArcSegment

The creation of an instance uses `IDirectPositionImpl` objects, which respectively define a start position, a mid point, and an end position for the arc.

```
ICircularArcSegment as012122 = geomFactory.CreateCircularArcSegment(pos01, pos21, pos22);
```

ILineStringSegment

The creation of an instance uses a `DirectPositionCollection`, which contains `IDirectionPositionImpl` objects.

```
positions.Add(pos23);
positions.Add(pos30);
positions.Add(pos33);
ILineStringSegment ls233033 = geomFactory.CreateLineStringSegment(positions);
positions.Clear();
```

ILinearRing

The creation of an instance uses a `DirectPositionCollection`, which contains `IDirectionPositionImpl` objects.

```
positions.Add(pos01);
positions.Add(pos21);
positions.Add(pos12);
positions.Add(pos01);
ILinearRing lr01211201 = geomFactory.CreateLinearRing(positions);
positions.Clear();
```

IRing

An IRing is constructed using a CurveSegmentCollection. This collection can contain ICircularArcSegment and ILineStringSegment objects.

```
curveSegments.Add(as1340n20);
curveSegments.Add(lsn2013);
IRing rAs1340n20Lsn2013 = geomFactory.CreateRing(curveSegments);
curveSegments.Clear();
```

Collection Classes

The following collection classes are used during the construction of the various geometries:

```
// Contains IDirectPositionImpl objects
DirectPositionCollection positions = new DirectPositionCollection();
// Contains IPoint objects
PointCollection points = new PointCollection();
// Contains ILineString objects
LineStringCollection lines = new LineStringCollection();
// Contains ICircularArcSegment and ILineStringSegment objects
CurveSegmentCollection curveSegments = new CurveSegmentCollection();
// Contains ICurveString objects
CurveStringCollection curves = new CurveStringCollection();
// Contains ILinearRing objects
LinearRingCollection linearRings = new LinearRingCollection();
// Contains IRing objects
RingCollection curveRings = new RingCollection();
// Contains IPolygon objects
PolygonCollection linearPolygons = new PolygonCollection();
// Contains ICurvePolygon objects
CurvePolygonCollection curvePolygons = new CurvePolygonCollection();
GeometryCollection geometries = new GeometryCollection();
```

C# Geometries Constructed using IDirectPositionImpl

These geometries are IPoint and ILineString.

IPoint

The creation of an instances uses an IDirectPositionImpl object.

```
IPoint pt00 = geomFactory.CreatePoint(pos00);
```

ILineString

The creation of an instances uses a DirectPositionCollection object, which is composed of IDirectionPositionImpl object.

```
positions.Add(pos01);
positions.Add(pos21);
ILineString line0121 = geomFactory.CreateLineString(positions);
positions.Clear();
```

C# Geometries Constructed Using Geometry Subcomponents

These geometries are ICurveString, IPolygon, and ICurvePolygon.

ICurveString

The creation of an instance uses a CurveSegmentCollection object, which contains an ICircularArcSegment object and an ILineStringSegment object.

```
// ICurveString composed of a circular arc segment and a line
string segment
curveSegments.Add(as022223);
curveSegments.Add(ls233033);
ICurveString csAs022223Ls233033 = geomFactory.CreateCurveString(curveSegments);
curveSegments.Clear();
```

IPolygon

The creation of an instance uses an ILinearRing object for the exterior ring and a LinearRingCollection object, composed of ILinearRing objects, for the interior rings.

```
// IPolygon with one exterior ring and no interior rings
IPolygon polyEr01211201 = geomFactory.CreatePolygon(lr01211201,
null);
// IPolygon with one exterior ring and one interior ring
linearRings.Add(lr01211201);
polyEr13n204013Ir01211201 = geomFactory.CreatePolygon(lr13n204013,
linearRings);
linearRings.Clear();
```


ICurvePolygon

The creation of an instance uses an `IRing` object for the exterior ring and a `RingCollection` object, composed of an `IRing` object, for the interior rings.

```
// ICurvePolygon with one exterior ring, rAs1340n20Lsn2013, which
// is composed of an arc segment and a line segment
// and one interior ring, rLs0121Ls2112Ls1201, which is a triangle
// made of three line segments
curveRings.Add(rLs0121Ls2112Ls1201);
ICurve Polygon curvPolyErAs1340n20Lsn2013IrLs0121Ls2112Ls1201 =
geomFactory.CreateCurvePolygon(rAs1340n20Lsn2013, curveRings);
curveRings.Clear();
```

C# Aggregate Geometries

The aggregate geometries are `IMultiPoint`, `IMultiLineString`, `IMultiCurveString`, `IMultiPolygon`, `IMultiCurvePolygon`, and `IMultiGeometry`.

IMultiPoint

The creation of an instance uses a `PointCollection` object.

```
points.Add(pt00);
points.Add(pt11);
IMultiPoint pt00pt11 = geomFactory.CreateMultiPoint(points);
points.Clear();
```

IMultiLineString

The creation of an instance uses a `LineStringCollection` object.

```
lines.Add(line0121);
lines.Add(line0222);
IMultiLineString line0121line0222 = geomFactory.CreateMultiLineS
tring(lines);
lines.Clear();
```

IMultiCurveString

The creation of an instance uses a `CurveStringCollection` object.

```

curves.Add(csAs012122);
curves.Add(csAs022223Ls233033);
IMultiCurveString aMultiCurvePolygon = geomFactory.CreateMultiCurveString(curves);
curves.Clear();

```

IMultiPolygon

The creation of an instance uses a PolygonCollection object.

```

linearPolygons.Add(polygonNum1);
linearPolygons.Add(polygonNum2);
IMultiPolygon aMultiPolygon = geomFactory.CreateMultiPolygon(linearPolygons);
linearPolygons.Clear();

```

IMultiCurvePolygon

The creation of an instance uses a CurvePolygonCollection object.

```

curvePolygons.Add(curvePolygonNum1);
curvePolygons.Add(curvePolygonNum2);
IMultiCurvePolygon aMultiCurvePolygon = geomFactory.CreateMultiCurvePolygon(curvePolygons);
curvePolygons.Clear();

```

IMultiGeometry

The creation of an instance uses a GeometryCollection object.

```

geometries.Add(aPoint);
geometries.Add(aMultiPoint);
geometries.Add(aLineString);
geometries.Add(aMultiLineString);
geometries.Add(aCurveString);
geometries.Add(aMultiCurveString);
geometries.Add(aPolygon);
geometries.Add(aMultiPolygon);
geometries.Add(aCurvePolygon);
geometries.Add(aMultiCurvePolygon);
IMultiGeometry aMultiGeometry = geomFactory.CreateMultiGeometry(geometries);
geometries.Clear();

```

C# Geometries from Text Specifications

You can use the geometry factory's `CreateGeometry` method to create geometries from text specifications. Here's an `IPoint` example.

```
IPoint pt11 = geomFactory.CreatePoint(pos11);  
IPoint aSecondPt11 = geomFactory.CreateGeometry(pt11.Text) as  
IPoint;  
aThirdPt11 = geomFactory.CreateGeometry("POINT (1 1)") as IPoint;
```

Deconstruction

Deconstruction means getting information such as dimensionality, envelope, text specification, geometry, geometry subcomponents, and positions (coordinates). The following table specifies the information available from the classes.

Position means coordinate values. Dimensionality values are XY, XYZ, XYM, or XYZM. Envelope means bounding rectangle. The envelope for an aggregate geometry contains all of the included geometries. Text means FGF well-known text specification. `IsClosed` is a boolean specifying whether or not the start and end coordinates are identical.

| Class | Information available |
|----------------------------------|--|
| <code>IDirectPosition</code> | position, dimensionality |
| <code>IPoint</code> | position, dimensionality, envelope, text |
| <code>IMultiPoint</code> | dimensionality, envelope, text, <code>IPoint</code> |
| <code>ILineString</code> | position, dimensionality, envelope, text |
| <code>IMultiLineString</code> | dimensionality, envelope, text, <code>ILineString</code> |
| <code>ILineStringSegment</code> | <code>IsClosed</code> , position, dimensionality, envelope |
| <code>ICircularArcSegment</code> | <code>IsClosed</code> , position, dimensionality, envelope |
| <code>ICurveString</code> | position, dimensionality, envelope, text, <code>ICircularArcSegment</code> , <code>ILineStringSegment</code> |
| <code>IMultiCurveString</code> | dimensionality, envelope, text, <code>ICurveString</code> |

| Class | Information available |
|--------------------|---|
| ILinearRing | position, dimensionality, envelope |
| IPolygon | dimensionality, envelope, text, ILinearRing |
| IMultiPolygon | dimensionality, envelope, text, IPolygon |
| IRing | position, dimensionality, ICircularArcSegment, ILineStringSegment |
| ICurvePolygon | dimensionality, envelope, text, IRing |
| IMultiCurvePolygon | dimensionality, envelope, text, ICurvePolygon |
| IMultiGeometry | dimensionality, envelope, text, IPoint, IMultiPoint, ILineString, IMultiLineString, ICurveString, IMultiCurveString, IPolygon, IMultiPolygon, ICurvePolygon, IMultiCurvePolygon |

Coordinates

```
// IDirectPosition aPosition
double X = aPosition.X;
double Y = aPosition.Y;
double Z;
double M;
int dim = aPosition.Dimensionality;
if ((dim & 1) == 1)
    Z = aPosition.Z;
if ((dim & 2) == 2)
    M = aPosition.M;
```

IDirectPosition (IPoint, ILineString, ICircularArcSegment, ILineStringSegment, ILinearRing)

```
// IPoint aPoint
IDirectPosition position = aPoint.Position;

// ILineString line
IDirectPosition aPosition = null;
IDirectPosition startPosition = line.StartPosition;
IDirectPosition endPosition = line.EndPosition;
for (int i = 0; i < line.Count; i++)
    aPosition = line.get_Item(i);
```

```

// ICircularArcSegment arcSegment
IDirectPosition startPosition = arcSegment.StartPosition;
IDirectPosition midPoint = arcSegment.MidPoint;
IDirectPosition endPosition = arcSegment.EndPosition;

// ILineStringSegment lineSegment
IDirectPosition startPosition = lineSegment.StartPosition;
IDirectPosition endPosition = lineSegment.EndPosition;
DirectPositionCollection positions = lineSegment.Positions;
foreach(IDirectPosition position in positions)

// ICurveString curve
IDirectPosition startPosition = curve.StartPosition;
IDirectPosition endPosition = curve.EndPosition;

// ILinearRing linearRing
IDirectPosition position = null;
for (int i = 0; i < ring.Count; i++)
    position = ring.get_Item(i);

// ILinearRing ring
IDirectPosition position = null;
for (int i = 0; i < ring.Count; i++)
    position = ring.get_Item(i);

```

Dimensionality, Envelope, and Text

```

int dimensionality = aPoint.Dimensionality;
IEnvelope envelope = aPoint.Envelope;
string text = aPoint.Text;

```

IsClosed (ICircularArcSegment and ILineStringSegment)

```

// ICircularArcSegment arcSegment
bool isClosed = arcSegment.IsClosed;

// ILineStringSegment lineSegment
bool isClosed = lineSegment.IsClosed;

```

IPoint and IMultiPoint

```

// IMultiPoint aMultiPoint
IPoint aPoint = null;
for (int i = 0; i < aMultiPoint.Count; i++)
    aPoint = aMultiPoint.get_Item(i);

```

ILineString and IMultiLineString

```
// IMultiLineString aMultiLineString
ILineString line = null;
for (int i = 0; i < aMultiLineString.Count; i++)
    line = aMultiLineString.get_Item(i);
```

ICurveString and IMultiCurveString

```
// IMultiCurveString aMultiCurveString
ICurveString curve = null;
for (int i = 0; i < aMultiCurveString.Count; i++)
    curve = aMultiCurveString.get_Item(i);
```

ICircularArcSegment and ILineStringSegment (ICurveString and IRing)

```
// ICurveString aCurveString
ICurveSegmentAbstract curveSegment = null;
ICircularArcSegment arcSegment = null;
ILineStringSegment lineSegment = null;
for (int i = 0; i < aCurveString.Count; i++)
{
    curveSegment = aCurveString.get_Item(i);
    if (curveSegment.DerivedType == GeometryComponentType.Geometry
        ComponentType_CircularArcSegment)
        arcSegment = curveSegment as ICircularArcSegment;
    else if (curveSegment.DerivedType == GeometryComponentType.Geometry
        ComponentType_LineStringSegment)
        lineSegment = curveSegment as ILineStringSegment;
}

// IRing ring
ICurveSegmentAbstract curveSegment = null;
ICircularArcSegment arcSegment = null;
ILineStringSegment lineSegment = null;
for (int i = 0; i < ring.Count; i++)
{
    curveSegment = ring.get_Item(i);
    if (curveSegment.DerivedType == GeometryComponentType.Geometry
        ComponentType_CircularArcSegment)
        arcSegment = curveSegment as ICircularArcSegment;
    else if (curveSegment.DerivedType == GeometryComponentType.Geometry
        ComponentType_LineStringSegment)
        lineSegment = curveSegment as ILineStringSegment;
}
```

ILinearRing (IPolygon)

```
// IPolygon aPolygon
ILinearRing anExteriorRing = aPolygon.ExteriorRing;
ILinearRing anInteriorRing = null;
for (int i = 0; i < aPolygon.InteriorRingCount; i++)
    anInteriorRing = aPolygon.GetInteriorRing(i);
```

IPolygon and IMultiPolygon

```
// IMultiPolygon aMultiPolygon
IPolygon aPolygon = null;
for (int i = 0; i < aMultiPolygon.Count; i++)
    aPolygon = aMultiPolygon.get_Item(i);
```

IRing (ICurvePolygon)

```
// ICurvePolygon aCurvePolygon
IRing anExteriorRing = aCurvePolygon.ExteriorRing;
IRing anInteriorRing = null;
for (int i = 0; i < count; i++)
    anInteriorRing = aCurvePolygon.get_InteriorRing(i);
```

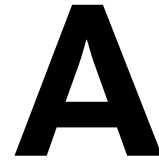
ICurvePolygon and IMultiCurvePolygon

```
// IMultiCurvePolygon aMultiCurvePolygon
ICurvePolygon aCurvePolygon = null;
for (int i = 0; i < aMultiCurvePolygon.Count; i++)
    aCurvePolygon = aMultiCurvePolygon.get_Item(i);
```

IMultiGeometry

```
// IMultiGeometry aMultiGeometry
IGeometry geom = null;
GeometryType geomType;
IPoint pt = null;
IMultiPoint mpt = null;
ILineString line = null;
IMultiLineString mline = null;
ICurveString curve = null;
IMultiCurveString mcurve = null;
IPolygon poly = null;
IMultiPolygon mpoly = null;
ICurvePolygon curvPoly = null;
IMultiCurvePolygon mCurvPoly = null;
for (int i = 0; i < aMultiGeometry.Count; i++)
{
    geom = aMultiGeometry.get_Item(i);
    geomType = geom.DerivedType;
    switch (geomType)
    {
        case GeometryType.GeometryType_Point: pt = geom as IPoint; break;
        case GeometryType.GeometryType_MultiPoint: mpt = geom as IMultiPoint; break;
        case GeometryType.GeometryType_LineString: line = geom as ILineString; break;
        case GeometryType.GeometryType_MultiLineString: mline = geom as IMultiLineString; break;
        case GeometryType.GeometryType_CurveString: curve = geom as ICurveString; break;
        case GeometryType.GeometryType_MultiCurveString: mcurve = geom as IMultiCurveString; break;
        case GeometryType.GeometryType_Polygon: poly = geom as IPolygon; break;
        case GeometryType.GeometryType_MultiPolygon: mpoly = geom as IMultiPolygon; break;
        case GeometryType.GeometryType_CurvePolygon: curvPoly = geom as ICurvePolygon; break;
        case GeometryType.GeometryType_MultiCurvePolygon: mCurvPoly = geom as IMultiCurvePolygon; break;
        default: break;
    }
}
```


Autodesk FDO Provider for Oracle



This appendix discusses FDO API development issues that are related to using FDO Provider for Oracle.

What Is FDO Provider for Oracle?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. FDO Provider for Oracle provides FDO with access to an Oracle-based data store.

FDO Provider for Oracle API provides custom commands specifically designed to work with the FDO API. For example, using these commands, you can do the following:

- Gather information about a provider.
- Transmit client services exceptions.
- Get lists of accessible data stores.
- Create connection objects.
- Create and dropping spatial indexes.

FDO Provider for Oracle General Requirements

For Autodesk Map 3D users, a pre-requisite for creating schema and managing long transactions is to include the setting WM_ADMIN_ROLE in the user definition.

If a user definition does not have this setting, use the *FDO User Manager Tool* to delete the user definition and then recreate it to include WM_ADMIN_ROLE.

Create the f_user role

You must create the f_user_role in Oracle prior to using the User Management API to add a user. Do the following in a cmd.exe window:

- 1 Change directories to the folder containing the CreateRoles.sql script.
For Map 3D, this folder is <Map 3D root folder>\FDO\Bin\Com.
- 2 Login to sqlplus using the sys as sysdba account.
- 3 At the SQL prompt enter @CreateRoles.sql.

FDO Provider for Oracle Connection

This information supplements the Establishing a Connection chapter. You connect to a data store directly through FDO Provider for Oracle, and the underlying data source for the data store is an Oracle database.

You can connect to the data store in one step if you already know its name. Otherwise, you must connect in two steps.

The minimum required connection parameters for the initial call to Open() are service, username, and password.

The service parameter is the Oracle Net Service Name of an Oracle instance. An instance could be running on your machine or on some other machine in the network. You can use the Oracle Net Manager to identify which Oracle instances are available to you and what their Net Service Names are. In an Oracle 10g installation on a Microsoft Windows XP machine, Net Manager can be accessed with Start ► Programs ► Oracle ► Configuration and Migration Tools ► Net Manager. The connection information for the Net Service Name is contained in a file named *tnsnames.ora*, which is located in the Network/Admin folder in either the Oracle instance or the client installation directory.

Multiple users can access the data store. However, access is password-protected.

An Oracle data source, when accessed by FDO Provider for Oracle, may contain more than one data store. For the first call to `Open()`, a data store name is optional. If successful, the first call to `Open()` results in the data store parameter becoming a required parameter and a list of the names of the data stores in the data source becoming available. You must choose a data store and call `Open()` again.

If you know the name of the data store, you can provide it for the first call to `Open()` and make the connection in one step.

FDO Provider for Oracle and Foreign Schemas

FDO Provider for Oracle supports the creation of foreign schemas. A foreign schema is capable of mapping a table to Oracle instances. This allows users with a pre-existing application (for example, one created with Autodesk GIS Design Server) to map their application to FDO. As a result, both the FDO capability and conventional capability can be used by the same application.

Foreign Schema Settings

In order to use a foreign schema, certain privileges for FDO are required. To support the foreign schema capability, the following settings are required:

- FDO schema instance
- Foreign schema instance
- Oracle identity property

These settings are required for accessing the foreign schema objects (that is, tables, views, and sequences.).

Settings on the FDO Schema Instance

If the foreign schema is on a different Oracle instance, create a PUBLIC database link. A database link is a schema object that enables accessing of objects on another Oracle instance.

Settings on the Foreign Schema Instance

If the foreign schema is on a different Oracle instance, create an FDO user using the same Username and password as on the instance where the FDO schema exists.

NOTE This FDO user does not need to have been granted the `f_user_role` role.

Grant the select, update, delete, and insert privileges on tables (views, sequences) to the FDO user that is mapped to the FDO schema. Note that if the Foreign Schema tables are enabled for Oracle Workspace Manager, the privileges must be granted for the based table.

Oracle Identity Property

When specifying the main identifier for a feature class with FDO Provider for Oracle, the data type must be `Int64`. It must also have the following settings: `ReadOnly=True` and `Autogenerated=True`. If an identifier is not created with these properties, it will be created by FDO Provider for Oracle. Otherwise, an exception may be raised in certain conditions.

If the foreign schema uses a different data type for identifiers, the user must define the identifier as a `fdo int64` type with `ReadOnly=True` and `Autogenerated=True` in the XML configuration file. If the identifier in the foreign schema uses an Oracle sequence to generate the unique numbers, the `sequencename` override must be defined in the override XML file.

Read-Write Privileges

If FDO requires read-write privileges to work with a foreign schema, the owner of the foreign schema must grant these privileges. Also, access to the Oracle metaschema on the tables is required.

NOTE The owner must explicitly grant these privileges. These privileges will vary, according to the schema owner.

Foreign Schema Limitations

This section provides information about known limitations of foreign schemas.

Ensuring Valid Views When Applying a Feature Schema Against a Foreign Schema

The ApplySchema command can create invalid views when the feature schema is applied against a foreign schema. When you apply against a foreign schema, tables and columns are not automatically created if they do not already exist. A view is created, however, in the connected data store that references these foreign tables and columns. ApplySchema succeeds even if certain tables or columns cannot be obtained for various reasons, such as:

- The database link to the foreign tables is invalid.
- The Oracle instance containing the foreign tables is shut down or unreachable due to network problems.
- The foreign table or column simply does not exist.

When one of these situations occurs, ApplySchema creates invalid (dangling) views because these views reference tables or columns that cannot be reached.

Invalid views can occur regardless of whether schema overrides have been specified because the default schema mappings can also reference unreachable tables or columns. For example, if the feature schema being applied has a Pole class with no table name override, the Pole class is mapped to the POLE table in the foreign schema and a POLE view (referencing the POLE table) is created in the data store that the Oracle provider is currently connected to. If the POLE table does not exist, then the POLE view is dangling, or invalid.

Use one of the following procedures to correct invalid views, depending on whether the correct table name was specified (either through a schema override or the default class to table mapping rule):

Procedure When Table Name Is Correct

If the table name is correct, but it is not reachable for the reasons listed above:

- 1 Create the table or make it reachable by fixing the database link, fixing network problems, or starting the Oracle instance that contains the table.
- 2 Recompile the view that references the table.

The Oracle SQL statement for recompiling the view is:

```
Alter view <view_name> compile;
```

If there are a number of views to recompile, an alternative method is to use the following procedure for a wrong table name, but only do Steps 1 and 3.

Procedure When Wrong Table Name Is Specified

If the wrong table name is specified, to ensure valid views:

- 1 Destroy the feature schema. As long as the feature schema maps onto a foreign schema, destroying it does not result in loss of data. No tables or columns are dropped, only the referencing views created by FDO Provider for Oracle are dropped.
- 2 Fix the schema overrides to supply the proper table name. In some cases, you may need to add a schema override. For example, if a class named "Pole" corresponds to the foreign table "telco_pole", then a table name override must be specified for the Pole class, since the class and table names differ.
- 3 Re-apply the feature schema.

If your feature schema contains a mixture of classes mapped to foreign tables and classes mapped to non-foreign tables, then the procedure is slightly more complex, especially if any non-foreign table contains data. In this case, the following steps must be done programmatically throughout the FDO API:

- 1 Describe the feature schema using the DescribeSchema command. Retain this description.
- 2 Remove every class, except the one with the wrong table name, from the feature schema returned. However, do not delete the classes (that is, do not call FdoClassDefinition::Delete()).
- 3 Describe the feature schema again (ensure that you retain the feature schema from the first Describe).
- 4 Delete the class with the wrong table name from the feature schema returned by the second Describe (by calling its FdoClassDefinition::Delete() function).
- 5 Fix the schema overrides to supply the proper table name.
- 6 Ensure that FdoIApplySchema::SetIgnoreStates() is set to false, then Apply the feature schema described from Step 3. This deletes the class to repair.
- 7 Ensure that FdoIApplySchema::SetIgnoreStates() is set to true, then Apply the feature schema described from Step 1, along with the schema overrides. This re-creates the class to repair.

Overrides Capable of Causing Invalid Views

Any schema mapping between a feature schema element and a table or column can create invalid views. This is true for mappings specified through schema overrides or for default mappings. The specific schema mappings that can cause invalid views are as follows:

- Class to Table name
- Class to Geometry Column name
- Data Property to Column name
- Geometric Property to Column name
- Object Property to Table name

Table Name Restrictions When Working with a Foreign Schema

The ApplySchema command does not automatically create tables in foreign schemas. Therefore, the table specified for each class must already exist in the foreign schema. The Schema Overrides, specified through ApplySchema.SetPhysicalMapping, must contain a class to table mapping for each class whose table is named differently from the class. No mapping is required for classes where the table name and class name are the same.

Schema Access on a Different Oracle Instance

The following are Oracle limitations on foreign schema access if the schema is on a different (remote) Oracle instance:

- LOB type columns are not supported.
- Versioning and locking using Oracle Workspace Manager are not supported.

Logical to Physical Schema Mapping

The FDO ApplySchema command creates a table for each class in the command's class collection. The mapping of logical FDO data types to native types is described here.

NOTE The definition of the FDO Decimal data type requires that the precision property be set as a minimum to 1; in this case the scale property is set to 0 by default. Precision is the number of digits in the number, and scale is the number of digits to the right of the decimal point. The maximum number of digits in the native type is 38.

NOTE The definition of the FDO String data type requires that the length property be set.

| FDO Data Type | Native Type |
|---|----------------|
| BLOB | BLOB |
| Boolean | NUMBER(1) |
| Byte | NUMBER(3) |
| CLOB | Not supported. |
| DateTime | DATE |
| Decimal (with a precision of 20 and a scale of 4) | NUMBER(20,4) |
| Double | NUMBER |
| Int16 | NUMBER(5) |
| Int32 | NUMBER(10) |
| Int64 | NUMBER(20) |
| Single | NUMBER |
| String (the length property is set to 64) | NVARCHAR2(64) |

Physical to Logical Schema Mapping

When FDO describes the schema of an existing table that was not created by the ApplySchema command, it maps native data types to FDO data types. This mapping is described here.

The schema name is a concatenation of 'Fdo' and the Oracle user name, for example, 'FdoEXISTINGUSER'. The Oracle user name is used for the value of the Username and DataStore connection parameters. The class name is the name of the table. The property names are the column names.

The presence of a geometry property qualifies the class to be of the feature class type. If only one geometry property is present, it is referenced by the `OSGeo.FDO.Schema.FeatureClass.GeometryProperty` property. When a class has multiple geometric properties, the class `GeometryProperty` is chosen by some tie-breaking rules. A property with a spatial index is preferred over a property without a spatial index. A property that cannot be null is preferred over a property that is nullable. In the event of a tie, the `GeometryProperty` is null.

No column in an existing Oracle table can be mapped to an FDO integer type (Int16, Int32, and Int64). Integers are specified using the Oracle Number type where scale is 0 as in `NUMBER(9)` or `NUMBER(9,0)`. The maximum number that can be stored in a column of type `NUMBER(9)` is less than the maximum Int32 (2^{31}), so not all possible FDO Int32 values can be stored in a `NUMBER(9)`. The maximum number that can be stored in `NUMBER(10)` is greater than the maximum Int32, so not all of the possible values stored in a `NUMBER(10)` can be read into an Int32.

The mapping of Oracle native types to FDO data types is in the following table.

| Native Type As It Appears In Desc <Tablename> Output | Native Type As Specified In ALTER TABLE State- ment | FDO Data Type |
|---|--|---------------|
| CHAR(10) | Not different | String |
| VARCHAR2(128) | Not different | String |
| NCHAR(10) | Not different | String |
| NVARCHAR2(128) | Not different | String |
| NUMBER(38,1) | NUMBER(*,1) | Double |
| NUMBER(9) | Not different | Decimal |
| NUMBER(9,2) | Not different | Decimal |

| Native Type As It Appears In Desc <Tablename> Output | Native Type As Specified In ALTER TABLE State- ment | FDO Data Type |
|---|---|--------------------|
| NUMBER(7,-2) | Not different | Decimal |
| NUMBER | Not different | Double |
| NUMBER(20) | Not different | Decimal |
| NUMBER(10) | Not different | Decimal |
| DATE | Not different | DateTime |
| TIMESTAMP(6) | Not different | String |
| TIMESTAMP(6) WITH TIME ZONE | Not different | String |
| TIMESTAMP(6) WITH LOCAL TIME ZONE | Not different | String |
| BINARY_FLOAT | Not different | Single |
| BINARY_DOUBLE | Not different | Double |
| CLOB | Not different | String |
| NCLOB | Not different | String |
| XMLTYPE | Not different | Not mapped. |
| BLOB | Not different | BLOB |
| NUMBER(38) | INTEGER | Double |
| NUMBER(38) | Not different | Decimal |
| RAW(2000) | Not different | Not mapped. |
| URITYPE | Not different | Not mapped. |
| MDSYS.SDO_GEOMETRY | Not different | Geometric property |

FDO Provider for Oracle and Schema Overrides

Schema overrides are supported through the Overrides API that is specific to FDO Provider for Oracle. This API is published as part of the FDO SDK.

Schema Override Set

A schema override set is the set of schema overrides for a particular Feature Schema and FDO Provider.

The top level of a schema override set is very similar to the Feature Schema, itself. There is a root class (`OraclePhysicalSchemaMapping`), with a list of classes and a list of relations. These lists are subsets of the lists in the corresponding Feature Schema. It is not necessary to list every class and relation; list only the ones for which overrides are being specified. `OraclePhysicalSchemaMapping` provides the Oracle-specific implementation of `FdoPhysicalSchemaMapping`.

The methods for these MetaClasses are stripped down from the methods on the corresponding Feature Schema MetaClasses. In the Schema Override set, only name and physical properties are specified. For example, the names for schema objects can be specified, but not the descriptions, since the descriptions cannot be overridden. Name cannot be overridden either, but each object needs a name for identification, so it must be specified.

The Schema Override Set is used to specify schema-wide overrides such as:

- Oracle Database for all tables for classes and object properties in the schema. Defaults to the current Oracle Database for the current connection.
- Oracle Owner for all tables for classes and object properties in the schema. Defaults to the current Oracle Schema for the current connection.
- Tablespace for all tables for classes and object properties in the schema. Defaults to the default table space for the Oracle Owner.
- Default table mapping type for all classes in the schema. If not specified, the default table mapping type is Concrete.

These schema-wide overrides can themselves be overridden on an element-by-element basis. For example, there are overrides available for class table, object property, and geometric property.

Class Table Overrides

The RDBMS table for storing class properties can be specified by adding a table to the class. The table specifies the table name and table primary key name. By default, the table name is set to be the same as the class name.

Data Property Overrides

The physical representation for a data property can be overridden by attaching a column to it. The column specifies the name of the property's corresponding column in the FDO database. If Column is not specified, then the column names default to Name (the property name).

Object Property Overrides

The type of an Object Property is a class in a Feature Schema. This class can be considered the referenced class. This referenced class has properties, so a home for each property must be provided in the RDBMS data store. There are two different ways to store these properties. The Mapping Definition for each Object Property is specified by setting its MappingDefinition to an object of one of the following classes:

- **PropertyMappingSingle.** The referenced class properties are embedded in the containing class's table. The containing class is the class containing the Object Property.
- **PropertyMappingConcrete.** The Object Property is not stored in the containing class's table. A separate table is automatically generated for it.

Geometric Property Overrides

The column for a Geometric property can be overridden by attaching a Geometric column to it. Only the column name can be specified. The column type must always be mdsys.sdo_geometry.

The default column depends on whether the F_Geometry_0 table is present and whether the Geometric Property is also the GeometryProperty for its containing class.

If it is the GeometryProperty and F_Geometry_0 exists, then the table for this property is F_Geometry_<n>, where <n> is the ID of the Spatial Context Group

for the associated Spatial Context. If the Geometric Property is not associated with a Spatial Context, then <n> is the ID of the active Spatial Context group, at the time the geometric property is created. The column for the property is always RDBMS_GEOM.

Otherwise, the column name is assumed to be the same as the property name. The column table is assumed to be the table for the containing class.

Oracle-Specific Schema Creation Restrictions

This section describes the restrictions that apply when creating schema(s) using FDO Provider for Oracle.

FDOFeatureClass

An FdoFeatureClass must have an identity.

Classes

- Class names must be unique across the data store.
- FdoFeatureClass must define or inherit at least one IdentityProperty.

Properties

Restrictions apply to specific types of properties.

Data Properties

- The default value must not be specified.
- A non-nullable data property cannot be added to a class that already has data.

Identity Properties

- Identity properties cannot be nullable.
- Read-only Identity properties must be autogenerated.

String Properties

String property length must be between 1 and 4000 bytes inclusive.

Decimal Properties

- Decimal property precision must be between 0 and 38 inclusive.
- Decimal property scale must be between -84 and 127 inclusive.

Geometric Properties

- Only `FdoFeatureClass` can have geometric properties. A feature class can have multiple geometric properties; main geometry is not mandatory. `HasMeasure` and `HasElevation` are supported.
- If the geometric property values are stored in a feature geometry system table (`F_GEOMETRY_<n>`), then `HasMeasure` must be false.

Object Properties

- The object property class must be an `FdoClass`. (`FdoFeatureClass` is not allowed.)
- `IdentityProperty` is mandatory if `ObjectType` is not `FdoObjectType_Value` and the object property class has no identity properties.

Oracle-Specific Schema Modification Restrictions

This section describes restrictions that apply when modifying schema(s) using FDO Provider for Oracle.

Almost all modifications are disallowed, with the exception of those that follow.

Schema Element Descriptions

- Any schema element description is allowed.

- Any schema attribute dictionary (entries can be added, deleted, or modified) is allowed.

Data Properties

The read-only setting for a data property can be modified if the property is not autogenerated.

Views

This section describes the creation of a view in Oracle and the subsequent updating of the view in FDO. The view has two base tables, a table called PARCEL and one called OWNER. The PARCEL table contains information about a parcel of land including a foreign key identifying the owner of the land. The foreign key is an index into the OWNER table. The view is a join of PARCEL with OWNER in a many to one relationship of parcels to owner. The view is indexed by the PARCEL identity property.

You can use FDO to create the base tables, but you must use sqlplus to create the view and its index. You can use FDO to update columns in the view provided that the view satisfies the rules for updatable views. See http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/views001.htm#i1006232 for an explanation of these rules. . Use the name of the view as the name of the feature class in the FDO feature commands.

FDO requires that an entry be made in table user_sdo_geom_metadata to allow it to reverse engineer the coordinate system.

What follows are a description of the PARCEL and OWNER tables, the sql statements that create the view and its index, and the sql statement that adds the entry to the user_sdo_geom_metadata table.

PARCEL Table

| Column name | Null? | Type |
|-------------|----------|---------------|
| ID | NOT NULL | NUMBER(20) |
| PARCELSIZE | NULL | NUMBER |
| DISTRICT | NULL | NVARCHAR2(64) |
| OWNERID | NULL | NUMBER(20) |

| Column name | Null? | Type |
|-------------|-------|--------------------|
| GEOM | NULL | MDSYS.SDO_GEOMETRY |

OWNER Table

| Column name | Null? | Type |
|-------------|----------|---------------|
| ID | NOT NULL | NUMBER(20) |
| NAME | NULL | NVARCHAR2(32) |
| ADDRESS | NULL | NVARCHAR2(96) |

CREATE VIEW Sql Statement

```
create view parcelwithowner as select parcel.id, parcel.geom,
parcel.parcelsize, parcel.district, parcel.ownerid, owner.name,
owner.address from parcel, owner where parcel.ownerid = owner.id;
```

NOTE The PARCEL table is classified as a key-preserved table because its key, the ID column, is a key in the result of the join. The OWNER table is classified as a non key-preserved table because its key, the ID column, is not in the result of the join. As a result, the view can be used to update the columns of the PARCEL table but not the columns of the OWNER table.

ALTER VIEW Sql Statement

```
alter view parcelwithowner add constraint primarykey primary key
(id) disable novalidate;
```

NOTE View constraints are only supported in DISABLE NOVALIDATE mode.

INSERT INTO user_sdo_geom_metadata

```
INSERT INTO user_sdo_geom_metadata VALUES ('PARCEL', 'GEOM', MD
SYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0, 100, 0.001), MD
SYS.SDO_DIM_ELEMENT('Y', 0, 100, 0.001), MDSYS.SDO_DIM_ELEMENT('Z',
0, 100, 0.001)), 262152);
```

NOTE The MDSYS.SDO_DIM_ELEMENT's arguments are dimension name, lower boundary, upper boundary, and tolerance. The '262152' value is the SRID of the Non-Earth (Meter) coordinate system. This SQL statement, `select cs_name, srid from mdsys.cs_srs where wktext like '%LOCAL%';`, was used to pick this SRID.

Oracle-Specific Deletion Restrictions

This section describes restrictions that apply when performing deletion in a schema while using FDO Provider for Oracle.

FDOClassDefinition

FdoClassDefinition cannot be deleted if it has data (objects).

FdoClassDefinition cannot be deleted if another class in the data store has it as its base class.

FDOClass

FdoClass cannot be deleted if it is referenced by any object property in the data store.

Property

- A data property cannot be deleted if it has any non-null values.
- An object property cannot be deleted if it has data.
- A geometric property cannot be deleted if its containing class has data.

Oracle Reserved Words Used with Filter and Expression Text

When using a filter string with Oracle reserved words, the string within the expression must be encapsulated inside single quotes (following the same convention used with the SQL language). Failure to do so will result in a parsing error because the parser cannot determine any difference between the value and the keyword.

Example of a filter string:

```
AND='linetype'
```

This FDO constraint applies to the Oracle reserved words:

- AND

- DATE
- IN
- LIKE
- NOT
- OR

Locking and Long Transactions

The purpose of this section is two-fold. First, it illustrates ways of understanding the subtleties of the interactions between locking and long transactions in an Oracle context. Secondly, it provides concrete examples of those subtleties.

An FDO long transaction version is called a workspace in an Oracle context. In this discussion, the FDO phrase “long transaction version” is shortened to “long transaction”. A key phrase in the example is “root,” which represents permanent data. Any long transaction has a root long transaction as an ancestor. The Oracle Workspace Manager (OWM) name for the FDO root long transaction is “LIVE”.

Version Enabling

The Autodesk FDO Provider for Oracle creates tables in the FDO data store that are not automatically version-enabled. Therefore, when you create a new Oracle data store using the default options, the resulting table is not version-enabled, so persistent locking and long transaction are not supported. (This differs from previous releases.)

OWM is used for versioning and persistent locking support. To enable versioning, you must execute the *EnableVersioning.sql* script in the */FDO/bin/com* folder. This will enable the tables for OWM. Use SQL*Plus to execute the scripts.

NOTE If you create a data store in AutoCAD Map 3D 2008 that you want to use with the previous version of Autodesk Map 3D, you must set the value of the lock and long transaction options in the table `F_Options` in the generated data store to 2. You can do this with the supplied SQL script *EnableVersioning.sql*, which also enables versioning for all tables and allows the creation of conditional data. Do not make this change to `F_Options` in the database if you do not plan to use it with the previous version of Autodesk Map 3D.

Read the documentation contained within the script files themselves to determine what privileges are required for each script, how to run the scripts, and what errors may occur. Severe consequences can occur if you respond incorrectly to any errors you encounter while running a script.

NOTE The *DisableVersioning.sql* script in the same folder provides the opposite functionality.

Before executing the scripts, the following conditions must be true:

- You always connect directly to the Oracle user (or FDO data store) to be processed.
- The Oracle user executing the script has sufficient privileges (this user has been granted the Workspace Manager role WM_ADMIN_ROLE).
- The Oracle user executing the script is the only user processing or accessing the current Oracle user (or FDO data store) during the execution of the script. Otherwise, a script failure may result from a session conflict.

You can create a script log file by executing the `spool <log file name>;` command before invoking the scripts and the `spool off;` command after the invoked script finishes. The log file can help you resolve any issues encountered by the scripts.

OWM and FDO Lock Types

The following table shows the names of the Oracle Workspace Manager locks used to implement each FDO lock:

| FDO Lock Type | Oracle Lock Type |
|--------------------------------|---------------------|
| Shared | Shared |
| Exclusive | Workspace Exclusive |
| Long Transaction Exclusive | Version Exclusive |
| All Long Transaction Exclusive | Exclusive |

Example: AllLongTransactionExclusiveLock

The following is a proven example using the AllLongTransactionExclusiveLock type with multiple users and the Update command. When you connect to an Oracle data store, you are placed in the already-activated, default root long transaction. If a long transaction is created in root, it is considered a child of root. When the new long transaction is activated (for example, as LT1), the subsequent actions take place in the context of LT1. If another long transaction is subsequently created (for example, as LT2), it is created as a child of LT1.

NOTE When using FDO Provider for Oracle long transactions and locking, the combination of Oracle Workspace Manager capabilities and, potentially, other third-party applications can introduce many variables and combinations. The possible resulting conflicts in locking and long transactions can be similarly wide and varied.

This example considers two closely related cases. The same set of actions are taken in both cases, but in slightly different sequences, yielding different results. User1 creates a long transaction in the context of root and it is activated. User1 applies an AllLongTransactionExclusiveLock to a feature object in a data store. User1 updates that feature object in the data store. User2 attempts to update the same object, in the same data store, in the context of root. In the first case, User2 succeeds, and in the second case User2 fails (that is, a lock conflict is reported).

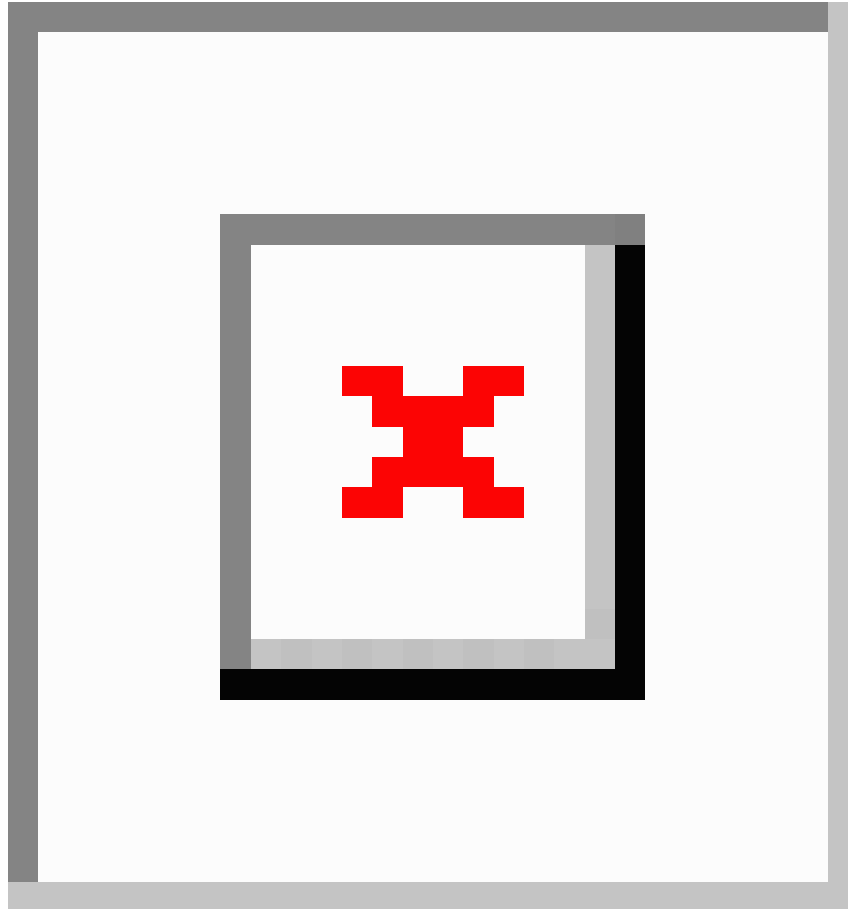
More specifically, the sequence of events for both cases is captured in the accompanying Long Transaction and Locking Sequencing Example diagram. For this example, all events occur in a single data store. The sequence of events are:

- 1 User1 creates LT1, activates LT1, and updates feature object “a” in LT1.
- 2 User2 successfully updates object “a” in root.
- 3 User1 creates LT2, activates LT2, and updates feature object “b” in LT2.
- 4 User2 fails to update object “b” in root.

The key difference is that, in LT1, User1 updates feature object “a” before the lock is applied, and, in LT2, User1 applies the lock to feature object “b” before it is updated. Prior to update, a copy of object “b” has not been made in LT2. This causes the lock to be applied to the copy of the object in root, because there is not yet a copy in LT2.

Therefore, if User1 intends to prevent anyone from modifying the object from the root level, User1 must apply the lock to the object *before* updating it.

For more information about Oracle Workspace Manager and its lock management, see the Oracle documentation.



Long Transaction and Locking Sequencing Example

FDO Provider for Oracle Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema

- Commands
- Filters
- Expressions
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- locking
- lock types: shared, exclusive, transaction, `AllLongTransactionExclusive`, `LongTransactionExclusive`
- transactions
- long transactions
- SQL
- XML configuration
- multiple spatial contexts
- specification of the coordinate system by name without specifying the WKT
- Write
- Multi-user write

Schema Capabilities

Use the `FdoSchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoSchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of 1 byte
- Byte data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of 165 digits (maximum decimal precision of 38 and maximum decimal scale of 127)
- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Int64 data type with a maximum length of 8 bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of 4000
- BLOB data type with a maximum length of 4000000000 bytes
- CLOB data type with a maximum length of 4000000000 bytes
- Int64 auto-generated data type
- Identity properties of type Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String
- Name size limitation of 30 for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Class`

- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: `.,:`
- Association properties
- Auto ID generation
- Composite ID
- Composite unique value constraints
- Datastore scope unique ID generation
- Default value
- Exclusive value range constraints
- Inclusive value range constraints
- Inheritance
- Multiple schemas
- Null value constraints
- Object properties
- Unique value constraints
- Schema modification
- Schema overrides
- Unique value constraints
- Value constraints list

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- FdoCommandType_Select
- FdoCommandType_SelectAggregates
- FdoCommandType_Insert
- FdoCommandType_Delete
- FdoCommandType_Update
- FdoCommandType_DescribeSchema
- FdoCommandType_DescribeSchemaMapping
- FdoCommandType_ApplySchema
- FdoCommandType_DestroySchema
- FdoCommandType_ActivateSpatialContext
- FdoCommandType_CreateSpatialContext
- FdoCommandType_DestroySpatialContext
- FdoCommandType_GetSpatialContexts
- FdoCommandType_CreateDataStore
- FdoCommandType_DestroyDataStore
- FdoCommandType_ListDataStores
- FdoCommandType_SQLCommand
- FdoCommandType_AcquireLock
- FdoCommandType_GetLockInfo
- FdoCommandType_GetLockedObjects
- FdoCommandType_GetLockOwners
- FdoCommandType_ReleaseLock
- FdoCommandType_ActivateLongTransaction
- FdoCommandType_CommitLongTransaction
- FdoCommandType_CreateLongTransaction
- FdoCommandType_DeactivateLongTransaction

- FdoCommandType_GetLongTransactions
- FdoCommandType_RollbackLongTransaction
- FdoRdbmsCommandType_CreateSpatialIndex
- FdoRdbmsCommandType_DestroySpatialIndex
- FdoRdbmsCommandType_GetSpatialIndexes

The following capabilities are supported:

- simple functions in Select and SelectAggregate commands
- use of expressions for properties in Select and SelectAggregates commands
- use of Distinct in SelectAggregates command
- availability of ordering in Select and SelectAggregates command
- availability of grouping criteria in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, spatial, and distance
- the Within distance operation
- spatial operations of type CoveredBy, Inside, Intersects, EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic, function, and parameter expressions are supported.

The following functions are supported:

- Double Avg(<type> value) where <type> is one of Decimal, Double, Single, Int16, Int32, or Int64.
- Decimal Ceil(<type> value) where <type> is one of Decimal, Double, or Single
- String Concat(String str1Val, String str2Val)
- Int64 Count(<type> value) where <type> is one of Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, String, BLOB, or CLOB
- Decimal Floor(<type> value) where <type> is one of Decimal, Double, or Single
- String Lower(String value)
- Byte Max(<type> value) where <type> is one of Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String.
- Byte Min(<type> value) where <type> is one of Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String.
- Double Sum(<type> value) where <type> is one of Decimal, Double, Int16, Int32, Int64, or Single.
- String Upper(String value)
- GeometricProperty SpatialExtents(GeometricProperty property)

Geometry Capabilities

Use the `FdoGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types `Ring`, `LinearRing`, `CircularArcSegment`, and `LineStringSegment` are supported. The following geometry types are supported.

- Point
- LineString
- Polygon

- MultiPoint
- MultiLineString
- MultiPolygon
- CurveString
- CurvePolygon
- MultiCurveString
- MultiCurvePolygon

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for ArcSDE

B

This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for ArcSDE.

What Is FDO Provider for ArcSDE?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. ESRI® ArcSDE® (Spatial Database Engine) is part of the ArcGIS 9 system. ArcSDE manages the exchange of information between an (ArcGIS 9 Desktop) application and a relational database management system. FDO Provider for ArcSDE provides FDO with access to an ArcSDE 9-based data store, which, in this case, must be Oracle 9i (9.2.0.6).

FDO Provider for ArcSDE Software Requirements

Installed Components

FDO Provider for ArcSDE dynamically linked libraries are installed with the FDO SDK. They are located in <FDO SDK Install Location>\FDO\bin. You do not have to do anything to make these DLLs visible.

External Dependencies

The operation of FDO Provider for ArcSDE is dependent on the presence of ArcSDE 9 and a supported data source, such as Oracle 9i, in the network environment. The host machine running FDO Provider for ArcSDE must also have the required DLLs present, which are available by installing either an ArcGIS 9.1 Desktop application or the ArcSDE SDK. For example, the required DLLs are present if either ArcView®, ArcEditor®, or ArcInfo® are installed. For more information about ArcGIS 9.1 Desktop applications and the ArcSDE SDK, refer to the ESRI documentation.

Specifically, in order for FDO Provider for ArcSDE to run, three dynamically linked libraries, sde91.dll, sg91.dll, and pe91.dll, are required and you must ensure that the PATH environment variable references the local folder containing these DLLs. For example, in Microsoft Windows, if ArcGIS 9.1 Desktop is installed to C:\Program Files\ArcGIS, then the required ArcSDE binaries are located at C:\Program Files\ArcGIS\ArcSDE\bin. Similarly, if the ArcSDE SDK is installed to the default location, then the required ArcSDE binaries are located at C:\ArcGis\ArcSDE\bin. The absence of this configuration may cause the following exception message "The ArcSDE runtime was not found."

FDO Provider for ArcSDE Limitations

The FDO Provider for ArcSDE is based on a subset of the ArcSDE API. This subset does not include the following:

- Raster functionality
- Native ArcSDE metadata
- The annotation data, with the exception of the ANNO_TEXT column

ArcSDE Limitations

FDO Provider for ArcSDE must abide by limitations of the ArcSDE technology to which it connects. This section discusses these limitations.

Relative to ArcObjects API and ArcGIS Server API

The ArcSDE API does not support the following advanced functionality found in the ArcObjects API and the newer ArcGIS Server API:

- Advanced geometries, such as Bezier curves and ellipses
- Relationships
- Topology
- Networks
- Analysis
- Linear referencing

Curved Segments

If ArcSDE encounters curved segments, it will automatically tessellate them. This means that if you create a geometry containing an arc segment in an ArcSDE table using ArcObjects API and then you try to read that geometry back using the ArcSDE API, you will get a series of line segments that approximate the original arc segment. That is, you get an approximation of the original geometry.

Locking and Versioning

ArcSDE permits row locks or table versioning provided that the ID column, which uniquely identifies the row, is maintained by ArcSDE. If there is no ID column or the ID column is maintained by the user, ArcSDE does not permit row locking or table versioning to be enabled.

NOTE In ArcSDE you can either lock rows in a table or version a table, but you cannot do both at the same time. To do either, you must alter the table's registration.

The following sections illustrate these three steps:

- 1 The creation of a table.
- 2 The alteration of the table registration to identify one of the column definitions as the row ID column and to enable row locking.

- 3 The alteration of the table registration to disable row locking and to enable versioning.

Table Creation

The command is:

```
sddtable -o create -t hassdemaintainedrowid -d "name string(20),  
fid integer(9)" -u t_user -p test
```

The output of the describe registration command (sddtable -o describe_reg) for this table is as follows:

NOTE The Row Lock has no value and the value of Dependent Objects is None.

| | |
|--------------------|----------------------------------|
| Table Owner | : T_USER |
| Table Name | : HASSEMAINTAINEDROWID |
| Registration Id | : 18111 |
| Row ID Column | : |
| Row ID Column Type | : |
| Row Lock | : |
| Minimum Row ID | : |
| Dependent Objects | : None |
| Registration Date | : 02/24/05 13:08:02 |
| Config. Keyword | : DEFAULTS |
| User Privileges | : SELECT, UPDATE, INSERT, DELETE |
| Visibility | : Visible |

Identity Row ID Column and Enable Row Locking

The command is:

```
sddtable -o alter_reg -t hassdemaintainedrowid -c fid -C sde -L  
on -u t_user -p test
```

The output of the describe registration command (sddtable -o describe_reg) for this table is as follows.

NOTE The Row ID Column value is FID, the Row ID Column Type value is SDE Maintained, and the Row Lock value is Enable.

```

Table Owner          : T_USER
Table Name           : HASSEMAINTAINEDROWID
Registration Id       : 18111
Row ID Column        : FID
Row ID Column Type    : SDE Maintained
Row ID Allocation     : Many
Row Lock             : Enable
Minimum Row ID       : 1
Dependent Objects    : None
Registration Date     : 02/24/05 13:08:02
Config. Keyword       : DEFAULTS
User Privileges      : SELECT, UPDATE, INSERT, DELETE
Visibility           : Visible

```

Disable Row Locking and Enable Versioning

The command is:

```

sddtable -o alter_reg -t hassdemaintainedrowid -L off -V MULTI -u
t_user -p test

```

The output of the describe registration command (sddtable -o describe_reg) for this table is as follows:

NOTE The “Row Lock” is “Not Enable” and “Dependent Objects” is “Multiversion Table”.

```

Table Owner          : T_USER
Table Name           : HASSEMAINTAINEDROWID
Registration Id       : 18111
Row ID Column        : FID
Row ID Column Type    : SDE Maintained
Row ID Allocation     : Many
Row Lock             : Not Enable
Minimum Row ID       : 1
Dependent Objects    : Multiversion Table
Dependent Object Names : A18111, D18111
Registration Date     : 02/24/05 13:08:02
Config. Keyword       : DEFAULTS
User Privileges      : SELECT, UPDATE, INSERT, DELETE
Visibility           : Visible

```

FDO Provider for ArcSDE Connection

This information supplements the Establishing a Connection chapter. You connect to an ArcSDE data store indirectly through the ArcSDE server. The underlying data source for the data store is a database, such as Oracle. The ArcSDE server is connected to the data source and mediates the requests that you send it.

You can connect to FDO Provider for ArcSDE in one step if you already know the name of the data store that you want to use. Otherwise, you must connect in two steps.

The minimum required connection properties for the initial call to `Open()` are server, instance, username, and password. Multiple users can access the data store. However, access is password-protected. The server property is the name of the machine hosting the ArcSDE server. The instance property acts as an index into an entry in the services file. An entry contains port and protocol information used to connect to the ArcSDE server. On a Windows machine, the services file is located in `C:\WINDOWS\system32\drivers\etc`. Assuming that the instance name is `"esri_sde"`, an entry would look something like this: `"esri_sde 5151/tcp #ArcSDE Server Listening Port"`.

An ArcSDE data source may contain more than one data store. For the first call to `Open()`, a data store name is optional. If successful, the first call to `Open()` results in the data store parameter becoming a required parameter and a list of the names of the data stores in the data source becoming available. You must choose a data store and call `Open()` again.

If the data source supports multiple data stores, the list returned by the first call to `Open()` will contain a list of all of the data stores resident in the data source. Otherwise, the list will contain one entry: `"Default Data Store"`.

If you know the name of the data store, you can provide it for the first call to `Open()` and make the connection in one step.

Data Type Mappings

This section shows the mappings from FDO data types to ArcSDE data types to Oracle data types:

| FDO DataType | sdetable Column Definition | Oracle Column Type |
|---------------------|----------------------------|--------------------|
| FdoDataType_Boolean | Not supported | Not supported |
| FdoDataType_Byte | Not supported | Not supported |

| FDO DataType | sdetable Column Definition | Oracle Column Type |
|----------------------|--|----------------------------|
| FdoDataType_DateTime | date | DATE |
| FdoDataType_Decimal | Not supported | Not supported |
| FdoDataType_Double | double(38,8) | NUMBER(38,8) |
| FdoDataType_Int16 | integer(4) | NUMBER(4) |
| FdoDataType_Int32 | integer(10) | NUMBER(10) |
| FdoDataType_Int64 | Not supported | Not supported |
| FdoDataType_Single | float(6,2) // typical float(0<n<=6, 0<m<DBMSLimit)) // possible | NUMBER(6,2) NUMBER(n,8) |
| FdoDataType_String | string(<length>) | VARCHAR2(<length>) |
| FdoDataType_BLOB | blob | LONG RAW |
| FdoDataType_CLOB | Not supported | Not supported |
| FdoDatatype_UniqueID | Not supported | Not supported |

Creating a Feature Schema

This section describes the creation of the SampleFeatureSchema, which is the example feature schema described in the Schema Management chapter. It also describes the creation of the OGC980461FS schema, which is the schema defined in the OpenGIS project document 98-046r1.

FDO Provider for ArcSDE does not support the creation or destruction of feature schema (that is, does not support the FdoIApplySchema and FdoIDestroySchema commands.) However, it does support the FdoIDescribeSchema command. The intended use of FDO Provider for ArcSDE is to operate on already existing feature schemas. FDO Provider for ArcSDE supports inserting, selecting, updating, and deleting data in existing schemas.

You can use FDO Provider for ArcSDE to operate on a new feature schema. However, you must create the schema using ArcSDE tools. In particular you use the sdetable and sdelayer commands, which can be used to create a schema in any of the data store technologies used by ArcSDE. This part of the

description is generic. Other parts of the description are specific to Oracle and to Windows XP because Oracle is the data store technology and Windows XP is the operating system for this exercise.

First, you must create an Oracle username for the feature schema (that is, the name of the Oracle user is the name of the feature schema.) To do this, you connect as system administrator to the Oracle instance used by the ArcSDE server. The following command creates the user and grants to that user the privileges necessary for the ArcSDE tool commands to succeed:

```
grant connect,resource to <schemaName> identified by <password>
```

Secondly, you must log in to the host where the ArcSDE server is running. ArcSDE tools are on the host machine where the ArcSDE server resides.

TIP NetMeeting can be used to remotely login to where the ArcSDE Server is running and launch a command window (that is, in the Run dialog box, enter cmd) The ArcSDE tool commands can be executed through the command window. Do not use C:\WINDOWS\SYSTEM32\COMMAND.COM because the line buffer is too short to contain the entire text of some of the SDE tool command strings.

Finally, execute the `sdetable` and `sdelay` commands in a command window to create each of the classes. Since you are executing these commands on the host where the ArcSDE server is located, you can omit the server name option. If the ArcSDE server is connected to only one data store, you can omit the service option. For more information about all of the ArcSDE commands, consult the ArcSDE Developer Help Guide.

SampleFeatureSchema

In this sample a feature schema called `SampleFeatureSchema` is created, which contains one feature class called `SampleFeatureClass`. This feature class has the following three properties:

- An `Int32` called `SampleIdentityDataProperty`.
- A string called `SampleNameDataProperty`.
- A polygon geometry called `SampleGeometricProperty`.

First, use the `sdetable -o create` command to add the integer and string properties to `SampleFeatureClass`. Then, use the `sdetable -o alter_reg` command to identify the `SampleIdentityDataProperty` as an identity property. Finally, use the `sdelay -o add` command to add the geometric property to `SampleFeatureClass`. This assumes that only one ArcSDE server service is

running so that the -i option is optional. The -i option takes a service name as an argument.

The `sddtable -o create` command can be invoked as follows:

```
sddtable -o create -t SampleFeatureClass -d "SampleIdentityData
Property INTEGER(10), SampleNameDataProperty STRING(64)" -u
SampleFeatureSchema -p test.
```

The -o option takes the command option name. The -d option takes the column definitions, which is a quoted list of column name/column type pairs delimited by commas. The -u option takes an Oracle database user name, which becomes the feature schema name. The -p option takes a password.

The `sddtable -o alter_reg` command is invoked as follows:

```
sddtable -o alter_reg -t SampleFeatureClass -c SampleIdentityData
Property -C USER -u SampleFeatureSchema -p test
```

The -c option identifies the column name that will be the identity property. The -C option indicates whether SDE is supposed to generate the value or obtain it from the user. You will be prompted to confirm that you want to alter the registration of the table.

The `sddlayer` command is invoked as follows:

```
sddlayer -o add -l SampleFeatureClass,SampleGeometricProperty -E
0,0,100,50 -e a -u SampleFeatureSchema -p test
```

The -o option takes the command option name. The -l option identifies the table and column. The -E option identifies the extents; the arguments are <xmin,ymin,xmax,ymax>. The -e option identifies the geometry type with 'a' indicating an area shape.

OGC980461FS

This schema contains the ten classes defined in the OpenGIS Project Document *980946r1*. The types of the properties belonging to the classes is similar to that of `SampleFeatureClass`, namely, an integer, a string, and a geometry. One difference is that the geometry in three of the classes is multipart. Two of them have MULTIPOLYGON geometries, and one of them has a MULTILINESTRING geometry. A multipart geometry is indicated by adding a '+' to the entity argument to the -e option in the `sddlayer` command. A MULTIPOLYGON geometry is indicated by "-e a+", and a MULTILINESTRING geometry is indicated by "-e l+".

An ArcSDE table cannot have two geometries. This restriction impacts the definition of the buildings class, which has a POLYGON and a POINT geometry. We have chosen to add the POINT geometry. The OpenGIS 98-046r1

document defines one query that references building objects, and the POINT geometry supports this query.

NOTE The use of -E option in the sdelayer command defines the extents. The arguments are <xmin,ymin,xmax,ymax>. The values provided below ensure that you will not receive any “ordinate out of bounds” errors when inserting the 98046r1 data.

ArcSDE Commands That Define the OGC980461FS Classes

```

sdetable -o create -t lakes -d "fid integer(10), name string(64)"
-u OGC980461FS -p test
sdetable -o alter_reg -t lakes -c fid -C user -u OGC980461FS -p
test
sdelayer -o add -l lakes,shore -E 0,0,100,50 -e a -u OGC980461FS
-p test
sdetable -o create -t road_segments -d "fid integer(10), name
string(64), aliases string(64), num_lanes integer(10)" -u
OGC980461FS -p test
sdetable -o alter_reg -t road_segments -c fid -C user -u
OGC980461FS -p test
sdelayer -o add -l road_segments,centerline -E 0,0,100,50 -e l -u
OGC980461FS -p test
sdetable -o create -t divided_routes -d "fid integer(10), name
string(64), num_lanes integer(10)" -u OGC980461FS -p test
sdetable -o alter_reg -t divided_routes -c fid -C user -u
OGC980461FS -p test
sdelayer -o add -l divided_routes,centerlines -E 0,0,100,50 -e l+
-u OGC980461FS -p test
sdetable -o create -t forests -d "fid integer(10), name string(64)"
-u OGC980461FS -p test
sdetable -o alter_reg -t forests -c fid -C user -u OGC980461FS -p
test
sdelayer -o add -l forests,boundary -E 0,0,100,50 -e a+ -u
OGC980461FS -p test
sdetable -o create -t bridges -d "fid integer(10), name string(64)"
-u OGC980461FS -p test
sdetable -o alter_reg -t bridges -c fid -C user -u OGC980461FS -p
test
sdelayer -o add -l bridges,position -E 0,0,100,50 -e p -u
OGC980461FS -p test
sdetable -o create -t streams -d "fid integer(10), name string(64)"
-u OGC980461FS -p test
sdetable -o alter_reg -t streams -c fid -C user -u OGC980461FS -p
test
sdelayer -o add -l streams,centerline -E 0,0,100,50 -e l -u
OGC980461FS -p test
sdetable -o create -t buildings -d "fid integer(10), address
string(64)" -u OGC980461FS -p test
sdetable -o alter_reg -t buildings -c fid -C user -u OGC980461FS
-p test
sdelayer -o add -l buildings,position -E 0,0,100,50 -e p -u
OGC980461FS -p test

```



```

sddtable -o create -t ponds -d "fid integer(10), name string(64),
type string(64)" -u OGC980461FS -p test
sddtable -o alter_reg -t ponds -c fid -C user -u OGC980461FS -p
test
sddlayer -o add -l ponds,shores -E 0,0,100,50 -e a+ -u OGC980461FS
-p test
sddtable -o create -t named_places -d "fid integer(10), name
string(64)" -u OGC980461FS -p test
sddtable -o alter_reg -t named_places -c fid -C user -u OGC980461FS
-p test
sddlayer -o add -l named_places,boundary -E 0,0,100,50 -e a -u
OGC980461FS -p test
sddtable -o create -t map_neatlines -d "fid integer(10)" -u
OGC980461FS -p test
sddtable -o alter_reg -t map_neatlines -c fid -C user -u
OGC980461FS -p test
sddlayer -o add -l map_neatlines,neatline -E 0,0,100,50 -e a -u
OGC980461FS -p test

```

Logical to Physical Schema Mapping

This mapping does not apply because the ArcSDE provider does not support the ApplySchema command.

Physical to Logical Schema Mapping

When FDO describes the schema of an existing table that was not created by the ApplySchema command, it maps native data types to FDO data types. The ArcSDE provider talks to the ArcSDE server. The server uses either Oracle or Sql Server as the back-end data store technology. So two schema mappings are described here.

Oracle 10gR2 Back-End

The schema name is the name of the user supplied as the value of the Username connection parameter. The class name is the name of the table created using the sddtable command. The property names are the column names.

The following ArcSDE commands were used to create a table. The sddtable creates a business table. The sddlayer command converts the business table into a feature class table.

NOTE The `sdetable -i` option designates the ArcSDE server instance against which to apply the command. The value can be a symbolic name or a port number. If the value is a symbolic name, the symbolic name must be mapped to a tcp port number in the `C:\WINDOWS\system32\drivers\etc\services` file.

NOTE The `sdelayer -e` argument “np” means that the geometry is either a 2D point or NULL. The `-l` argument “existingtable,featgeom” means a geometry column called `featgeom` in a table called `existingtable`. The `-E` argument “empty” means that the layer envelope is empty. The `-P` argument “32” means that the geometry is stored with 32-bit precision. The `-t` argument “S” means that the storage type used for the geometry is Oracle Spatial. The `-G` argument “26910” is the SRID of a coordinate system in the `pedef.h` file as well as in the Oracle `mdsys.cs_srs` table. You can access this file by googling “pedef.h”. It contains the following line:

```
#define PE_PCS_NAD_1983_UTM_10N 26910 /* NAD 1983 UTM Zone 10N */.
```

The `sqlplus` command `select cs_name from mdsys.cs_srs where srid = 26910;` returns NAD83 / UTM zone 10N.

```
sdetable -o create -t existingtable -d "smInt4Col smallint(4),
int16Col int16, int32Col int32, float32Col float32, float64Col
float64, int10 integer(10), float4_4 float(4,4), double7_2
double(7,2), string64 string(64), blobCol blob, clobCol clob, id
uuid, nStringCol nstring, nClobCol nclob, dateCol date" -u existin
guser -p test -i esri_sde
sdelayer -o add -l existingtable,featgeom -e np -E empty -i
esri_sde -u existinguser -p test -G 26910 -P 32 -t S
```

The following table maps the output of the `sdetable -o describe` command to the output of the `sqlplus desc` command to the output of the FDO `DescribeSchema` command.

| Column | ArcSDE Type, Length, decimal places | Oracle Type | FDO Type |
|------------|-------------------------------------|--------------|----------|
| SMINT4COL | SE_INT16, 4 | NUMBER(4) | Int16 |
| ID | SE_UUID, 38 | CHAR(38) | Unmapped |
| INT16COL | SE_INT16, 4 | NUMBER(4) | Int16 |
| INT32COL | SE_INT32, 10 | NUMBER(38) | Int32 |
| FLOAT32COL | SE_FLOAT32, 6, 2 | NUMBER(6,2) | Single |
| FLOAT64COL | SE_FLOAT64, 7, 2 | NUMBER(15,4) | Double |

| Column | ArcSDE Type, Length, decimal places | Oracle Type | FDO Type |
|------------|-------------------------------------|--------------------|----------|
| INT10 | SEINT32, 10 | NUMBER(10) | Int32 |
| FLOAT4_4 | SE_FLOAT32, 4, 4 | NUMBER(4,4) | Single |
| DOUBLE7_2 | SE_FLOAT64, 7, 2 | NUMBER(7,2) | Double |
| STRING64 | SE_STRING, 64 | VARCHAR2(64) | String |
| BLOBCOL | SE_BLOB, 0 | BLOB | BLOB |
| CLOBCOL | SE_CLOB, 0 | CLOB | Unmapped |
| NSTRINGCOL | SE_NSTRING, 255 | NVARCHAR2(255) | String |
| NCLOBCOL | SE_NCLOB, 0 | NCLOB | Unmapped |
| DATECOL | SE_DATE, 0 | DATE | DateTime |
| FEATGEOM | SE_SHAPE, 0 | MDSYS.SDO_GEOMETRY | Geometry |
| OBJECTID | SE_INT32, 10 | NUMBER(38) | Int32 |

NOTE The OBJECTID column was not specified in the -d arguments of the sdetable command.

Sql Server 2005 Back-End

The schema name is the name of the user supplied as the value of the Username connection parameter prefixed by "SDE_". The class name is the name of the table created using the sdetable command. The property names are the column names.

The following ArcSDE commands were used to create a table. The sdetable creates a business table. The sdelayer command converts the business table into a feature class table.

NOTE The sdetable -i option designates the ArcSDE server instance against which to apply the command. The value can be a symbolic name or a port number. If the value is a symbolic name, the symbolic name must be mapped to a tcp port number in the C:\WINDOWS\system32\drivers\etc\services file.

NOTE The `sdelayer -e` argument “np” means that the geometry is either a 2D point or NULL. The `-l` argument “existingtable,featgeom” means a geometry column called `featgeom` in a table called `existingtable`. The `-E` argument “empty” means that the layer envelope is empty. The `-P` argument “32” means that the geometry is stored with 32-bit precision. The `-t` argument “B” means that the storage type used for the geometry is Esri Binary. The `-G` argument “26910” is the SRID of a coordinate system in the `pedef.h` file as well as in the Oracle `mdsys.cs_srs` table. You can access this file by googling “pedef.h”. It contains the following line:

```
#define PE_PCS_NAD_1983_UTM_10N 26910 /* NAD 1983 UTM Zone 10N */.
```

```
sddtable -o create -t existingtable -d "smInt4Col smallint(4),
int16Col int16, int32Col int32, float32Col float32, float64Col
float64, int10 integer(10), float4_4 float(4,4), double7_2
double(7,2), string64 string(64), blobCol blob, clobCol clob, id
uuid, nStringCol nstring, nclobCol nclob, dateCol date" -u existin
guser -p test -i esri_sde_ss
sdelayer -o add -l existingtable,featgeom -e np -E empty -i
esri_sde_ss -u existinguser -p test -G 26910 -P 32 -t B
```

The following table maps the output of the `sddtable -o describe` command to the output of the `sqlplus desc` command to the output of the FDO `DescribeSchema` command.

| Column | ArcSDE Type, Length, decimal places | Sql Server Type | FDO Type |
|------------|-------------------------------------|------------------|----------|
| SMINT4COL | SE_INT16, 4 | smallint | Int16 |
| ID | SE_UUID, 38 | uniqueidentifier | Unmapped |
| INT16COL | SE_INT16, 4 | smallint | Int16 |
| INT32COL | SE_INT32, 10 | int | Int32 |
| FLOAT32COL | SE_FLOAT32, 6, 2 | real | Single |
| FLOAT64COL | SE_FLOAT64, 7, 2 | float | Double |
| INT10 | SEINT32, 10 | int | Int32 |
| FLOAT4_4 | SE_FLOAT32, 4, 4 | numeric(4,4) | Single |
| DOUBLE7_2 | SE_FLOAT64, 7, 2 | numeric(7,2) | Single |

| Column | ArcSDE Type, Length, decimal places | Sql Server Type | FDO Type |
|------------|-------------------------------------|-----------------|----------|
| STRING64 | SE_STRING, 64 | varchar(64) | String |
| BLOBCOL | SE_BLOB, 0 | image | BLOB |
| CLOBCOL | SE_CLOB, 0 | text | Unmapped |
| NSTRINGCOL | SE_NSTRING, 255 | nvarchar(255) | String |
| NCLOBCOL | SE_NCLOB, 0 | ntext | Unmapped |
| DATECOL | SE_DATE, 0 | date | DateTime |
| FEATGEOM | SE_SHAPE, 0 | int | Geometry |

FDO Provider for ArcSDE Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- locking
- exclusive locking type
- transactions
- long transactions
- SQL
- multiple spatial contexts
- specifying coordinate systems by name or ID without specifying WKT
- Write
- Multi-user write

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- DateTime data type with a maximum length of 12 bytes
- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of 4294967296
- BLOB data type with a maximum length of 4294967296
- Int32 auto-generated data type

- Identity properties of type DateTime, Double, Int16, Int32, Single, String, and BLOB
- Name size limitation of 123 for a schema element name of type FdoSchemaElementNameType_Datastore
- Name size limitation of 65 for a schema element name of type FdoSchemaElementNameType_Schema
- Name size limitation of 160 for a schema element name of type FdoSchemaElementNameType_Class
- Name size limitation of 32 for a schema element name of type FdoSchemaElementNameType_Property
- Name size limitation of 64 for a schema element name of type FdoSchemaElementNameType_Description
- Characters that cannot be used for a schema element name: ::
- Auto ID generation
- Composite unique value constraints
- Multiple schemas
- Null value constraints
- Unique value constraints

Command Capabilities

Use the FdoICommandCapabilities object methods to learn about these capabilities. You can get this object by calling the GetCommandCapabilities() method on the FdoIConnection object. For an explanation of the meaning of the capabilities, consult the FdoICommandCapabilities class description in the FDO API Reference documentation.

The following commands are supported:

- FdoCommandType_Select
- FdoCommandType_SelectAggregates
- FdoCommandType_Insert
- FdoCommandType_Delete
- FdoCommandType_Update

- FdoCommandType_DescribeSchema
- FdoCommandType_ActivateSpatialContext
- FdoCommandType_CreateSpatialContext
- FdoCommandType_DestroySpatialContext
- FdoCommandType_GetSpatialContexts
- FdoCommandType_SQLCommand
- FdoCommandType_AcquireLock
- FdoCommandType_GetLockInfo
- FdoCommandType_GetLockedObjects
- FdoCommandType_GetLockOwners
- FdoCommandType_ReleaseLock
- FdoCommandType_ActivateLongTransaction
- FdoCommandType_DeactivateLongTransaction
- FdoCommandType_CommitLongTransaction
- FdoCommandType_CreateLongTransaction
- FdoCommandType_GetLongTransactions
- FdoCommandType_RollbackLongTransaction
- FdoCommandType_ListDataStores

The following capabilities are supported:

- command parameterization
- simple functions in Select and SelectAggregate commands
- use of Distinct in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities,

consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, spatial, and distance
- the Beyond and Within distance operations
- spatial operations of type Contains, Crosses, Disjoint, Equals, Intersects, Overlaps, Touches, Within, CoveredBy, Inside, and EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic expressions are supported.

The following functions are supported:

- `Double Sum(<type> value)` where <type> is one of Double, Single, Int16, or Int32.
- `Int64 Count(<type> value)` where <type> is one of Boolean, Double, Single, Decimal, Byte, DateTime, Int16, Int32, Int64, String, BLOB, CLOB, ObjectProperty, GeometricProperty, AssociationProperty, or RasterProperty
- `Double Avg(<type> value)` where <type> is one of Double, Single, Int16, or Int32.
- `Double Max(<type> value)` where <type> is one of Double, Single, Int16, or Int32.
- `Double StdDev(<type> value)` where <type> is one of Double, Single, Int16, or Int32.

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types LinearRing and LineStringSegment are supported. The following geometry types are supported.

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

Raster Capabilities

Use the FdoIRasterCapabilities object methods to learn about these capabilities. You can get this object by calling the GetRasterCapabilities() method on the FdoIConnection object. For an explanation of the meaning of the capabilities, consult the FdoIRasterCapabilities class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for MySQL



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for MySQL.

What Is FDO Provider for MySQL?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for MySQL provides FDO with access to a MySQL-based data store.

The FDO Provider for MySQL API provides custom commands that are specifically designed to work with the FDO API. For example, using these commands, you can do the following:

- Gather information about a provider.
- Transmit client services exceptions.
- Get lists of accessible data stores.
- Create connection objects.
- Create and execute spatial queries.

The MySQL architecture supports different storage engines. Choose an engine as needed, depending on its characteristics and capabilities, such as the following:

- MyISAM is a disk-based storage engine. It does not support transactions.
- InnoDB is a disk-based storage engine. It has full ACID transaction capability.

- Memory (Heap) is a storage engine utilizing only RAM. It is very fast.
- NDB is the MySQL Cluster storage engine.
- MERGE is a variation of MyISAM. A MERGE table is a collection of identical MyISAM tables, which means that all tables have the same columns, column types, indexes, and so on.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *OSGeo FDO Provider for MySQL API Reference Help* (MySQL_Provider_API.chm).

Logical to Physical Schema Mapping

The FDO ApplySchema command creates a table for each class in the command's class collection. The mapping of logical FDO data types to native types is described here.

NOTE The definition of the FDO Decimal data type requires that the precision property be set as a minimum to 1; in this case the scale property is set to 0 by default. Precision is the number of digits in the number, and scale is the number of digits to the right of the decimal point. The maximum number of digits in the native type is 64.

NOTE The definition of the FDO String data type requires that the length property be set.

| FDO Data Type | Native Data Type |
|-------------------------------------|---------------------|
| Boolean | tinyint(4) |
| Byte | tinyint(3) unsigned |
| DateTime | datetime |
| Decimal (precision = 20; scale = 4) | decimal(20,4) |
| Double | double |
| Int16 | smallint(6) |
| Int32 | int(11) |

| FDO Data Type | Native Data Type |
|-----------------------|------------------|
| Int64 | bigint(20) |
| Single | double |
| String (length is 64) | varchar(64) |

Physical to Logical Schema Mapping

When FDO describes the schema of an existing table that was not created by the ApplySchema command, it maps native data types to FDO data types. This mapping is described here.

The name of the schema is the concatenation of 'Fdo' and the name of the database. This is the database created using the 'CREATE DATABASE' command and containing the existing table(s). The table names are the class names, and the column names are the property names.

NOTE The length of a bit field is in the range 1-64. A row in a column of type bit(64) can hold 64 bit values.

NOTE The following native types are not mapped to FDO data types: float, float unsigned, real unsigned, double unsigned, binary, varbinary, tinyblob, blob, mediumblob, longblob, tinytext, mediumtext, and longtext.

| Native Type | Defaults | FDO Data Type |
|-------------------|------------|---------------|
| bit(| length = 1 | Boolean |
| bit(2) | | Byte |
| bit(64) | | Int64 |
| tinyint | length = 4 | Int16 |
| tinyint unsigned | length = 3 | Byte |
| smallint | length = 6 | Int16 |
| smallint unsigned | length = 5 | Int32 |

| Native Type | Defaults | FDO Data Type |
|--------------------|---------------------------|-------------------------------------|
| mediumint | length = 9 | Int32 |
| mediumint unsigned | length = 8 | Int32 |
| int | length = 11 | Int32 |
| int unsigned | length = 10 | Int64 |
| bigint | length = 20 | Int64 |
| bigint unsigned | length = 20 | Int64 |
| real | | Double |
| double | | Double |
| numeric | precision = 10; scale = 0 | Decimal (precision = 10; scale = 0) |
| numeric unsigned | precision = 10; scale = 0 | Decimal (precision = 10; scale = 0) |
| date | | DateTime |
| datetime | | DateTime |
| timestamp | | DateTime |
| year | | Int32 |
| char(64) | | String (length = 192) |
| varchar(64) | | String (length = 192) |
| enum | | String |
| set | | String |

FDO Provider for MySQL Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- transactions
- SQL
- multiple spatial contexts
- Write
- Multi-user write

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of 1 byte
- Byte data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of 95 digits (maximum decimal precision of 65 and maximum decimal scale of 30)
- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Int64 data type with a maximum length of 8 bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of 2147483647
- Int64 auto-generated data type
- Identity properties of type Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String
- Name size limitation of 64 for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of 200 for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of 200 for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: `::`
- Association properties
- Auto ID generation

- Composite ID
- Composite unique value constraints
- Datastore scope unique ID generation
- Default value
- Exclusive value range constraints
- Inclusive value range constraints
- Inheritance
- Multiple schemas
- Null value constraints
- Object properties
- Unique value constraints
- Schema modification
- Schema overrides
- Unique value constraints

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_Insert`
- `FdoCommandType_Delete`
- `FdoCommandType_Update`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_ApplySchema`

- FdoCommandType_DestroySchema
- FdoCommandType_CreateSpatialContext
- FdoCommandType_DestroySpatialContext
- FdoCommandType_GetSpatialContexts
- FdoCommandType_CreateDataStore
- FdoCommandType_DestroyDataStore
- FdoCommandType_ListDataStores
- FdoCommandType_DescribeSchemaMapping
- FdoCommandType_SQLCommand
- FdoRdbmsCommandType_CreateSpatialIndex
- FdoRdbmsCommandType_DestroySpatialIndex
- FdoRdbmsCommandType_GetSpatialIndexes

The following capabilities are supported:

- simple functions in Select and SelectAggregate commands
- use of expressions for properties in Select and SelectAggregates commands
- use of Distinct in SelectAggregates command
- availability of ordering in Select and SelectAggregates command
- availability of grouping criteria in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, spatial, and distance
- spatial operations of type `Intersects` and `EnvelopeIntersects`

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic, function, and parameter expressions are supported.

The following functions are supported:

- `Double Avg(<type> value)` where `<type>` is one of `Decimal`, `Double`, `Single`, `Int16`, `Int32`, or `Int64`.
- `Decimal Ceil(<type> value)` where `<type>` is one of `Decimal` or `Double`.
- `String Concat(String str1Val, String str2Val)`
- `Int64 Count(<type> value)` where `<type>` is one of `Boolean`, `Byte`, `DateTime`, `Decimal`, `Double`, `Int16`, `Int32`, `Int64`, `Single` or `String`
- `Decimal Floor(<type> value)` where `<type>` is one of `Decimal`, `Double`, or `Single`
- `String Lower(String value)`
- `Double Max(<type> value)` where `<type>` is one of `Byte`, `DateTime`, `Decimal`, `Double`, `Single`, `Int16`, `Int32`, `Int64`, `Single`, or `String`.
- `Byte Min(<type> value)` where `<type>` is one of `Byte`, `DateTime`, `Decimal`, `Double`, `Single`, `Int16`, `Int32`, `Int64`, `Single`, or `String`.
- `Double Sum(<type> value)` where `<type>` is one of `Decimal`, `Double`, `Int16`, `Int32`, `Int64`, or `Single`.
- `String Upper(String value)`

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XY is supported. The geometry component type LinearRing is supported. The following geometry types are supported.

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for ODBC



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for ODBC.

What Is FDO Provider for ODBC?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for ODBC provides FDO with access to an ODBC-based data store.

The FDO Provider for ODBC can access simple x, y, z feature objects that can run in a multi-platform environment, including Windows, Linux, and UNIX.

The FDO Provider for ODBC has the following characteristics:

- The FDO Provider for ODBC supports the definition of one or more feature classes in terms of any relational database table that contains an X, Y, and optionally, Z columns.
- Metadata, which maps the table name, and X, Y, and optionally, Z columns to a feature class, is maintained outside the database in a configuration file. This information, in conjunction with the table structure in the database, provides the definition of the feature class.
- The x, y, and z locations of objects are stored in separate properties in the primary object definition of a feature, but are accessible through a single class property 'Geometry'.
- Read-only access is provided to pre-existing data defined and populated through 3rd party applications (that is, FDO Provider for ODBC will not be

responsible for defining the physical schema of the data store nor for populating the object data).

- The schema configuration of the data store is provided to the FDO Provider for ODBC through an optional XML file containing the Geographic Markup Language (GML) definition of the schema that maps ‘tables’ and ‘columns’ in the data store to feature classes and property mappings in the FDO data model.

NOTE Microsoft Excel (must have at least one named range; do not use DATABASE or other reserved words as a range name).

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *OSGeo FDO Provider for ODBC API Reference Help* (ODBC_Provider_API.chm).

Physical to Logical Schema Mappings

Mappings of data in three data sources, a Microsoft Access database (an .accdb file), an Excel spreadsheet (.xlsx file), and a text file, to FDO data types are shown.

Microsoft Access

Use the ODBC provider to connect to a Microsoft Access database.

The schema name is “Fdo”. There is only one database per file so there can be only one schema. The table names are used for class names. Column names are used for property names.

Some native types are not mapped to FDO data types and so do not appear in the output of the describe schema command even if they exist in the table. They are AutoNumber/Replication ID, Number/Replication ID, and OLE object. The mapping of the rest of the native data types is in the following table.

NOTE The AutoNumber, Number, and Date/Time native types are sub-typed. If a column named X and a column named Y, both typed Number:Double, are present, they are mapped to an FDO geometric property whose type is Point and whose spatial context association property is “Default”.

| Native Type:SubType | FDO Type |
|-------------------------|----------|
| AutoNumber:Long Integer | Int32 |

| Native Type:SubType | FDO Type |
|---------------------------|-------------|
| AutoNumber:Replication ID | Not mapped. |
| Text | String |
| Memo | String |
| Number:Byte | Byte |
| Number:Integer | Int16 |
| Number:Long Integer | Int32 |
| Number:Single | Single |
| Number:Double | Double |
| Number:Decimal | Decimal |
| Number:Replication ID | Not mapped. |
| Date/Time:General Date | DateTime |
| Date/Time:Long Date | DateTime |
| Date/Time:Medium Date | DateTime |
| Date/Time:Short Date | DateTime |
| Date/Time:Long Time | DateTime |
| Date/Time:Medium Time | DateTime |
| Date/Time:Short Time | DateTime |
| Currency | Decimal |
| Yes/No | Boolean |

| Native Type:SubType | FDO Type |
|---------------------|-------------|
| OLE object | Not mapped. |
| Hyperlink | String |
| Attachment | String |

Microsoft Excel

Use the ODBC provider to connect to an Excel file.

The schema name is “Fdo”. There is only one schema per file. The range names are used for class names. Column names are used for property names. A column name is a text entry in a cell that is at the top of a column as defined by the boundaries of the named range.

If a column named X and a column named Y are present, they are mapped to an FDO geometric property whose type is Point and whose spatial context association property is “Default”.

Three approaches were used to explore the mapping. The first approach looked at the effect of cell content; the second approach looked at the effect of cell format; the third approach looked at the effect of both.

The first approach involved copying data from a Microsoft Access database into an Excel spreadsheet. The copy operation created a named range that included all of the copied data. The name for the range came from the Access database. By default all of the Excel cells have a format type of “General”. The accompanying table shows the mapping of sample cell content to FDO data type.

| Cell Content | FDO Type |
|--------------|----------|
| 2 | Double |
| 64.52067 | Double |
| Russia | String |
| <empty> | String |
| N | String |

NOTE All of the data in each column was consistent. The column containing “2” contained only positive integers. The column containing “64.52067” contained only similar numbers including negative values. The column containing the ‘N’ contained only single characters that were either ‘N’ or ‘Y’. The column containing “Russia” contained only names. The column containing an empty cell contained only empty cells.

The second approach involved setting up a named range consisting of 9 columns by 2 rows. The cells of the first row are formatted as “General” and contain column labels. The cells of the second row are formatted respectively “General,” “Number,” “Currency,” “Accounting,” “Date,” “Time,” “Percentage,” “Fraction,” “Scientific” and “Text”. No cell in the second row has content. The accompanying table shows the mapping of cell format to FDO data type.

| Excel Cell Format | FDO Type |
|-------------------|----------|
| General | String |
| Number | Double |
| Currency | Decimal |
| Accounting | Decimal |
| Date | DateTime |
| Time | DateTime |
| Percentage | Double |
| Fraction | Double |
| Scientific | Double |
| Text | String |

The intent of the third approach is to understand how the mapping would be affected by content that violated the cell’s format or was inconsistent with the data in the rest of the column.

Inserting “here” in the cells formatted as Number, Currency, Accounting, Date, Time, Percentage, Fraction and Scientific caused them to be mapped to String.

Inserting -12.25 in the cell formatted as Currency caused no change in the mapping to Decimal.

Inserting 12.25 in the cell formatted as Date caused no change in the mapping to DateTime. Excel converted 12.25 to "1/12/1900 6:00:00 AM".

Inserting "here" in one of the cells in the column of integers copied from the Access database resulted in no change to the mapping to Double. This may be a bug.

Text File

Use the ODBC provider to connect to a text file. You must first create a DSN using the XP Data Sources (ODBC) administrative tool. The DSN maps to a directory, not a file. Each file in the directory is mapped to an FDO class.

NOTE When selecting the directory for the DSN, you will notice a listing of the files recognized as text files by the driver. It will recognize Vim backup files, for example, Cities.txt~, as text files. The execution of a describe schema command on a directory containing a Vim backup file caused an exception whose message is "RDBMS: [Microsoft][ODBC Text Driver] Cannot update. Database or object is read-only.". Removal of the Vim backup file prevented this exception.

The schema name is "Fdo". There is only one schema per directory. The class name is the name of the file, for example, "Cities_txt". The property names are taken from the first line in the file. Here are the first two lines from a comma-delimited text file. The property names are "CITY_ID", "NAME", etc. The table shows the mapping of the sample text to FDO data type.

```
"CITY_ID", "NAME", "COUNTRY", "POPULATION", "CAPITAL", "LATITUDE", "LONGITUDE", "URL", "FOUNDED"
1, "Murmansk", "Russia", 468000, "N", 68.96, 33.08, , 7/17/2008 0:00:00
```

| Sample Text | FDO Data Type |
|-------------|---------------|
| 468000 | Int32 |
| Russia | String |
| 68.96 | Double |
| <empty> | String |

NOTE Changing “LATITUDE” to “Y” and “LONGITUDE” to “X” causes these two fields to be mapped to a geometric property called “Geometry” whose spatial context association is to a spatial context called “Default”.

FDO Provider for ODBC Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- SQL
- XML configuration

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning

of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of 1 byte
- Byte data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of 56 digits (maximum decimal precision of 28 and maximum decimal scale of 28)
- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Int64 data type with a maximum length of 8 bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of unknown
- Auto-generated data types Int16 and Int64
- Identity properties of type Boolean, Byte, and DateTime.
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: `::`
- Auto ID generation

- Composite ID
- Default value
- Inheritance
- Multiple schemas
- Null value constraints
- Schema overrides

Command Capabilities

Use the `Fdo ICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `Fdo IConnection` object. For an explanation of the meaning of the capabilities, consult the `Fdo ICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `Fdo CommandType_Select`
- `Fdo CommandType_SelectAggregates`
- `Fdo CommandType_DescribeSchema`
- `Fdo CommandType_DescribeSchemaMapping`
- `Fdo CommandType_Insert`
- `Fdo CommandType_Delete`
- `Fdo CommandType_Update`
- `Fdo CommandType_GetSpatialContexts`
- `Fdo CommandType_SQLCommand`

The following capabilities are supported:

- simple functions in `Select` and `SelectAggregates` commands
- use of expressions for properties in `Select` and `SelectAggregates` commands
- use of `Distinct` in `SelectAggregates` command
- availability of ordering in `Select` and `SelectAggregates` command

- availability of grouping criteria in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, and spatial.
- spatial operations of type Intersects, Within, Inside, and EnvelopeIntersects.

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic and function expressions are supported.

The full range of expression functions are supported as set out in the Capabilities chapter.

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZ is supported. The Point geometry types is supported.

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

Autodesk FDO Provider for Raster



This appendix discusses FDO API development issues that are related to Autodesk FDO Provider for Raster.

What Is FDO Provider for Raster?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The Autodesk FDO Provider for Raster is a stand-alone file format that supports GIS data.

The FDO Provider for Raster has the following characteristics:

- The FDO Provider for Raster supports georeferenced file-based raster images and file-based grid coverages. Raster images are pixel-based images, such as digital photographs (satellite images, for example). Raster images are very useful as background images underneath your vector data, for example, an aerial photograph of a city with a layer of streets overlaying it.
- The FDO Provider for Raster can run in a multi-platform environment, including Windows and Linux.

Supported Formats

The following list shows the raster image file formats that are supported, along with their acronyms and file extensions:

- JPEG (.jpg, .jpeg) - Joint Photographic Experts Group
- JPG2K (.jp2, .j2k) - Joint Photographic Experts Group

- MrSID (.sid) - Multi-Resolution Seamless Image Database
- PNG (.png) - Portable Network Graphic
- TIFF (.tif, .tiff) - Tagged Image File Format
- DEM (.dem) - USGS Format Digital Elevation Model
- ECW (.ecw) - Enhanced Compressed Wavelet
- DTED (.dt0, .dt1, dt2) - Digital Terrain Elevation Data
- ESRI ASCII GRID (.asc) - ESRI Surface
- ESRI Binary GRID (.adf) - ESRI Surface

Supported Data Models

The following are the data models supported:

| ModelType | BitsPerPixel | Organization | DataType |
|-----------|--------------|--------------|------------------|
| Bitonal | 1 | Pixel | Unsigned Integer |
| Grey | 8 | Pixel | Unsigned Integer |
| RGB | 24 | Pixel | Unsigned Integer |
| RGBA | 32 | Pixel | Unsigned Integer |
| Pallete | 8 | Pixel | Unsigned Integer |
| Data | 1 | Pixel | Unsigned Integer |
| Data | 8 | Pixel | Unsigned Integer |
| Data | 8 | Pixel | Signed Integer |
| Data | 16 | Pixel | Unsigned Integer |
| Data | 16 | Pixel | Signed Integer |
| Data | 32 | Pixel | Unsigned Integer |
| Data | 32 | Pixel | Signed Integer |

| ModelType | BitsPerPixel | Organization | DataType |
|-----------|--------------|--------------|----------|
| Data | 32 | Pixel | Float |

NOTE Only DEM, TIFF, and ECW images support the 'Data' ModelType.

NOTE All 2- and 4-BitsPerPixel images are promoted to 8 BitsPerPixel as per the underlying ATIL behavior.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *Autodesk FDO Provider for Raster API Reference Help* (Raster_Provider_API.chm).

FDO Provider for Raster Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Single threaded
- static spatial content extent type
- XML configuration

- multiple spatial contexts

Schema Capabilities

Use the `FdoSchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoSchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- feature class class type
- String data type with a maximum length of unknown bytes
- BLOB data type with a maximum length of unknown bytes
- Identity properties of type String
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: (null)
- Inheritance
- Multiple schemas
- Schema overrides

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning

of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_DescribeSchemaMapping`
- `FdoCommandType_GetSpatialContexts`
- `FdoCommandType_GetMeasureUnits`

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type in and spatial
- the Within distance operation
- spatial operations of type Within, Inside, Intersects, EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Function expressions are supported.

The following functions are supported:

- `BLOB MOSAIC(BLOB raster)`
- `BLOB CLIP(BLOB raster, Double minX, Double minY, Double maxX, Double maxY)`

- BLOB RESAMPLE(BLOB raster, Double minX, Double minY, Double maxX, Double maxY, Int32 height, Int32 width)
- GeometricProperty SpatialExtents(BLOB raster)

Geometry Capabilities

Use the `FdoGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XY is supported.

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

Supports raster, stitching, and subsampling. Supports the following raster data models:

- Bitonal/1-bit/pixel/Unsigned Integer
- Gray/8-bit/pixel/Unsigned Integer
- RGB/24-bit/pixel/Unsigned Integer
- RGBA/32-bit/pixel/Unsigned Integer
- Palette/8-bit/pixel/Unsigned Integer
- Data/16-bit/pixel/Unsigned Integer
- Data/16-bit/pixel/Signed Integer
- Data/32-bit/pixel/Unsigned Integer
- Data/32-bit/pixel/Signed Integer
- Data/32-bit/pixel/Float

OSGeo FDO Provider for SDF



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for SDF.

What Is FDO Provider for SDF?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for SDF is a standalone file format that supports GIS data.

The FDO Provider for SDF uses Autodesk's spatial database format, which is a file-based personal geodatabase that supports multiple features/attributes, spatial indexing, interoperability, file-locking, and high performance for large data sets.

The SDF file format has the following characteristics:

- SDF files can be read on different platforms.
- The SDF file has its own spatial indexing.
- SDF files can store geometric and non-geometric data with minimum overhead.
- Although it does not support concurrency control (locking), the SDF file format is a valid alternative to RDBMS.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *OSGeo FDO Provider for SDF API Reference Help* (SDF_Provider_API.chm).

FDO Provider for SDF Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per command threaded
- dynamic spatial content extent type
- Write
- Flush

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types

- Boolean data type with a maximum length of 1 byte
- Byte data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of unknown digits (maximum decimal precision of unknown and maximum decimal scale of unknown)
- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Int64 data type with a maximum length of 8 bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of unknown
- Int32 auto-generated data type
- Identity properties of type Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: `::`
- Association properties
- Auto ID generation
- Composite ID
- Exclusive value range constraints

- Inclusive value range constraints
- Inheritance
- Null value constraints
- Schema modification
- Value constraints list

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_Insert`
- `FdoCommandType_Delete`
- `FdoCommandType_Update`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_ApplySchema`
- `FdoCommandType_CreateSpatialContext`
- `FdoCommandType_GetSpatialContexts`
- `FdoCommandType_CreateDataStore`
- `FdoCommandType_DestroyDataStore`
- `SdfCommandType_ExtendedSelect`
- `SdfCommandType_CreateSDFFile`

The following capabilities are supported:

- simple functions in `Select` and `SelectAggregate` commands

- use of expressions for properties in Select and SelectAggregates commands
- use of Distinct in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, and spatial.
- spatial operations of type Contains, Disjoint, Inside, Intersects, Within, and EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic and function expressions are supported.

The following functions are supported:

- `GeometricProperty SpatialExtents(GeometricProperty geomValue)`
- `Double Avg(<type> value)` where <type> is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single.
- `Double Ceil(<type> value)` where <type> is one of Decimal, Double, or Single
- `String Concat(String str1Val, String str2Val)`
- `Int64 Count(<type> value)` where <type> is one of Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, String, BLOB, CLOB, GeometricProperty, AssociationProperty
- `Double Floor(<type> value)` where <type> is one of Decimal, Double, or Single

- String Lower(String value)
- Double Max(<type> value) where <type> is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single.
- Byte Min(<type> value) where <type> is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single.
- Double Sum(<type> value) where <type> is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single.
- String Upper(String value)

Geometry Capabilities

Use the `FdoGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types `Ring`, `LinearRing`, `CircularArcSegment`, and `LineStringSegment` are supported. The following geometry types are supported.

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- MultiGeometry
- CurveString
- CurvePolygon
- MultiCurveString
- MultiCurvePolygon

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for SHP



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for SHP.

What Is FDO Provider for SHP?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for SHP provides FDO with access to an SHP-based data store.

The FDO Provider for SHP uses a standalone file format that supports GIS data. The FDO Provider for SHP (Shape) has the following characteristics:

- Read-only access is provided to pre-existing spatial and attribute data from an Environmental Systems Research Institute (ESRI) Shape file (SHP).
- The FDO Provider for SHP can run in a multi-platform environment, including Windows and Linux.
- A Shape file consists of three separate files: SHP (shape geometry), SHX (shape index), and DBF (shape attributes in dBASE format).
- The FDO Provider for SHP accesses the information in each of the three separate files, and treats each SHP, and its associated DBF file, as a feature class with a single geometry property, and optionally, with data attribute properties.
- Schema configuration of the data store is provided to the FDO Provider for SHP through an XML file containing the Geographic Markup Language

(GML) definition of the schema that maps SHP and DBF data in the data store to feature classes and property mappings in the FDO data model.

- Although it does not support concurrency control (locking), the SHP file format is a valid alternative to RDBMS.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *OSGeo FDO Provider for SHP API Reference Help* (SHP_Provider_API.chm).

Creating SHP Files for the Logical to Physical Schema Mapping

See <http://shapelib.maptools.org/>. The shape files used in this exercise were created using a .NET wrapper that you can obtain from the ShapeLib site. The .NET wrapper package contains test code that was easily adapted to the purpose of creating shape files for the schema mapping exercise.

The shape files used for the purpose of mapping FDO logical types to shape native types were created using the following method. The path argument is the path to a filename whose pattern is <base>.shp. The type argument specifies the type of the shape file's geometry, for example, `ShapLib.ShapeType.Point`. Execution of the method creates the named <base>.shp file as well as a <base>.shx file.

```
using MapTools;
public bool CreateShpFile(string path, ShapeLib.ShapeType type)
{
    IntPtr hShp = ShapeLib.SHPCreate(path, type);
    if (hShp.Equals(IntPtr.Zero)) return false;
    else
    {
        ShapeLib.SHPClose(hShp);
        return true;
    }
}
```

The FDO `ApplySchema` creates the <base>.dbf and <base>.cpg. It will also create the <base>.prj file if a spatial context has been created and the name of the coordinate system referenced by the spatial context definition has been assigned to the feature geometric property definition's spatial context association property.

Creating SHP Files for the Physical to Logical Schema Mapping

To do the physical to logical schema mapping requires shp files containing all possible shp data types . Two approaches for creating such shp files have been used. The first approach is to use the ArcSDE sde2shp command to convert an ArcSDE schema to a shapefile schema. The second approach is to use the ShapeLib C library to create the shapefiles .

sde2shp

This command converts the contents of an ArcSDE data store into a set of shape files. The conversion is guided by a text specification that maps an ArcSDE column name to a tuple consisting of a shape table column name, a shape column type identifier, a column width, and optionally the number of decimal places.

The source ArcSDE data store for this exercise is that specified in the ArcSDE appendix in the topic about physical to logical schema mapping. The data store contained a table called “ExistingTable”. This table containing a geometry column called “featgeom”. The featgeom column allows only point geometries to be stored. A shape geometry can only be of one type. So, for the conversion to work, the ArcSDE geometry column has to be restricted to contain only one geometry type. The source ArcSDE data store contained no data. The ArcSDE geometry was created with a coordinate system specified.

The syntax of the command follows: `sde2shp -o init -l ExistingTable,featgeom -f points -a file=points.txt -t point -i esri_sde_ss -u sde -p fdotest`. The `-f` argument becomes the root name of the shape files that are created by the command (points.shp, points.shx, points.dbf, points.idx, and points.prj). The `-t` argument is the shape geometry type. The `-i` argument is the name of the ArcSDE server instance; the symbolic name is mapped to a TCP port in the `C:\WINDOWS\system32\drivers\etc\services` file. The `-u` and `-p` arguments identify a username and password in the data store. The `-a` argument identifies the file used to specify the conversion of the non-geometry column conversions.

The contents of the points.txt conversion file are as follows:

```

smInt4Col smInt4 N 4
int16Col int16Col N 4
int32Col int32Col N 10
float32Col float32C N 6 2
float64Col float64C N 15 4
int10 Int10 N 10
float4_4 float4_4 N 4 3
double7_4 doubl7_2 N 7 2
string64 str64 C 64
nStringCol nStrCol C 254
dateCol dateCol D 8

```

NOTE Initial values for column widths and number of decimal places were taken from the output of the `sddtable -o describe` command. These values were changed over the course of several executions of the `sde2shp` in response to warning and error messages issued by the `sde2shp` command.

ShapeLib

See <http://shapelib.maptools.org/>. The shape files used in this exercise were created using a .NET wrapper that you can obtain from the ShapeLib site. The .NET wrapper package contains test code that was easily adapted to the purpose of creating shape files for the schema mapping exercise.

The `<base>.shp` and `<base>.shx` files are created by the `ShapeLib.SHPCreate` method.

The `<base>.prj` file is created by using a text editor to create the file and inserting the WKT string specification for the coordinate system.

The `<base>.dbf` file is created by the following code:

```

IntPtr dbfFile = MapTools.ShapeLib.DBFCre
ate("../Points/points.dbf");
int logicalColNum = MapTools.ShapeLib.DBFAAddField(dbfFile, "Logic
alCol", ShapeLib.DBFFieldType.FTLogical, 1, 0);
int strColNum = MapTools.ShapeLib.DBFAAddField(dbfFile, "StrCol",
ShapeLib.DBFFieldType.FTString, 64, 0);
int dateColNum = MapTools.ShapeLib.DBFAAddField(dbfFile, "DateCol",
ShapeLib.DBFFieldType.FTDate, 8, 0);
int integer20_0Num = MapTools.ShapeLib.DBFAAddField(dbfFile, "In
teger20", ShapeLib.DBFFieldType.FTInteger, 20, 0);
int double18_4Num = MapTools.ShapeLib.DBFAAddField(dbfFile,
"Double18_4", ShapeLib.DBFFieldType.FTDouble, 18, 4);
MapTools.ShapeLib.DBFClose(dbfFile);

```

Logical to Physical Schema Mapping

The SHP provider supports the DestroySchema and ApplySchema commands. So it is possible to create shape files whose schema has been defined using FDO and then map the logical FDO data types to the shape native types. This mapping is described here. The creation of the shape file used for this purpose is described in this appendix.

| FDO Type | Native Type, Column Width, Decimal Places |
|----------|---|
| Boolean | L, 1 |
| DateTime | D, 8 |
| Decimal | N, >0, >0 |
| Int32 | N, 11 |
| String | C, >0 |

Physical to Logical Schema Mapping

When FDO describes the schema of an existing shapefile that was not created by the ApplySchema command, it maps native data types to FDO data types. Two shapfiles have been mapped. One shapfile has been created using the ArcSDE command sde2shp. The creation of the source ArcSDE schema is described in the appendix for the ArcSDE provider. The other shapfile has been created using a .NET wrapper for the ShapeLib C Library. These two methods for creating shapefiles are described in this appendix.

The schema name is "Default". The class name is the root name of the file. For example, if the filename is "points.shp", the class name is "points". The attribute property names are the dBASE column names. The geometry property name is "Geometry". The value of the feature class's SpatialContextAssociation property is the name of the coordinate system specified in the .prj file. For example, if the coordinate system specification in the .prj has the pattern "PROJCS[...]", the value of the SpatialContextAssociation property is the value of the <name> parameter in the pattern "PROJCS[<name>...]".

sde2shp

The mapping of SHP (.dbf) native types to FDO data types for the SHP schema that came from the conversion of an ArcSDE schema is in the following table.

| Native Type, Column Width, Decimal Places | FDO Data Type |
|---|---------------|
| N, 4 | Decimal |
| N, 10 | Decimal |
| N, 6, 2 | Decimal |
| N, 15, 4 | Decimal |
| N, 4, 3 | Decimal |
| N, 7, 2 | Decimal |
| C | String |
| D | DateTime |

ShapeLib

The mapping of SHP (.dbf) native types to FDO data types for the SHP schema that was created by a ShapeLib program is in the following table.

| Native Type, Column Width, Decimal Places | FDO Data Type |
|---|---------------|
| L, 1 | Boolean |
| C, 64 | String |
| D, 8 | DateTime |
| N, 20, 0 | Decimal |
| N, 18. 4 | Decimal |

FDO Provider for SHP Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- XML configuration
- multiple spatial contexts
- Write
- Multi-user write
- Flush

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of 255 digits (maximum decimal precision of 255 and maximum decimal scale of 255)
- Int32 data type with a maximum length of 4 bytes
- String data type with a maximum length of 255
- Int32 auto-generated data type
- Identity properties of type Int32
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of 7 for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of 255 for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of 11 for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of 0 for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: `::`
- Auto ID generation
- Multiple schemas
- Null value constraints
- Schema modification
- Schema overrides

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()`

method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_Insert`
- `FdoCommandType_Delete`
- `FdoCommandType_Update`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_DescribeSchemaMapping`
- `FdoCommandType_ApplySchema`
- `FdoCommandType_DestroySchema`
- `FdoCommandType_CreateSpatialContext`
- `FdoCommandType_GetSpatialContexts`
- `SdfCommandType_CreateSDFFile`

The following capabilities are supported:

- simple functions in `Select` and `SelectAggregate` commands
- use of expressions for properties in `Select` and `SelectAggregates` commands
- use of `Distinct` in `SelectAggregates` command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, and spatial

- spatial operations of type Within, Inside, Intersects, EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic and function expressions are supported.

The following functions are supported:

- `Double Avg(<type> value)` where `<type>` is one of Byte, Decimal, Double, Single, Int16, Int32, or Int64.
- `Double Ceil(<type> value)` where `<type>` is one of Decimal, Double, or Single
- `String Concat(String str1Val, String str2Val)`
- `Int64 Count(<type> value)` where `<type>` is one of Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, String, BLOB, CLOB, ObjectProperty, GeometricProperty, AssociationProperty, or RasterProperty
- `Decimal Floor(<type> value)` where `<type>` is one of Decimal, Double, or Single
- `String Lower(String value)`
- `Double Max(<type> value)` where `<type>` is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single
- `Byte Min(<type> value)` where `<type>` is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single
- `Double Sum(<type> value)` where `<type>` is one of Byte, Decimal, Double, Int16, Int32, Int64, or Single.
- `String Upper(String value)`
- `GeometricProperty SpatialContexts(GeometricProperty property)`

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()`

method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types `LinearRing` and `LineStringSegment` are supported. The following geometry types are supported.

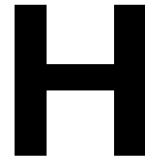
- `Point`
- `LineString`
- `Polygon`
- `MultiPoint`
- `MultiLineString`

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

Autodesk FDO Provider for SQL Server



This appendix discusses FDO API development issues that are related to Autodesk FDO Provider for SQL Server.

What Is FDO Provider for SQL Server?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for SQL Server provides FDO with access to a Microsoft SQL Server-based data store.

The Autodesk FDO Provider for SQL Server API provides custom commands that are specifically designed to work with the FDO API. For example, using these commands, you can do the following:

- Read and create schema.
- Read and write geospatial and non-geospatial data.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *Autodesk FDO Provider for SQL Server API Reference Help* (SQLServer_Provider_API.chm).

Logical to Physical Schema Mapping

The FDO ApplySchema command creates a table for each class in the command's class collection. The mapping of logical FDO data types to native types is described here.

NOTE The definition of the FDO Decimal data type requires that the precision property be set as a minimum to 1; in this case the scale property is set to 0 by default. Precision is the number of digits in the number, and scale is the number of digits to the right of the decimal point. The maximum number of digits in the native type is 64 for version 5.0.3 or higher.

NOTE The definition of the FDO String data type requires that the length property be set.

| FDO Data Type | Native Type |
|---|---------------|
| Boolean | tinyint |
| Byte | tinyint |
| DateTime | datetime |
| Decimal (precision property is set to 20, and scale property is set to 4) | decimal(20,4) |
| Double | float |
| Int16 | smallint |
| Int32 | int |
| Int64 | bigint |
| Single | real |
| String (length property set to 64) | nvarchar(64) |
| geometry | image |

Physical to Logical Schema Mapping

When FDO describes the schema of an existing table that was not created by the ApplySchema command, it maps native data types to FDO data types. This mapping is described here.

The feature schema name is 'dbo'. The table names are used for class names. Column names are used for property names.

The xml native type is not mapped to FDO data types and does not appear in the output of the describe schema command even if it exists in the table. The mapping of the rest of the native data types is in the following table.

NOTE The presence of native types varchar(MAX) and nvarchar(MAX) causes an exception when the DescribeSchema command is executed. The workarounds are the following:

- Delete the column
- Move the table to another datastore and add a view on the table that excludes the column in the original datastore.
- Create another datastore and add views for the table in the original datastore, making sure to exclude the column.

| Native Type | FDO Type |
|---------------|--|
| bigint | Int64 |
| binary(50) | BLOB |
| bit | Boolean |
| char(10) | String with length of 10 |
| datetime | DateTime |
| decimal(18,0) | Decimal (precision = 18 and scale = 0) |
| float | Double |
| image | BLOB |
| int | Int32 |
| money | Decimal (precision = 19 and scale = 4) |
| nchar(10) | String with length of 10 |
| ntext | String with length of 1073741823 |
| numeric(18,0) | Decimal (precision = 18 and scale = 0) |

| Native Type | FDO Type |
|------------------|--|
| nvarchar(50) | String with length of 50 |
| nvarchar(MAX) | Cause an exception in DescribeSchema |
| real | Single |
| smalldatetime | DateTime |
| smallint | Int16 |
| smallmoney | Decimal (precision = 10 and scale = 4) |
| text | String with length of 2147483647 |
| timestamp | DateTime |
| tinyint | Byte |
| uniqueidentifier | String with length of 36 |
| varbinary(50) | BLOB |
| varchar(50) | String with length of 50 |
| varchar(MAX) | Causes an exception in DescribeSchema |
| varbinary(MAX) | BLOB |

FDO Provider for SQL Server Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters

- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threaded
- static spatial content extent type
- transactions
- SQL
- multiple spatial contexts
- Write
- Multi-user write

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of 1 byte
- Byte data type with a maximum length of 1 byte
- DateTime data type with a maximum length of 12 bytes
- Decimal data type with a maximum length of 76 digits (maximum decimal precision of 38 and maximum decimal scale of 38)

- Double data type with a maximum length of 8 bytes
- Int16 data type with a maximum length of 2 bytes
- Int32 data type with a maximum length of 4 bytes
- Int64 data type with a maximum length of 8bytes
- Single data type with a maximum length of 4 bytes
- String data type with a maximum length of 2147483647
- Int64 auto-generated data type
- Identity properties of type Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String
- Name size limitation of 123 for a schema element name of type FdoSchemaElementType_Datastore
- Name size limitation of 200 for a schema element name of type FdoSchemaElementType_Schema
- Name size limitation of 200 for a schema element name of type FdoSchemaElementType_Class
- Name size limitation of 255 for a schema element name of type FdoSchemaElementType_Property
- Name size limitation of 255 for a schema element name of type FdoSchemaElementType_Description
- Characters that cannot be used for a schema element name: .:
- Association properties
- Auto ID generation
- Composite ID
- Composite unique value constraints
- Datastore scope unique ID generation
- Default value
- Exclusive value range constraints
- Inclusive value range constraints
- Inheritance

- Multiple schemas
- Null value constraints
- Object properties
- Unique value constraints
- Schema modification
- Schema overrides
- Unique value constraints
- Value constraints list

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_Insert`
- `FdoCommandType_Delete`
- `FdoCommandType_Update`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_DescribeSchemaMapping`
- `FdoCommandType_ApplySchema`
- `FdoCommandType_DestroySchema`
- `FdoCommandType_ActivateSpatialContext`
- `FdoCommandType_CreateSpatialContext`
- `FdoCommandType_DestroySpatialContext`
- `FdoCommandType_GetSpatialContexts`

- FdoCommandType_CreateDataStore
- FdoCommandType_DestroyDataStore
- FdoCommandType_ListDataStores
- FdoCommandType_SQLCommand
- FdoRdbmsCommandType_CreateSpatialIndex
- FdoRdbmsCommandType_DestroySpatialIndex
- FdoRdbmsCommandType_GetSpatialIndexes

The following capabilities are supported:

- simple functions in Select and SelectAggregate commands
- use of expressions for properties in Select and SelectAggregates commands
- use of Distinct in SelectAggregates command
- availability of ordering in Select and SelectAggregates command
- availability of grouping criteria in SelectAggregates command

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Conditions of type comparison, like, in, null, spatial, and distance
- spatial operations of type Inside, Intersects, EnvelopeIntersects

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic, function, and parameter expressions are supported.

The following functions are supported:

- Double Avg(<type> value) where <type> is one of Decimal, Double, Single, Int16, Int32, or Int64.
- Decimal Ceil(<type> value) where <type> is one of Decimal, Double, or Single
- String Concat(String str1Val, String str2Val)
- Int64 Count(<type> value) where <type> is one of Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String
- Decimal Floor(<type> value) where <type> is one of Decimal, Double, or Single
- String Lower(String value)
- Byte Max(<type> value) where <type> is one of Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String.
- Byte Min(<type> value) where <type> is one of Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, or String.
- Double Sum(<type> value) where <type> is one of Decimal, Double, Int16, Int32, Int64, or Single.
- String Upper(String value)

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types `Ring`, `LinearRing`, `CircularArcSegment`, and `LineStringSegment` are supported. The following geometry types are supported.

- `Point`
- `LineString`
- `Polygon`
- `MultiPoint`

- MultiLineString
- MultiPolygon
- CurveString
- CurvePolygon
- MultiCurveString
- MultiCurvePolygon

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for SQL Server Spatial



Logical to Physical Schema Mapping

The FDO ApplySchema command creates a table for each class in the command's class collection. The mapping of logical FDO data types to native types is described here.

NOTE The definition of the FDO Decimal data type requires that the precision property be set as a minimum to 1; in this case the scale property is set to 0 by default. Precision is the number of digits in the number, and scale is the number of digits to the right of the decimal point. The maximum number of digits in the native type is 38.

NOTE The definition of the FDO String data type requires that the length property be set.

| FDO Data Type | Native Type |
|---|---------------|
| Boolean | bit |
| Byte | tinyint |
| DateTime | datetime |
| Decimal (precision property is set to 20, and scale property is set to 4) | decimal(20,4) |
| Double | float |

| FDO Data Type | Native Type |
|---|--------------|
| Int16 | smallint |
| Int32 | int |
| Int64 | bigint |
| Single | real |
| String (length property set to 64) | nvarchar(64) |
| geometry with a geodetic coordinate system | geography |
| geometry with a Euclidean coordinate system | geometry |

Physical to Logical Schema Mapping

When FDO describes the schema of an existing table that was not created by the ApplySchema command, it maps native data types to FDO data types. This mapping is described here.

The database name is used for the feature schema name. The table names are used for class names. Column names are used for property names.

Some native types are not mapped to FDO data types and so do not appear in the output of the describe schema command even if they exist in the table. They are date, datetim2(7), datetimeoffset(7), hierarchyid, sql_variatn, time(7), and xml. The mapping of the rest of the native data types is in the following table.

NOTE The geography and geometric native types map to the geometric property type. FDO properties are classified as data, geometric, object, association, or raster. Only the data properties are further classified.

| Native Type | FDO Type |
|-------------|----------|
| bigint | Int64 |
| binary(50) | BLOB |

| Native Type | FDO Type |
|---------------|-----------|
| bit | Boolean |
| char(10) | String |
| datetime | DateTime |
| decimal(18,0) | Decimal; |
| float | Double |
| geography | Geometric |
| geometry | Geometric |
| image | BLOB |
| int | Int32 |
| money | Decimal |
| nchar(10) | String |
| ntext | String |
| numeric(18,0) | Decimal |
| nvarchar(50) | String |
| nvarchar(MAX) | String |
| real | Single |
| smalldatetime | DateTime |
| smallint | Int16 |
| smallmoney | Decimal |
| text | String |
| timestamp | DateTime |

| Native Type | FDO Type |
|------------------|----------|
| tinyint | Byte |
| uniqueidentifier | String |
| varbinary(50) | BLOB |
| varchar(50) | String |
| varbinary(MAX) | BLOB |

OSGeo FDO Provider for WFS



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for WFS.

What Is FDO Provider for WFS?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for WFS provides FDO with access to a WFS-based data store.

An OGC Web Feature Service (WFS) provides access to geographic features that are stored in an opaque data store in a client/server environment. A client uses WFS to retrieve geospatial data that is encoded in Geography Markup Language (GML) from a single or multiple Web Feature Service. The communication between client and server is encoded in XML. If the WFS response includes feature geometries, it is encoded in Geography Markup Language (GML), which is specified in the OpenGIS Geographic Markup Language Implementation Specification.

Using FDO Provider for WFS data manipulation operations, you can do the following:

- Query features based on spatial and non-spatial constraints.
- Create new feature instances.
- Delete feature instances.
- Update feature instances.

- Lock feature instances.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf).

NOTE There is no public API documentation for the FDO Provider for WFS; functionality is available through the main FDO API.

FDO Provider for WFS Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threaded
- static spatial content extent type
- multiple spatial contexts

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning

of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- Boolean data type with a maximum length of unknown bytes
- Byte data type with a maximum length of unknown bytes
- DateTime data type with a maximum length of unknown bytes
- Decimal data type with a maximum length of unknown digits (maximum decimal precision of unknown and maximum decimal scale of unknown)
- Double data type with a maximum length of unknown bytes
- Int16 data type with a maximum length of unknown bytes
- Int32 data type with a maximum length of unknown bytes
- Int64 data type with a maximum length unknown 8 bytes
- Single data type with a maximum length of unknown bytes
- String data type with a maximum length of unknown
- Identity properties of type Boolean, Byte, DateTime, Decimal, Double, Int16, Int32, Int64, Single, and String
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Description`
- Characters that cannot be used for a schema element name: (null)
- Composite ID
- Multiple schemas

- Object properties

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`
- `FdoCommandType_DescribeSchema`
- `FdoCommandType_GetSpatialContexts`
- `FdoRdbmsCommandType_GetSpatialIndexes`

The following capabilities are supported:

- use of expressions for properties in `Select` and `SelectAggregates` commands

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

No capabilities are supported:

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Basic expressions are supported.

The following functions are supported:

- `GeometricProperty SpatialExtents(GeometricProperty property)`

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XYZM is supported. The geometry component types `Ring`, `LinearRing`, `CircularArcSegment`, and `LineStringSegment` are supported. The following geometry types are supported.

- `Point`
- `LineString`
- `Polygon`
- `MultiPoint`
- `MultiLineString`
- `MultiPolygon`
- `MultiGeoemtry`
- `CurveString`
- `CurvePolygon`
- `MultiCurveString`
- `MultiCurvePolygon`

Raster Capabilities

Use the `FdoIRasterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetRasterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIRasterCapabilities` class description in the FDO API Reference documentation.

No Raster capabilities are supported.

OSGeo FDO Provider for WMS



This appendix discusses FDO API development issues that are related to OSGeo FDO Provider for WMS.

What Is FDO Provider for WMS?

The Feature Data Objects (FDO) API provides access to data in a data store. A provider is a specific implementation of the FDO API that provides access to data in a particular data store. The FDO Provider for WMS provides FDO with access to a WMS-based data store.

An Open Geospatial Consortium (OGC) Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information. This international standard defines a "map" to be a portrayal of geographic information as a digital image file suitable for display on a computer screen. A map is not the data itself. Maps by WMS are generally rendered in a pictorial format, such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

The FDO Provider for WMS has the following characteristics:

- The FDO Provider for WMS serves up map information originating from an OGC Basic Web Map Service that provides pictorially formatted images, such as PNG, GIF, or JPEG.
- WMS map data is exposed through an FDO feature schema whose classes contain an FDO Raster property definition. The FDO schema exposed from the FDO Provider for WMS conforms to a pre-defined FDO schema that is

specific to WMS and that acts as the basis for all FDO interaction with WMS data, regardless of the originating source of the WMS images.

- WMS data manipulation operations are limited to querying features based on spatial and non-spatial constraints. Schema manipulation operations are not supported.

The FDO Provider for WMS can run in a multi-platform environment, including Windows and Linux.

For more information, see *The Essential FDO* (FET_TheEssentialFDO.pdf) and the *OSGeo FDO Provider for WMS API Reference Help* (WMS_Provider_API.chm).

FDO Provider for WMS Capabilities

The capabilities of an FDO provider are grouped in the following categories:

- Connection
- Schema
- Commands
- Expressions
- Filters
- Geometry
- Raster

Connection Capabilities

Use the `FdoIConnectionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetConnectionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIConnectionCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- Per connection threading
- static spatial content extent type
- XML configuration

Schema Capabilities

Use the `FdoISchemaCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetSchemaCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoISchemaCapabilities` class description in the FDO API Reference documentation.

The following capabilities are supported:

- class and feature class class types
- String data type with a maximum length of unknown
- BLOB data type with a maximum length of unknown bytes
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Datastore`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Schema`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Class`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Property`
- Name size limitation of unknown for a schema element name of type `FdoSchemaElementNameType_Description`
- Inheritance
- Schema overrides

Command Capabilities

Use the `FdoICommandCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetCommandCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoICommandCapabilities` class description in the FDO API Reference documentation.

The following commands are supported:

- `FdoCommandType_Select`
- `FdoCommandType_SelectAggregates`

- FdoCommandType_DescribeSchema
- FdoCommandType_DescribeSchemaMapping
- FdoCommandType_GetSpatialContexts

The following capabilities are supported:

- simple functions in Select and SelectAggregate commands

Filter Capabilities

Use the `FdoIFilterCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetFilterCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIFilterCapabilities` class description in the FDO API Reference documentation.

No filter capabilities are supported:

Expression Capabilities

Use the `FdoIExpressionCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetExpressionCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIExpressionCapabilities` class description in the FDO API Reference documentation.

Function expressions are supported.

The following functions are supported:

- BLOB RESAMPLE(BLOB raster, Double minX, Double minY, Double maxX, Double maxY, Int32 height, Int32 width)
- BLOB CLIP(BLOB raster, Double minX, Double minY, Double maxX, Double maxY)
- GeometricProperty SpatialExtents(BLOB raster)

Geometry Capabilities

Use the `FdoIGeometryCapabilities` object methods to learn about these capabilities. You can get this object by calling the `GetGeometryCapabilities()` method on the `FdoIConnection` object. For an explanation of the meaning of the capabilities, consult the `FdoIGeometryCapabilities` class description in the FDO API Reference documentation.

Dimensionality XY is supported. The geometry component type LinearRing is supported. The following geometry types are supported.

- Polygon

Raster Capabilities

Use the FdoIRasterCapabilities object methods to learn about these capabilities. You can get this object by calling the GetRasterCapabilities() method on the FdoIConnection object. For an explanation of the meaning of the capabilities, consult the FdoIRasterCapabilities class description in the FDO API Reference documentation.

The Raster capability is supported. The following raster data models are supported:

- Bitonal/1-bit/pixel/Unsigned Integer
- Gray/8-bit/pixel/Unsigned Integer
- RGB/24-bit/pixel/Unsigned Integer
- RGBA/32-bit/pixel/Unsigned Integer
- Palette/8-bit/pixel/Unsigned Integer
-

Expression Functions



Introduction

The enhanced set includes aggregate, conversion, date, mathematical, numeric, string and geometry functions. All functions are supported by all providers, with the exception of the Raster, WFS and WMS providers.

This appendix outlines the following expression functions details:

- Signatures
- Built-in support for some of the functions based on the supported RDBMS
- Provider-specific support for some of the functions
- Implementation options

NOTE For more information and details about expression functions, see <http://fdo.osgeo.org>.

Expression Function List

This section lists and describes all supported expression functions.

Aggregate Expression Functions

The following aggregate expression functions are supported:

| Function Name | Description |
|----------------|--|
| Avg | Returns the average of the values identified by the provided expression. |
| Count | Returns the number of rows returned by a query identified by the provided expression. |
| Max | Returns the maximum value of the provided expression. |
| Median | Represents an inverse distribution function that assumes a continuous distribution model. It takes a numeric or date-time value and returns the middle value or an interpolated value that would be the middle value once the values are sorted. |
| Min | Returns the minimum value of the provided expression. |
| Stddev | Returns the sample standard deviation of the provided expression. |
| Sum | Returns the sum of the values identified by the provided expression. |
| Spatial Extent | Returns the spatial extent of a geometry. |

Conversion Expression Functions

The following conversion expression functions are supported.

| Function Name | Description |
|---------------|--|
| NullValue | Evaluates two given expressions and returns the first one if it does not evaluate to NULL, the second otherwise. |
| ToDate | Converts a string with date/time information to a date. |
| ToDouble | Converts a numeric or string expression to a double. |
| ToFloat | Converts a numeric or string expression to a float. |
| ToInt32 | Converts a numeric or string expression to an int32. |
| ToInt64 | Converts a numeric or string expression to an int64. |

| Function Name | Description |
|---------------|--|
| ToString | Converts a numeric or date expression to a string. |

Date Expression Functions

The following date expression functions are supported:

| Function Name | Description |
|---------------|---|
| AddMonths | Adds a specified number of months to a given date expression. |
| CurrentDate | Returns the current date. |
| Extract | Extracts a specified portion of a date. |
| MonthsBetween | Calculates the number of months between two provide date expressions. |

Geometry Expression Functions

The following geometry expression functions are supported.

| Function Name | Description |
|---------------|-------------------------------|
| Area2D | Returns area of a geometry. |
| Length2D | Returns length of a geometry. |

Mathematical Expression Functions

The following mathematical expression functions are supported:

| Function Name | Description |
|---------------|---|
| Abs | Returns the absolute value of a numeric expression. |
| Acos | Returns the arc cosine of a numeric expression. |
| Asin | Returns the arc sine of a numeric expression. |
| Atan | Returns the arc tangent of a numeric expression. |

| Function Name | Description |
|---------------|---|
| Cos | Returns the cosine of a numeric expression. |
| Exp | Returns e raised to the power of a numeric expression. |
| Ln | Returns the natural logarithm of a numeric expression. |
| Log | Returns the logarithm of a numeric expression using the provided base. |
| Mod | Returns the remainder of the division of two numeric expressions. |
| Power | Returns the result of a numeric expression raised to the power of another numeric expression. |
| Remainder | Returns the remainder of the division of two numeric expressions. |
| Sin | Returns the sine of a numeric expression. |
| Sqrt | Returns the square root of a numeric expression. |
| Tan | Returns the tangent of a numeric expression. |
| | |
| | |
| | |

```

M N  Mod(M/N)  Classical Modulus
11 4    3      3
11 -4   3     -1
-11 4   -3      1
-11 -4  -3     -3

```

Figure 1: Modulus Implementation Results

Numeric Expression Functions

The following numeric expression functions are supported.

| Function Name | Description |
|---------------|---|
| Ceil | Returns the smallest integer greater than or equal to a numeric expression. |
| Floor | Returns the largest integer less than or equal to a numeric expression. |

| Function Name | Description |
|---------------|--|
| Round | Returns the rounded value of a numeric expression. |
| Sign | Returns -1 if the provided numeric expression evaluates to a value less than 0, 0 if the expression evaluates to 0 and 1 if the expression evaluates to a value bigger than 0. |
| Trunc | Truncates a numeric or date expression. |

String Expression Functions

The following string expression functions are supported:

The function **Translate** does not translate a string from one character set to a different one. Instead, it just replaces a set of letters in a string with their corresponding replacement characters where each character specified in the set of characters to be replaced is replaced by the character in the set of replacement characters at the same place. For example, if the call is **Translate('SQL*Plus User's Guide', ' */', '---')**, each of the specified characters in the first set will be replaced by the character in the same position in the second set. In this example, the resulting string will be **SQL_Plus_Users_Guide** because:

- Any space will be replaced by a _
- Any * will be replaced by a _
- Any / will be replaced by a _ (in this case, nothing will be done because the string does not include the specified character)
- Any ' will be removed because there is no corresponding replacement character specified.

| Function Name | Description |
|---------------|---|
| Concat | Returns the concatenation of two string expressions. |
| Instr | Returns the position of a substring in a string expression. |
| Length | Returns the length of a string expression. |
| Lower | Converts all uppercase letters in a string expression into lowercase letters. |

| Function Name | Description |
|---------------|---|
| Lpad | Pads a string expression to the left to a predefined string length. |
| Ltrim | Removes leading blanks from a string expression. |
| Rpad | Pads a string expression to the right to a predefined string length. |
| Rtrim | Removes trailing blanks from a string expression. |
| Soundex | Returns the phonetic representation of a string expression. |
| Substr | Extracts a substring from a string expression. |
| Translate | Replaces a set of letters in a string. |
| Trim | Removes leading and/or trailing blanks from a string expression. |
| Upper | Converts all lowercase letters in a string expression into uppercase letters. |

Index

A

- AGF 142
- API
 - defined 5
 - FDO 5
- application development 11
- architecture and packages 7
- ArcSDE
 - limitations 228
- ArcSDE Provider 227
 - Capabilities 243, 253, 265, 273, 278, 291, 300, 312, 318
 - Software Requirements 227
- association property 15
- Autodesk Geometry Format (AGF) 142

B

- behaviors, GisPtr 25
- BinaryOperations 136

C

- calls, chain 25
- capabilities
 - FDO Provider for ArcSDE 243, 253, 265, 273, 278, 291, 300, 312, 318
 - FDO Provider for Oracle 219
- class
 - contained 83
 - feature 14
 - IdentityProperty and ObjectProperty 83
 - standalone 83
- Class Diagram, FDO Schema Element 85
- class type 14
- collection classes 23
- comparison operations 136

- connection
 - ArcSDE 232
 - establishing 29
 - Oracle 200
 - semantics 27
- constraints, expression text 131
- constraints, filter text 131
- constraints, provider-specific 131
- Contained Class 83
- context, spatial 17

D

- data concepts 13
- data property 211
 - defined 15
 - overrides 210
- data sources and data stores 27
- data store 18
 - defined 18
 - FDO Provider for ArcSDE 227
 - FDO Provider for MySQL 249
 - FDO Provider for ODBC 259
 - FDO Provider for Oracle 199
 - FDO Provider for SDF 277
 - FDO Provider for SHP 285
 - FDO Provider for WMS 317
 - filtering 19, 129
 - locking 19
 - Oracle 9
 - querying 119
 - schemas and the 77
 - transactions 20
- data stores, data sources and 27
- data type
 - DATETIME 135
 - DOUBLE 135
 - IDENTIFIER 134
 - INTEGER 135
 - PARAMETER 134
 - STRING 134

data type mappings 232

data types 134

DataStore

FDO Provider for ArcSDE 227

decimal properties 212

develop applications 11

dimensionality, defined 15

E

edit a GML schema file 93

element states, schema 86

elements of a schema 14

example

creating a schema 103

creating a schema read in from an
XML file 106

deleting property values 117

describing a schema 105

destroying a schema 106

inserting an Integer, a string, and a
Geometry Value 112

query 120

schema management 103

updating property values 115

expression grammar 133

expression text 130

Expression, defined 19

expressions 130

F

factory, abstract geometry 149

FDO

architecture and packages 7

FDO API

defined 5

FDO concepts

commands 18

data store 18

expression 19

feature class 14

filter 19

geometry property 15

locking 19

object property 17

property 15

spatial context 17

transactions 19

FDO In General 151

FDO Lock Types, OWM and 217

FDO Provider for ArcSDE

capabilities 243, 253, 265, 273, 278,
291, 300, 312, 318

connection 232

defined 227

software requirements 227

FDO Provider for MySQL

defined 249

FDO Provider for ODBC

defined 259

FDO Provider for Oracle

capabilities 219

connection 200

defined 199

schema overrides 209

software requirements 200

FDO Provider for Raster

defined 271

FDO Provider for SDF 277

defined 277

FDO Provider for SDE, defined 277

FDO Provider for SHP

defined 285

FDO Provider for SQL Server

defined 297

FDO Provider for WMS

defined 317

FDO schema element class diagram 85

FDO XML format 87

FDOClass 82, 215

FDOClassDefinition 215

FDOFeatureClass 82

FDOIActivateLongTransaction 126

FDOICommitLongTransaction 127

FDOICreateLongTransaction 127

FDOIDeactivateLongTransaction 126

FDOIGetLongTransactions 127

FDOIRollbackLongTransaction 126

feature class 14

feature schema, creating a 233

- filter
 - grammar 131
- Filter 129
 - defined 19
- filter text 130
- filters 19
- foreign schema
 - limitations 202
 - read-write privileges 202
- foreign schemas
 - FDO Provider for Oracle 201

G

- geometric property 212
 - overrides 210
- geometric types, mapping between
 - Geometry and 150
- geometry 16
- Geometry
 - basic or pure 142
 - properties 15
 - types 149
 - value 137
 - working with 141
- geometry and geometric types, mapping
 - between 150
- Geometry API 141
- GIS_SAFE_RELEASE (*ptr) 21–22
- GisPtr 22
- GML schema file, creating and
 - editing 93

H

- handler, exception 23

I

- identity properties 211

K

- keywords, filter and expression 134

L

- lock types 217
- lock, long transaction exclusive 218
- locking 19, 216
 - ArcSDE limitations 229
 - defined 19
- long transaction
 - defined 125
 - leaf 125
 - root 20, 125–126, 216
- long transactions, locking and 216

M

- mappings
 - data type 232
 - physical 78
- memory management 21
- models, modifying 85

N

- non-feature class issues 83
- non-smart ptr 26

O

- object property 212
 - defined 17
 - overrides 210
- ObjectProperty types 82
- OGC WKT 141
- operations, comparison 136
- operations, data maintenance 109
- operator precedence 136
- operators 135
- Oracle
 - lock types 217
 - long transaction versions 216
- Oracle identity property 202
- Oracle provider, software
 - requirements 200
- Oracle reserved words used with filter and
 - expression text 215

- Oracle Workspace Manager 218
- Oracle-specific
 - schema creation restrictions 211
- Oracle, FDO Provider for 199
- overrides
 - class table 210
 - data property 210
 - FDO Provider for Oracle and Schema 209
 - geometric property 210
 - object property 210
 - schema 80

P

- package
 - connections 27
 - Schema 77
- packages, FDO 7
- parent in the schema classes 78
- properties
 - base 77
 - data 211
 - decimal 212
 - geometric 212
 - Geometry 15
 - identity 211
 - object 212
 - string 212
- property 215
 - association 15
 - data 15
 - defined 15
 - object 17
 - Raster 17
- property definitions, adding GML for 97
- property overrides
 - data 210
 - geometric 210
 - object 210
- property values 110
 - data 110
 - geometry 111
- Provider for ArcSDE
 - connection 232
 - defined 227

- Provider for MySQL
 - defined 249
- Provider for ODBC
 - defined 259
- Provider for Oracle
 - capabilities, FDO 219
 - connection, FDO 200
 - defined 199
 - general requirements 200
- Provider for Raster
 - defined 271
- Provider for SHP
 - defined 285
- Provider for SQL Server
 - defined 297
- provider, defined 9

Q

- query
 - creating 119
 - example 120

R

- raster property
 - defined 17
- references, cross-schema 78
- requirements
 - FDO Provider for ArcSDE 227
 - FDO Provider for Oracle 200
- restrictions, Oracle-specific schema creation 211
- rollback mechanism, schema 87
- root long transaction, defined 20

S

- SampleFeatureSchema.xml 106
- schema
 - defined 13
 - schema elements 14
- schema creation restrictions, Oracle-specific 211
- schema management 103

- schema mapping, defined 14
- schema modification restrictions,
 - Oracle-specific 212
- schema overrides 14, 209
- schema, create 81, 233
- schemas
 - describing 84
 - element states 86
 - modifying models 85
 - rollback mechanism 87
 - working with 80
- SDF
 - FDO Provider for 277
- software requirements 200, 227
- spatial context 150
- spatial context, defined 17
- special characters 137
- standalone class 83
- states, schema element 86
- string properties 212
- supported interfaces, LT 126

T

- table overrides, class 210
- text, expression 130

- text, filter 130
- transaction, long 20
- types
 - Geometry 149
 - ObjectProperty 82

U

- UnaryOperations 136

V

- values
 - deleting 116
 - updating 115

W

- WKT 141
- workspace, Oracle 218

X

- XML Format, FDO 87

