Effective Geodatabase Programming

Colin Zwicker

~~Erik Hoel~~

Rich Ruh

2019 ESRI DEVELOPER SUMMIT
Palm Springs, CA

Purpose

- Cover material that is important to master in order for you to be an effective Geodatabase programmer
- Provide additional insight regarding how we (the Geodatabase development team) conceptualize the architecture
- General focus areas:
  - Best practices
  - Common questions
  - New programming patterns

Assumptions

- Good working knowledge of the Geodatabase and experience in programming against the GDB API
- Code examples will use Java, C#, or C++
- Lots of technical content, very little time (60 minutes)
  - Please save questions for the end or at the showcase island
- Slides intended to be later used by you as a reference

Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Core.Data - the Pro Managed SDK
- Top Developer Mistakes

Unique instancing of objects

Unique Instancing of Objects

- Geodatabase objects that will have at most one instance instantiated
  - Similar to COM singletons except they are not cocreateable
  - Examples:
    - Datasets (tables, feature classes, feature datasets)
    - Workspaces and versions
- Regardless of the API that handed out the reference, it is the same reference
- Changes to the object affect all holders of the reference

Unique Instancing of Objects

```
public void run(Workspace workspace) throws IOException, AutomationException {

   IVersionedWorkspace vw = new IVersionedWorkspaceProxy(workspace);
   IVersion default1 = vw.getDefaultVersion();
   IVersion default2 = vw.findVersion("DEFAULT");

   System.out.println(default2.equals(default1)); <<- true
 }
```

```
public void run(Workspace workspace) throws IOException, AutomationException {

   IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");
   IFeatureClass fc2 = workspace.openFeatureClass("gdb.DBO.StateBoundaries");

   IFeatureDataset fd3 = workspace.openFeatureDataset("UnitedStates");
   IFeatureClassContainer fcc = new IFeatureClassContainerProxy(fd3);
   IFeatureClass fc3 = fcc.getClassByName("StateBoundaries");

   System.out.println(fc1.equals(fc2)); <<- true
   System.out.println(fc1.equals(fc3)); <<- true
   System.out.println(fc2.equals(fc3)); <<- true
 }
```

Unique Instancing of Objects – Special Case

Rows / features uniquely instanced only within an edit session

```
public void run(Workspace workspace) throws IOException, AutomationException {

  IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");

  IFeature f1 = fc1.getFeature(1);
  IFeature f2 = fc1.getFeature(1);
  System.out.println(f2.equals(f1)); <<- false

  IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
  workspaceEdit.startMultiuserEditing(esriMultiuserEditSessionMode.esriMESMNonVersioned);

  IFeature fe1 = fc1.getFeature(1);
  IFeature fe2 = fc1.getFeature(1);
  System.out.println(fe2.equals(fe1)); <<- true

  workspace.stopEditing(false);
}
```

Cursors

Cursor Types

- Three class cursors (does not apply to the Pro Managed SDK)
  - Search (general queries)
  - Update (positioned updates)
  - Insert (bulk inserts)
- One QueryDef cursor
  - Defined query (e.g., IQueryDef.Evaluate)
- What's the difference?
  - Rows returned by class cursors are bound to the class which created the cursor
  - Rows returned by a QueryDef cursor are not bound to a class

Cursor Types – Example highlighting class binding of rows

```
public void run(Workspace workspace)
   throws IOException, AutomationException {

   ITable testTable = workspace.openTable("parcels");
   ICursor classCursor = testTable.ITable_search(null, true);
   IRow classRow = classCursor.nextRow();
   ITable rowTable = classRow.getTable();
   System.out.println(rowTable.equals(testTable)); <<-- PRINTS true

   IQueryDef qdef = workspace.createQueryDef();
   qdef.setTables("parcels");
   ICursor queryCursor = qdef.evaluate();
   IRow queryRow = queryCursor.nextRow();

   try {
     ITable queryTable = queryRow.getTable();
   }
   catch (AutomationException ax) {
     System.out.println(ax.getMessage()); <<-- PRINTS AutomationException Message
   }
 }
```

Search Cursors

- All purpose class-based query API
  - Common APIs which create search cursors
  - ITable.Search, GetRow, GetRows
  - ISelectionSet.Search
- When in an edit session, the query may be satisfied by a cache (spatial/feature cache, object pool)
- When used within an edit session will only flush the class' cached rows
  - Could result in a database write
- Resulting rows can be modified
  - Store / Delete supported on the row object
  - InsertRow / UpdateRow / DeleteRow not supported on the search cursor

Update Cursors

- Positional update cursor
  - NextRow -> UpdateRow -> NextRow -> … -> NextRow -> UpdateRow
  - Update the row at the current cursor position
- Common APIs which create update cursors
  - ITable.Update
  - ISelectionSet2.Update
- Query is never satisfied by a cache
  - Use within an edit session will only flush the class' cached rows
- Resulting rows can be modified using ICursor.UpdateRow or DeleteRow
  - Should not be combined with Store and Delete

Update Cursors – Store events

- If the class supports Store events
  - An internal search cursor is created and UpdateRow and DeleteRow become equivalent to Row.Store and Row.Delete
- As a developer how do you know if the class supports Store events?
  - Non-simple feature types (!esriFTSimple)
  - Class participates in geometric network or relationships that require messaging
  - Custom features (ObjectClassExtension overrides)
- UpdateRow method behaviors
  - Error if called with a row that was not retrieved from the update cursor
  - Error if called with a row not at the current position

Update Cursors – Best Practices

- Do not use an update cursor across edit operations when editing
  - Very important when editing versioned data
  - Will throw an error
- Why? Because update cursors point to a specific state in the database
  - This state could get trimmed during an edit session
  - Updates could be lost!
- Best Practice – always scope the cursor to the edit operation
  - If you start a new edit operation you should get a new update cursor

Insert Cursors

- Primary use is for bulk inserts
- API which creates an insert cursor
  - ITable.Insert
- Best performance when using buffering and proper flushing
- If the class supports Store events (non-simple), search cursor semantics will be used
  - InsertRow becomes CreateRow and Store
  - Similar to update cursors

Insert Cursors - Buffering

- Call the ITable.Insert method with the argument “useBuffering” set to “true”
- Periodically call Flush
  - 1000 rows per flush is a good starting number
  - The higher the flush interval the less network traffic (but more re-insertion)
- Try to ensure that the class has no spatial cache – extra processing is required to keep the cache in sync
- Crucial that calls to InsertRow and Flush have the proper error handling since both can result in rows written to the database

QueryDef Cursors

- Executes user defined query (not bound to class)
- QueryDef cursors always bypass any row cache held by the class / workspace
- IQueryDef.Evaluate within an edit session will cause all cached rows to be flushed
- Rows from QueryDef cursors do not support APIs which reactively modify the row
  - Store not supported
  - Delete not supported

Recycling Cursors – What are they?

- A recycling cursor is a cursor that does not create a new client side row object for each row retrieved from the database
- Internal data structures and objects will be re-used
  - Memory
  - Object instances (e.g., geometry)
- Geodatabase APIs which support the creation of recycling cursors have a boolean argument
  - recycling = true creates a recycling cursor

Recycling Cursors – Interfaces supporting recycling cursors

- ITable / IFeatureClass
  - GetRows / GetFeatures
  - Search
  - Update
- ISelectionSet / ISelectionSet2
  - Search
  - Update
- ITableWrite
  - UpdateRows

Recycling Cursors - Example

- Calls to NextRow result in the same row instance

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <<-- recycling

    IRow firstRow = cursor.nextRow();
    IRow secondRow = cursor.nextRow();

    if (firstRow != null && secondRow != null) {
      System.out.println(secondRow.equals(firstRow)); <<-- PRINTS true
    }
  }
```

Recycling Cursors - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <<-- recycling

    int index = cursor.findField("location");

    IRow firstRow = cursor.nextRow();
    IGeometry firstGeometry = new IGeometryProxy(firstRow.getValue(index));

    IRow secondRow = cursor.nextRow();
    IGeometry secondGeometry = new IGeometryProxy(secondRow.getValue(index));

    if (firstGeometry != null && secondGeometry != null) {
      System.out.println(secondGeometry.equals(firstGeometry)); <<-- true
    }
  }
```

Recycling Cursors – When they should be used?

- When you don’t need to persist a reference to a row
- Don’t pass it around
  - Isolate the use of row to the local method which created the recycling cursor to minimize potential bugs
  - Do not pass references to the row around as some other method may decide to hold it
- Never directly edit a recycled row
- Proper use within an edit session can dramatically reduce resource consumption

Recycling Cursor - Reduced resource consumption

```
public void run(Workspace workspace, boolean recycling)
    throws IOException, AutomationException {

  ITable testTable = workspace.openTable("road_node_small");
  IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
  workspaceEdit.startMultiuserEditing(esriMESMNonVersioned);

  ICursor cursor = testTable.ITable_search(null, recycling);
  IRow row = cursor.nextRow();
  while (row != null) {
    System.out.println("OID: " + row.getOID());
    row = cursor.nextRow();
  }
  workspace.stopEditing(false); <<-- BREAKPOINT
}
```

- Test data: 50,000 rows with ~75 fields
  - recycling = true ~60MB memory
  - recycling = false ~185MB memory

Non-Recycling Cursors - What are they?

- A cursor that creates a new client side row object for each row retrieved from the database
- New internal data structures and objects will be created for each row
  - Memory
  - Object instances (e.g., geometry)
- Geodatabase APIs which support the creation of non-recycling cursors have a boolean argument
  - recycling = false creates a non-recycling cursor

Non-Recycling Cursors - When to use?

- When references to the current row and its values need to be persisted
- Commonly used to cache sets of rows (long lived references)
- Some Geodatabase APIs require sets of rows – should be retrieved as non-recycled rows
- Always edit non-recycled rows

Cursor FAQs

Question:

When should I release a reference to a cursor?

Answer:

Do not hold cursor references if they are not needed

- Release ASAP after fetching is completed
- Release after application is done with the cursor

Cursor FAQs

Question:

If I need to use a cursor inside an edit session, where should I create the cursor?

Answer:

Inside the edit session, scope to edit operation

```
// AVOID THIS PATTERN
workspace.startEditOperation();
ICursor cursor = testTable.ITable_search(null, true);
IRow row = cursor.nextRow();
workspace.stopEditOperation();

workspace.startEditOperation();
   row = cursor.nextRow();
   System.out.println(row.getOID());
workspace.stopEditOperation();
```

Cursor FAQs

Question:

Should I use a search cursor to update rows?

Answer:

Yes.

In fact, using search cursors within an edit session is the recommended way to update rows

Cursor FAQs

If I am editing existing data, what type of cursor should I use?

	ArcMap/Pro	Engine Simple Data	Engine Complex Data
Inside Edit Session	Search	Search	Search
Outside Edit Session	Search	Update / ITableWrite	Search

- ArcMap/Pro – possibility that an active cache can satisfy the query and no DBMS query is required
- Engine Simple Data
  - Inside Edit Session – take advantage of batched updates for edit operations
  - Outside Edit Session – performance, can handle errors on a per row basis
- Engine Complex Data – the system will emulate Search regardless

Caching geodatabase data

What is the Spati

- Client side cachi
- Can speed up qu
  - Reduces round
- When to use?
  - If making many
- ISpatialCacheMa
  - FillCache / Emp
  - CacheExtent, C

ArcObjects Library Reference (GeoDatabase)

ISpatialCacheManager Interface

Provides access to members that control the Spatial Cache Management. **Note:** the ISpatialCacheManager interface has been superseded byISpatialCacheManager3. Please consider using the more recent version.

Product Availability

Available with ArcGIS Engine, ArcGIS Desktop, and ArcGIS Server.

Description

ISpatialCacheManager is an optional interface that can be used to enable and disable feature caching within a specified spatial envelope.

Members

	All	Description
	CacheExtent	The extent of the spatial cache.
	CacheIsFull	Indicates if the spatial cache is full.
	EmptyCache	Empties the spatial cache.
	FillCache	Fills the spatial cache using the specified extent.

CoClasses that implement ISpatialCacheManager

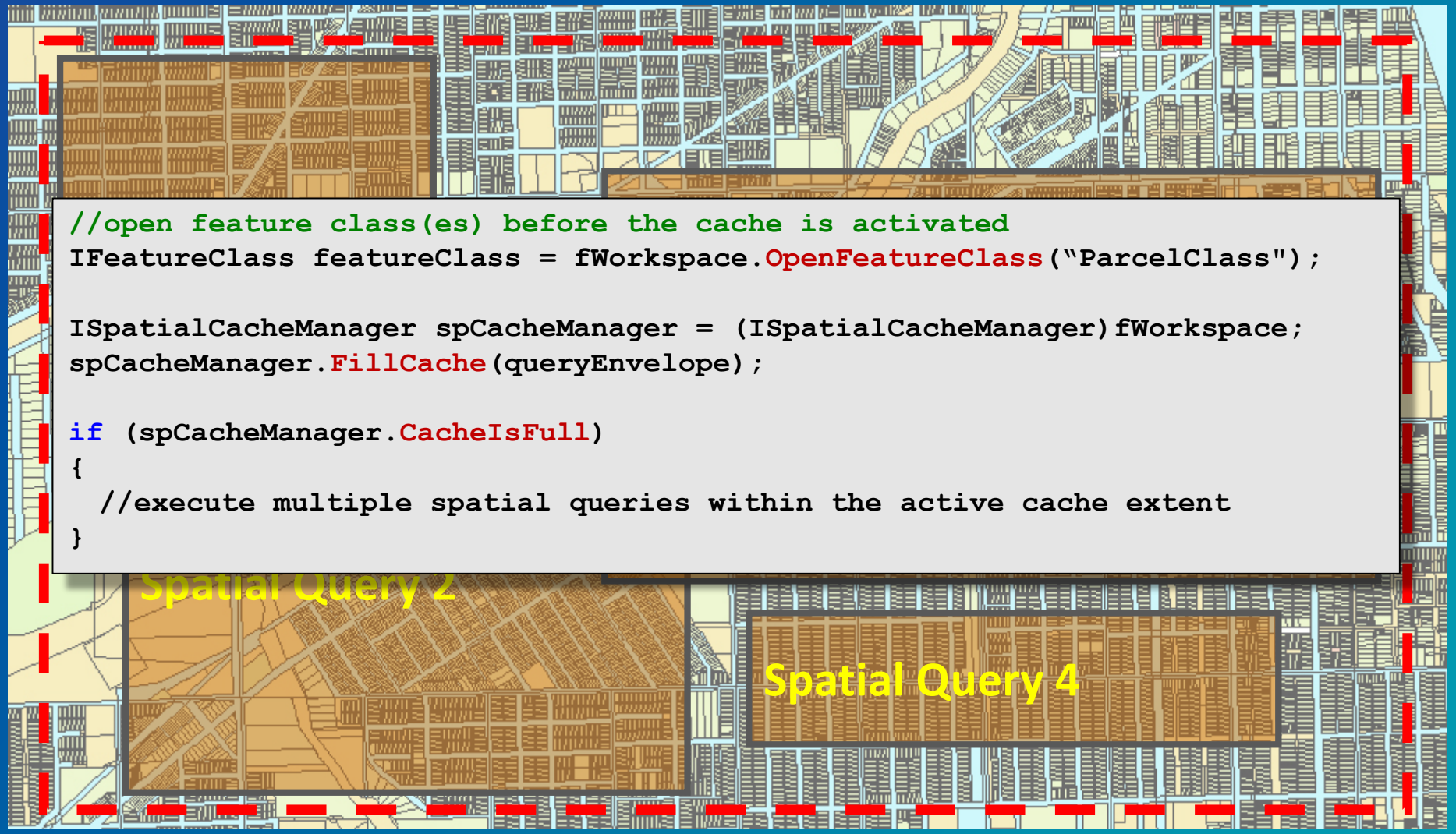
CoClasses and Classes	Description
VersionedWorkspace	VersionedWorkspace Object.
Workspace	Workspace Object.

Using the Spatial Cache

```
//open feature class(es) before the cache is activated
IFeatureClass featureClass = fWorkspace.OpenFeatureClass("ParcelClass");

ISpatialCacheManager spCacheManager = (ISpatialCacheManager)fWorkspace;
spCacheManager.FillCache(queryEnvelope);

if (spCacheManager.CacheIsFull)
{
  //execute multiple spatial queries within the active cache extent
}
```



What is the Schema C

- A cached snapshot of
  - Used by ArcGIS (ope
  - Requires a static dat
    - GDB schema chang
- Can improve perform
- APIs to access the sch
  - IWorkspaceFactoryS

ArcObjects Library Reference (GeoDatabase)

IWorkspaceFactorySchemaCache Interface

Manages Geodatabase workspace schema caches.

Product Availability

Available with ArcGIS Engine, ArcGIS Desktop, and ArcGIS Server.

Description

The schema cache is a cached snapshot of the geodatabase system tables (often referred to as the geodatabase schema). In an enterprise environment the geodatabase system tables (geodatabase schema) are in constant use by ArcGIS. Caching this schema locally can reduce database roundtrips by using the locally cached representation of the geodatabase schema.

Members

	All	Description
	DisableAllSchemaCaches	Disable the schema caches of all open workspaces.
	DisableSchemaCache	Disable the schema cache for a specific workspace.
	DisableSchemaCaching	All new workspaces handed out by the factory will not have schema caching enabled.
	EnableAllSchemaCaches	Enable the schema caches of all open workspaces.
	EnableSchemaCache	Enable the schema cache for a specific workspace.
	EnableSchemaCaching	All new workspaces handed out by the factory will have schema caching enabled.
	IsAnySchemaCacheStale	Checks all current schema caches for staleness.
	IsSchemaCacheStale	Checks a specific schema cache for staleness.
	RefreshAllSchemaCaches	Refreshes all current schema caches.
	RefreshSchemaCache	Refreshes the schema cache for a specific workspace.

CoClasses that implement IWorkspaceFactorySchemaCache

CoClasses and Classes	Description
SdeWorkspaceFactory (esriDataSourcesGDB)	ESRI SDE Workspace Factory.

When to use the Schema Cache

- Beneficial when working with large static data models
  - Tables, fields, domains, subtypes, and relationships are well defined and will not change
- If the application opens and uses many classes
  - These should be opened at the start of the application and references maintained throughout the lifetime of the application
  - E.g., utility network models

How to use the Schema Cache

- Enable schema cache before tables are opened
  - Calls to OpenFeatureClass, OpenTable, IName.Open will be optimized
  - Can enable schema caching at the factory level (IWorkspaceFactorySchemaCache)
- Cache needs to be “fresh”
  - If in dynamic environments schema changes will not be visible until cache is refreshed

```
if (sCache.IsSchemaCacheStale(workspace))
   sCache.RefreshSchemaCache(workspace);
```

- Your responsibility to disable the cache
  - Must be disabled before releasing the workspace object that is cached

Using Schema Caching

1. Cast to IWorkspaceFactorySchemaCache from the workspace factory and open the workspace

```
1  IWorkspaceFactorySchemaCache sCache =
   (IWorkspaceFactorySchemaCache)workspaceFactory;

   IWorkspace workspace = workspaceFactory.Open(propset, 0);

   sCache.EnableSchemaCache(workspace);

   IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
   IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");
   //...open all feature classes used in application

   sCache.DisableSchemaCache(workspace);
```

Using Schema Caching

2. Enable the schema cache on the workspace
   - Alternatively EnableSchemaCaching will enable caching on all workspace passed out by the factory

```
IWorkspaceFactorySchemaCache sCache =
(IWorkspaceFactorySchemaCache)workspaceFactory;

IWorkspace workspace = workspaceFactory.Open(propset, 0);

2 sCache.EnableSchemaCache(workspace);

IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");
//...open all feature classes used in application

sCache.DisableSchemaCache(workspace);
```

Using Schema Caching

3. Open all feature classes used in application
   - Largest benefit for stand alone feature classes and tables

```
IWorkspaceFactorySchemaCache sCache =
(IWorkspaceFactorySchemaCache)workspaceFactory;

IWorkspace workspace = workspaceFactory.Open(propset, 0);

sCache.EnableSchemaCache(workspace);

IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");
//...open all feature classes used in application

sCache.DisableSchemaCache(workspace);
```

3

Using Schema Caching

4 Disable the schema cache after the classes have been opened

```
IWorkspaceFactorySchemaCache sCache =
(IWorkspaceFactorySchemaCache)workspaceFactory;

IWorkspace workspace = workspaceFactory.Open(propset, 0);

sCache.EnableSchemaCache(workspace);

IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");
//...open all feature classes used in application

4 sCache.DisableSchemaCache(workspace);
```

Using Schema Caching - Gotchas

- Stale schema cached
  - Geodatabase schema changes will not be visible until cache is refreshed

```
IWorkspaceFactorySchemaCache sCache =
(IWorkspaceFactorySchemaCache)workspaceFactory;

IWorkspace workspace = workspaceFactory.Open(propset, 0);

sCache.EnableSchemaCache(workspace);

IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");
//...open all feature classes used in application

sCache.DisableSchemaCache(workspace);
```

Schema locks

Schema Locks

- Prevent clashes with other users when changing the geodatabase structure
- Exclusive and shared
  - ISchemaLock primarily used for establishing exclusive lock
  - Shared locks are applied when accessing the object
  - Promote a shared lock to an exclusive lock
  - Only one exclusive lock allowed
- Exclusive locks are not applied or removed automatically

Schema Locks – When to use

- You must promote a shared lock to exclusive when doing schema modifications (DDL):
  - Modifications to attribute domains; coded or range
  - Adding or deleting a field to a feature or object class
  - Associating a class extension with a feature class
  - Creating a topology, geometric network, network dataset, terrain, schematic dataset, representation or cadastral fabric on a set of feature classes
  - Any use of the IClassSchemaEdit interfaces
  - IFeatureClassLoad.LoadOnlyMode
  - Rebuilding spatial and attribute indexes

Schema Locks

- Demote exclusive lock to shared lock when the modification is complete
  - Includes when errors are raised during the schema modification
- Keep your use of exclusive schema locks tight
  - Prevents clashes with other applications and users
- If your application keeps a reference to an object with an exclusive schema lock
  - You will need to handle the exclusive schema lock when the object is used

Core.Data – the Pro managed SDK

Core.Data

- ArcGIS Pro does not use the managed API for its internal implementation

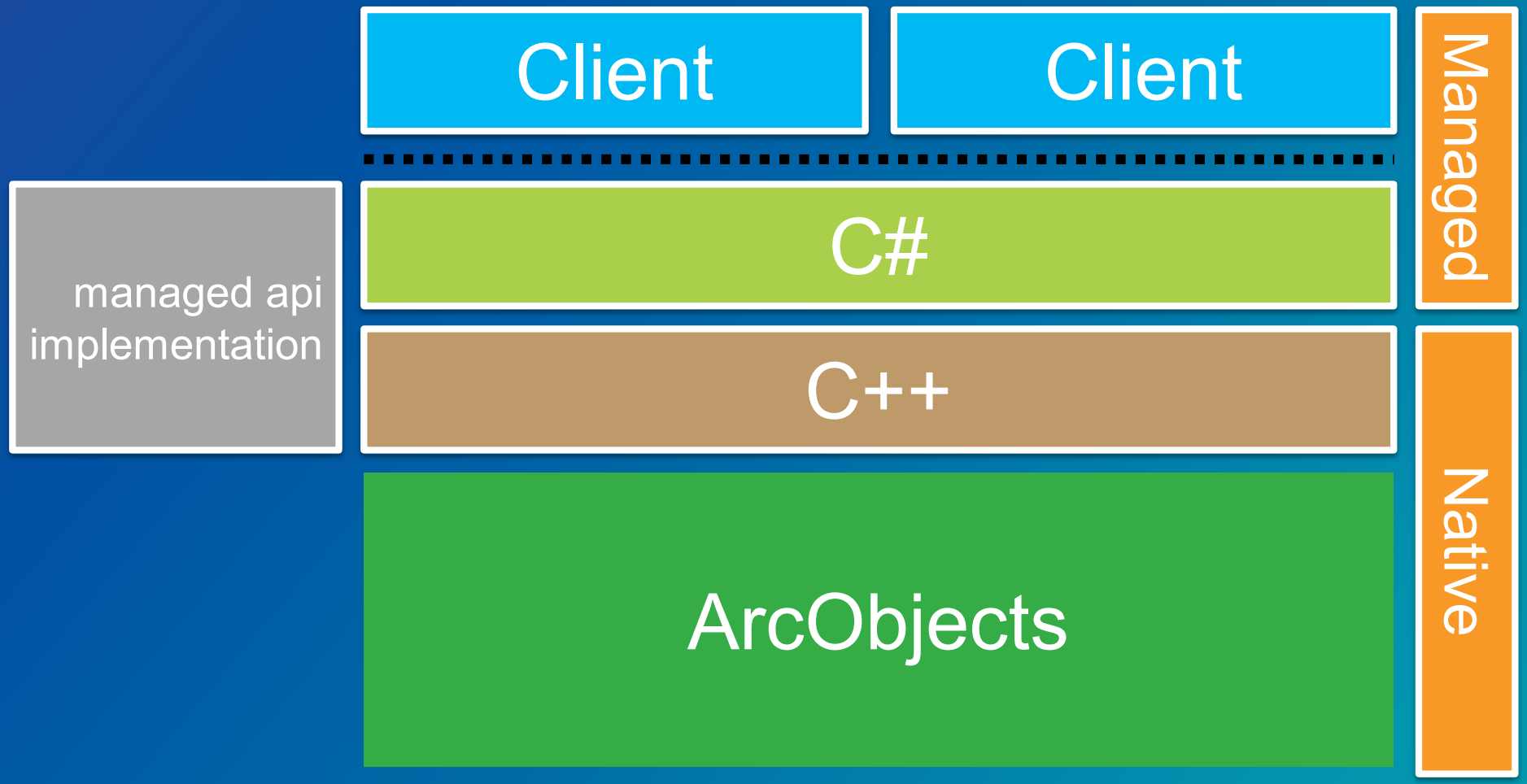
- ArcObjects is not exposed to external developers
- .NET API
- Coarse grained objects (layer, CIM model, etc.)

- Add-In extensibility mechanism for external developers
- Looser coupling at the application level than with ArcObjects Class Extensions

Core.Data - Data Manipulation Language Only

- Core.Data API is a DML-only (Data Manipulation Language) API
  - Cannot perform schema creation or modification operations:
    - creating tables or adding fields
    - creating domains or adding coded values
  - Schema operations are performed using the GP (Geoprocessing) tools
  - GP tools can be called from C# using the Geoprocessing API

Core.Data


Client
Client
Managed
managed api implementation
C#
C++
Native
ArcObjects

Core.Data - History

Grows on demand to support ArcGIS Pro

- 1.1 – web geodatabase support level (fileGDB, Enterprise)
- 1.2 – versioning, and feature service support
- 1.3 – database access, queryDef(initions)
- 1.4 – joins, SQLSyntax, feature service as geodatabase
- 2.0 – Datastore update, Raster, Blob fields, ~~datastore.GetPath~~
- 2.1 – Annotation, datastore.GetPath, table.GetCount & GetID
  - Versioning: Alter\Create\Delete version, reconcile, conflicts
  - Versioned feature service datastores
- 2.2 & 2.3 – Sorting Tables, Calculating statistics, ArchiveTable, Attribute Rules

General Geodatabase Access Pattern

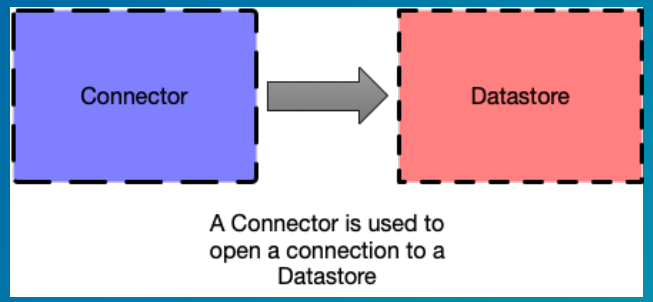
A Connector is used to open a connection to a Datastore

A Datastore is used to open a Dataset

Tables and Feature Classes execute queries to create RowCursors

Iterate through a RowCursor to fetch individual Rows or Features

Connectors



FileSystemConnectionPath
Sealed Class
→ Connector

- Properties
  - Path { get; } : Uri
  - Type { get; } : FileSystemDatastoreType
- Methods
  - FileSystemConnectionPath(Uri path, FileSysten

SQLiteConnectionPath
Class
→ Connector

- Properties
  - Path { get; } : Uri
- Methods
  - SQLiteConnectionPath(Uri path)

DatabaseConnectionProperties
Sealed Class
→ Connector

- Properties
  - AuthenticationMode { get; set; } : AuthenticationMode
  - Database { get; set; } : string
  - DBMS { get; set; } : EnterpriseDatabaseType
  - Instance { get; set; } : string
  - Password { get; set; } : string
  - ProjectInstance { get; set; } : string
  - User { get; set; } : string
  - Version { get; set; } : string
- Methods
  - DatabaseConnectionProperties(ConnectionInfo connectionInfo)
  - DatabaseConnectionProperties(EnterpriseDatabaseType databaseManagementSystemType)

FileGeodatabaseConnectionPath
Sealed Class
→ Connector

- Properties
  - Path { get; } : Uri
- Methods
  - FileGeodatabaseConnectionPath(Uri path)

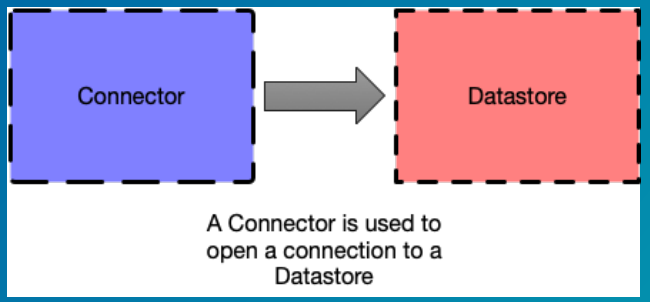
DatabaseConnectionFile
Class
→ Connector

- Properties
  - Path { get; } : Uri
- Methods
  - DatabaseConnectionFile(Uri path)

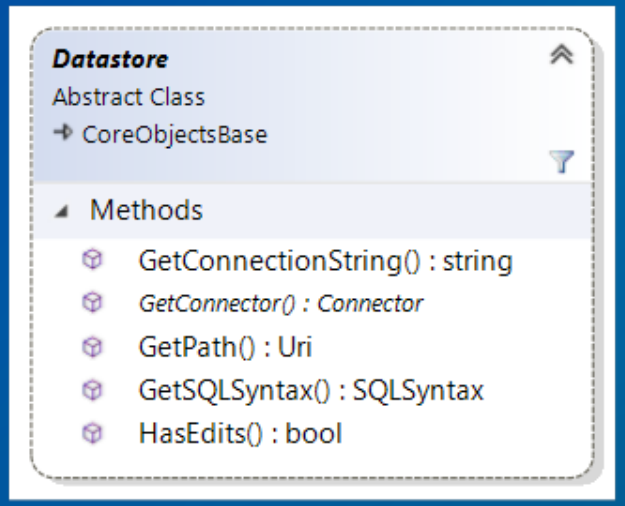
ServiceConnectionProperties
Sealed Class
→ Connector

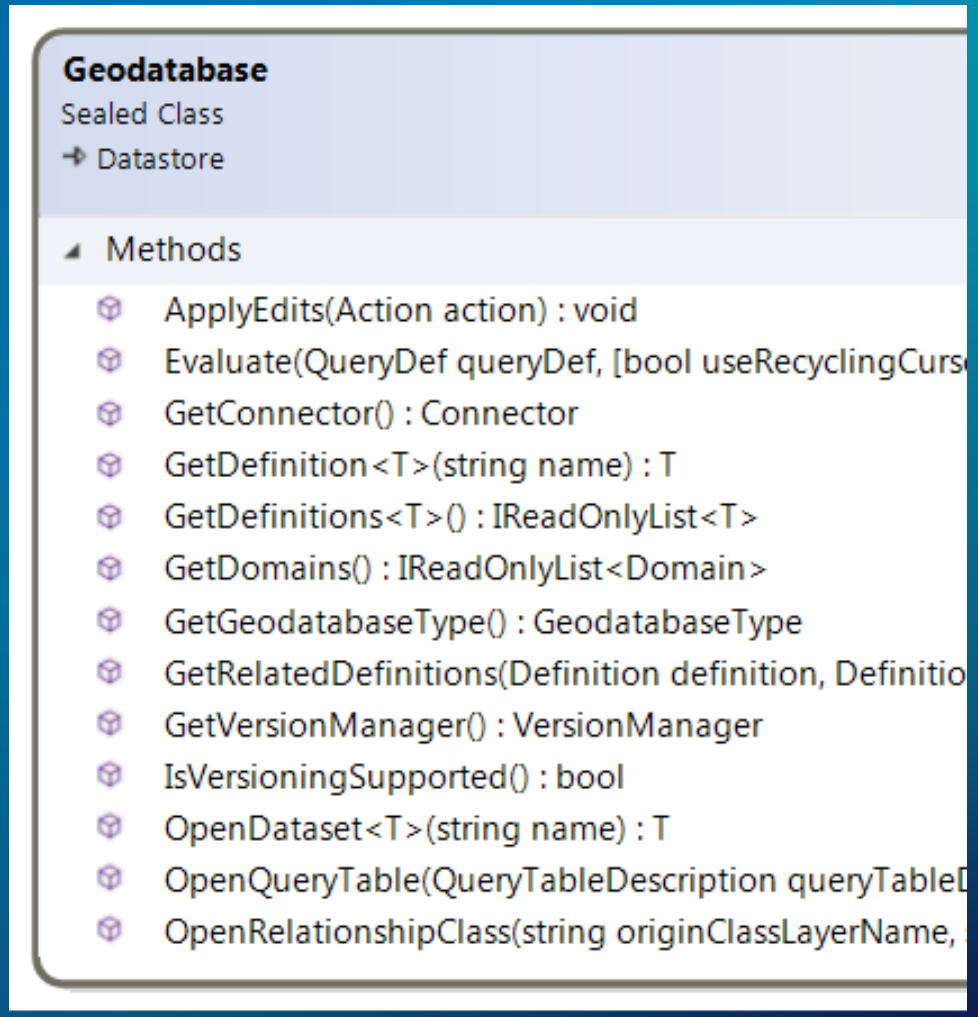
- Properties
  - Password { get; set; } : string
  - URL { get; } : Uri
  - User { get; set; } : string
- Methods
  - ServiceConnectionProperties(Uri serviceURL)

Datastore and Geodatabase

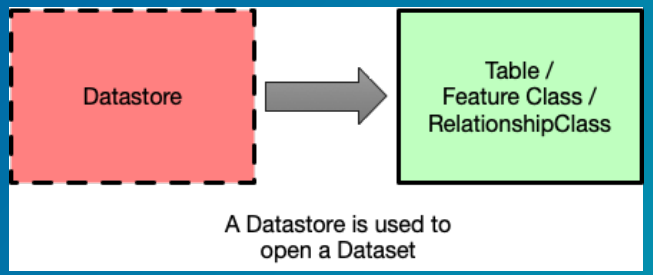


- Container of spatial and non-spatial datasets
- Created by a Connector
- Can also be obtained from a dataset
  - `Dataset.GetDatastore()`





Datasets — Table



- Table
  - Contains 0 or more Rows
  - Supports Search and Select

  - `geodatabase.OpenDataset<Table>("TableName");`
  - `featureLayer.GetTable();`
  - `row.GetTable();`

Table
Class
Dataset

Fields

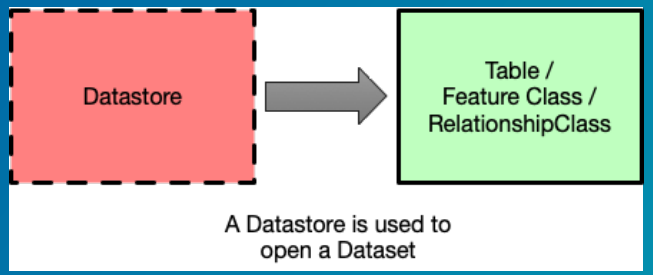
Properties

Type { get; } : DatasetType

Methods

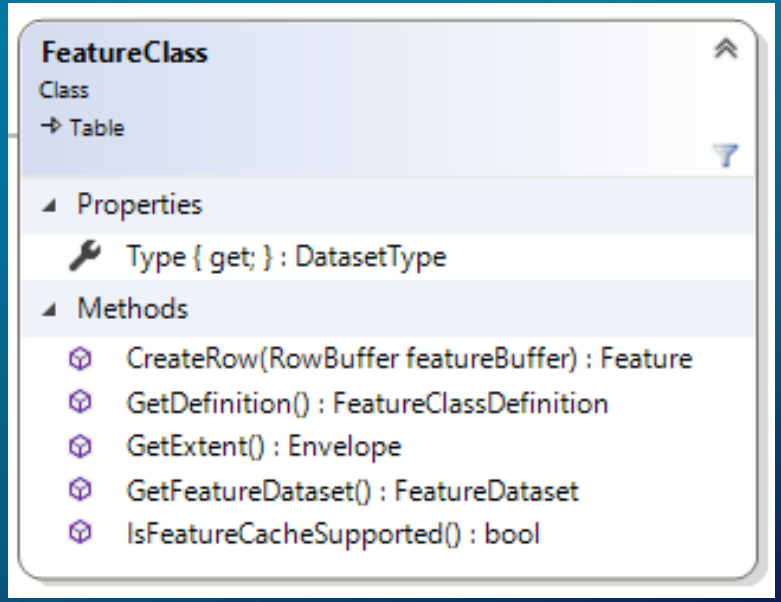
CreateRow(RowBuffer rowBuffer) : Row
CreateRowBuffer() : RowBuffer
CreateRowBuffer(Subtype subtype) : RowBuffer
DeleteRows(QueryFilter queryFilter) : void
GetControllerDatasets() : IReadOnlyList<Dataset>
GetCount() : int
GetCount(QueryFilter queryFilter) : int
GetDefinition() : TableDefinition
GetID() : long
GetJoin() : Join
IsAttachmentEnabled() : bool
IsControllerDatasetSupported() : bool
IsJoinedTable() : bool
RelateTo(Table destinationTable, VirtualRelationshipClassDescri
Search([QueryFilter queryFilter = null], [bool useRecyclingCurso
Validate(IReadOnlyList<Row> rows) : IReadOnlyDictionary<long
Validate(QueryFilter filter) : IReadOnlyDictionary<long, string>
Validate(Selection selection) : IReadOnlyDictionary<long, string

Datasets — Feature class

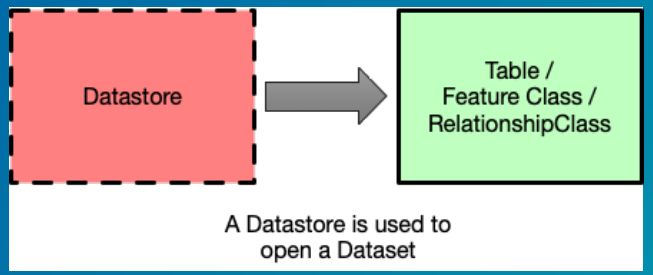


- Feature class
  - Inherit from tables
  - Tables with shape (point, line, polygon)
  - Contains 0 or more Features
  - Supports spatial queries and Selections

  - `geodatabase.OpenDataset<FeatureClass>("FeatureClassName");`
  - `featureLayer.GetTable() as FeatureClass;`
  - `row.GetTable() as FeatureClass;`
  - Open as table:
    - `Geodatabase.OpenDataset<Table>("FeatureClassName");`

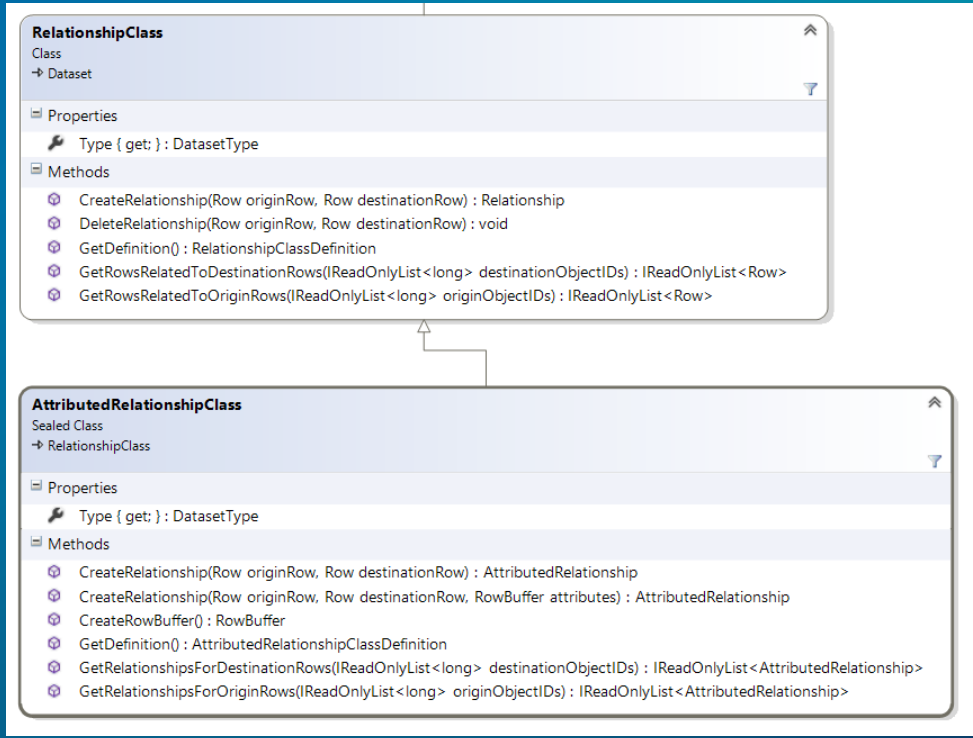


Datasets — Relationship Classes



- Geodatabase stored relationship between tables\feature classes
- Origin and Destination tables
- Cardinality of relationship between features

- Two types:
- RelationshipClass
  - May not have a backing table
- AttributedRelationshipClass
  - Inherits from RelationshipClass
  - Has a backing table
  - May have user defined attributes

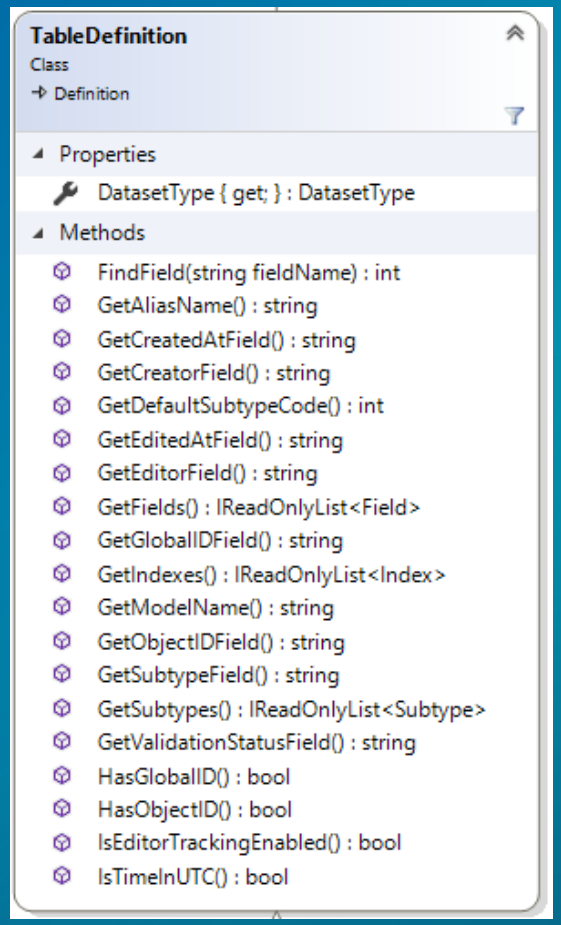


Definitions

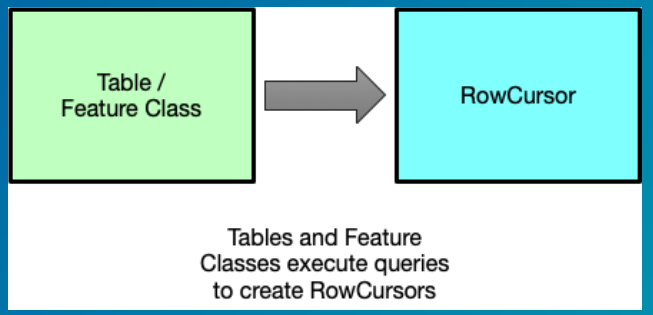
- Concept to represent information about the dataset
- Used to describe schema and unique properties
  - e.g., GetFields, HasGlobalID

- Separated from the dataset to facilitate a lightweight mechanism of discovery
- Opening dataset comparatively expensive
- Definitions can be use to filter datasets without opening them

- `table.GetDefinition();`
- `geodatabase.GetDefinition<TableDefinition>("TableName");`
- `geodatabase.GetDefinitions<TableDefinition>();`



Basic Queries using Search and Select



- Search
  - `table.Search()`
  - Return Rows (or subset of values) via RowCursor
  - Table bound
  - Supports Recycling

RowCursor
Sealed Class
CoreObjectsBase

Properties
- Current { get; } : Row

Methods
- FindField(string fieldName) : int
- GetFields() : IReadOnlyList<Field>
- MoveNext() : bool

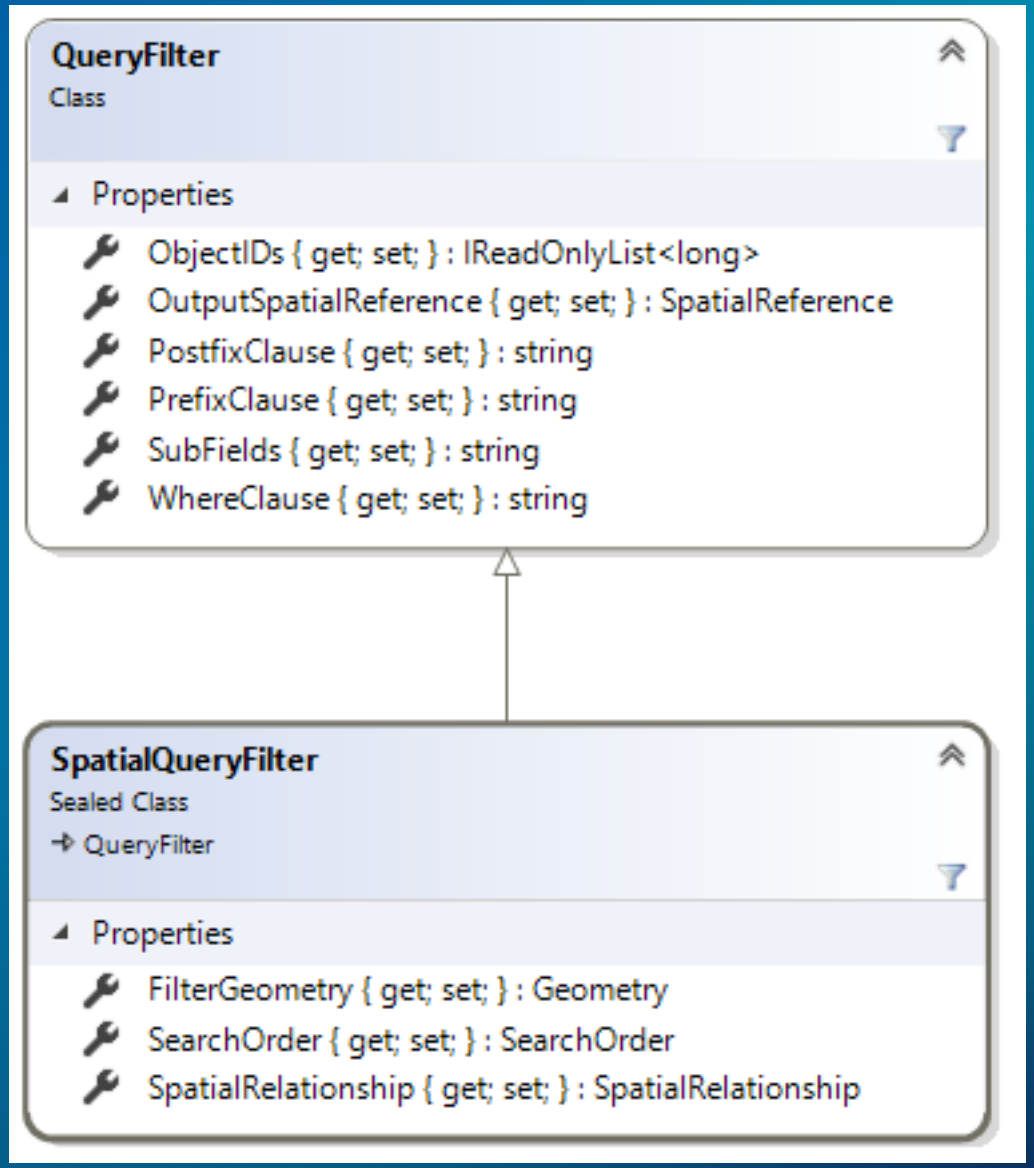
- Selection
  - `table.Select()`
  - List of Object IDs
  - Lightweight way to highlight features on map
  - Ability to combine

Selection
Sealed Class
CoreObjectsBase

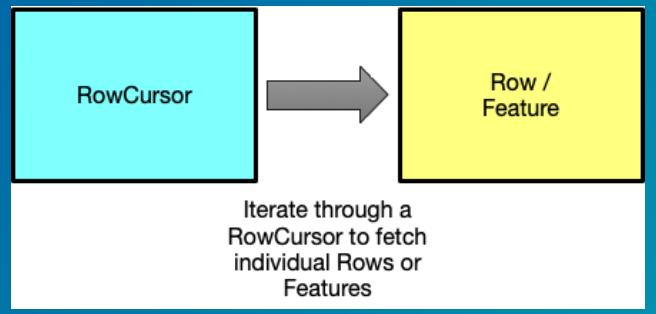
Methods
- Add(IReadOnlyList<long> objectIDs) : void
- Combine(Selection otherSelection, SetOperation setOperation) : Selection
- GetCount() : int
- GetObjectIDs() : IReadOnlyList<long>
- Remove(IReadOnlyList<long> objectIDs) : void
- Search([QueryFilter queryFilter = null], [bool useRecyclingCursor = true]) : RowCursor
- Select([QueryFilter queryFilter = null], [SelectionOption selectionOption = SelectionOption.Normal]) : Selection

Querying Data — Filtering queries

- QueryFilter
  - Used to restrict the records retrieved
    - Where clause
    - Subfields
  - Apply pre- or postfix clauses
    - DISTINCT
    - ORDER BY
- SpatialQueryFilter
  - Restrict retrieve records using a spatial query

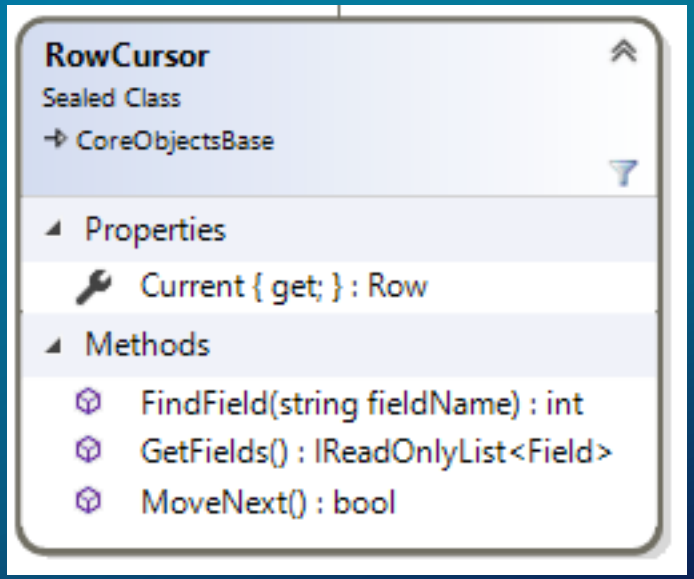


Row Cursor



- Represents a cursor
- Allows you to walk through a collection of rows

- `MoveNext()`
  - Positions the cursor at the next row in the sequence
  - Returns false when finished
- `Current`
  - Returns the current row



Querying Data — QueryDef

Geodatabase
Sealed Class
Datastore

Methods

- Evaluate(QueryDef queryDef, [bool useRecyclingCursor = true]) : RowCursor
- GetConnectionProperties() : ConnectionProperties
- OpenQueryTable(QueryTableDescription queryTableDescription) : Table

- QueryDef
  - Available with file and enterprise geodatabases
  - Tables are input parameter
    - Single table query
    - Or two or more joined tables within the same datastore
  - Rows do not implement GetTable()
  - Does not support field aliases
  - ‘Left’ most shape field supported

  - `geodatabase.Evaluate(queryDef, false);`

QueryDef
Class

Properties

- PostfixClause { get; set; } : string
- PrefixClause { get; set; } : string
- SubFields { get; set; } : string
- Tables { get; set; } : string
- WhereClause { get; set; } : string

Methods

- QueryDef()

Querying Data — Joins

- Used to combine fields from two tables into a single table representation
- Supports tables from one or two datastores
- Created using relationship classes
  - Relationship class stored in the geodatabase
  - Or virtual relationship class
- Result is read-only, but reflects updates to underlying tables
- Does not support many-many cardinality

Join
Sealed Class
CoreObjectsBase

Properties
- IsJoinPerformedOnClientSide { get; } : bool
- JoinType { get; } : JoinType

Methods
- FindFieldIgnoreQualification(string fieldName)...
- GetDestinationTable() : Table
- GetJoinedTable() : Table
- GetOriginTable() : Table
- GetRelationshipClass() : RelationshipClass
- GetTableAssociatedWithField(Field field) : Table
- Join(JoinDescription joinDescription)

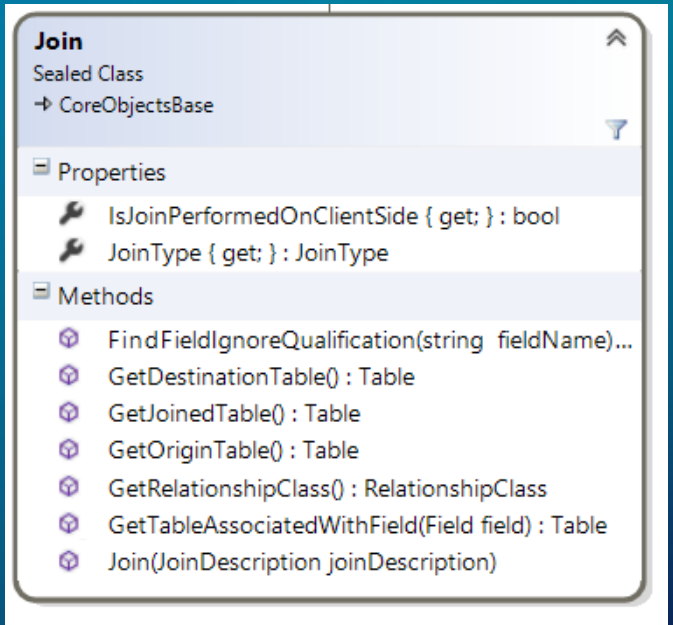
JoinDescription
Sealed Class

Properties
- ErrorOnFailureToProcessJoinOnServerSide { get; set; } : bool
- JoinDirection { get; set; } : JoinDirection
- JoinType { get; set; } : JoinType
- QueryFilter { get; set; } : QueryFilter
- RelationshipClass { get; } : RelationshipClass
- Selection { get; set; } : Selection
- TargetFields { get; set; } : IReadOnlyList<Field>

Methods
- JoinDescription(RelationshipClass relationshipClass)

Joins

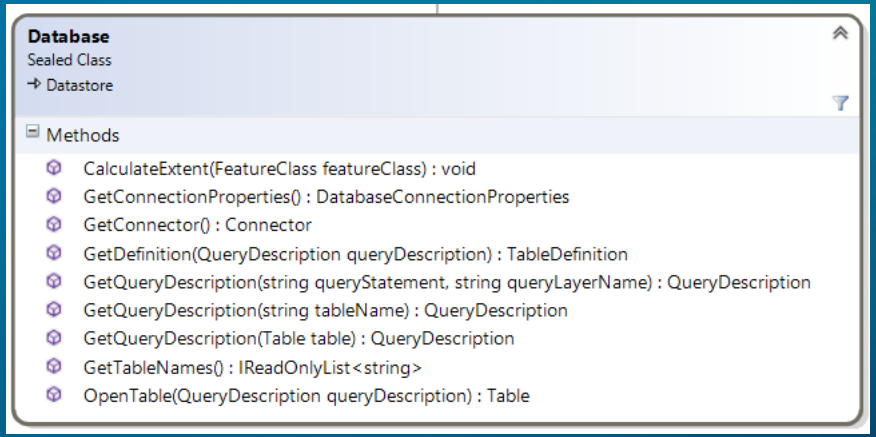
- `join.GetJoinedTable()`
- Definitions not supported on joined tables where tables are from multiple datastores
- Every attempt will be made to push join to database for performance
  - Within a single datastore



Databases and QueryLayers

- Database: Datastore for a DBMS without geodatabase enabled
- Allows queries of these tables via QueryLayers

- QueryLayers are created via `Database.GetQueryDescription`
  - From a table
  - From a table name
  - From a string containing a SQL query
- Modify `QueryDescription`
  - unique ID field, must have not Nulls
  - single geometry column
- Pass to `Database.OpenTable(QueryDescription)`



Versioning Support

Geodatabase
Sealed Class
Datastore

Methods
- GetVersionManager() : VersionManager
- IsVersioningSupported() : bool

VersionManager
Sealed Class
CoreObjectsBase

Methods
- ConnectToMoment(DateTime momentInTime) : Geodatabase
- CreateVersion(VersionDescription versionDescription) : Version
- CreateVersion(VersionDescription versionDescription, Version parent) : Version
- GetCurrentVersion() : Version
- GetHistoricalVersions() : IReadOnlyList<HistoricalVersion>
- GetVersion(string name) : Version
- GetVersions() : IReadOnlyList<Version>

- VersionManager
  - Available only if Geodatabase.IsVersioningSupported
  - Access to HistoricVersions or public\owned versions

VersionDescription
Sealed Class

Properties
- AccessType { get; set; } : VersionAccessType
- Description { get; set; } : string
- Name { get; set; } : string

Methods
- VersionDescription()

- Current capabilities
  - Create Versions
  - Connect to a specific named version
  - Connect to a Moment
  - List all versions in a geodatabase, including properties
  - List differences between Tables and Feature Classes from different Versions

Versioning Support

- Version
  - Alter or Delete
  - Reconcile and Post
  - Refresh

Version
Sealed Class
CoreObjectsBase

Methods
- Alter(VersionDescription versionDescription) : void
- Connect() : Geodatabase
- Delete() : void
- GetAccessType() : VersionAccessType
- GetChildren() : IReadOnlyList<Version>
- GetCreatedDate() : DateTime
- GetDescription() : string
- GetModifiedDate() : DateTime
- GetName() : string
- GetParent() : Version
- HasConflicts() : bool
- IsOwner() : bool
- Reconcile(ReconcileDescription reconcileDescription) : ReconcileResult
- Refresh() : void

ReconcileDescription
Sealed Class

Properties
- ConflictDetectionType { get; set; } : ConflictDetectionType
- ConflictResolutionMethod { get; set; } : ConflictResolutionMethod
- ConflictResolutionType { get; set; } : ConflictResolutionType
- TargetVersion { get; } : Version
- WithPost { get; set; } : bool

Methods
- ReconcileDescription()
- ReconcileDescription(Version targetVersion)

ConflictDetectionType
Enum
- ByRow
- ByColumn

ConflictResolutionMethod
Enum
- Abort
- Continue

ConflictResolutionType
Enum
- FavorTargetVersion
- FavorEditVersion

ReconcileResult
Sealed Class

Properties
- HasConflicts { get; } : bool

Core.Data - Conceptual Help

- https://github.com/Esri/arcgis-pro-sdk/wiki
- https://github.com/Esri/arcgis-pro-sdk-community-samples

- Geodatabase
  - ProSnippets: Geodatabase
  - ProConcepts: Geodatabase

- Editing
  - ProSnippets: Editing
  - ProConcepts: Editing

Top developer mistakes

Top Mistakes – Misusing recycling cursors

```
public static void RecyclingInappropriateExample(IFeatureClass featureClass, Boolean
    enableRecycling)
{
    using(ComReleaser comReleaser = new ComReleaser())
    {
        // Create a search cursor.
        IFeatureCursor featureCursor = featureClass.Search(null, enableRecycling);
        comReleaser.ManageLifetime(featureCursor);

        // Get the first two geometries and see if they intersect.
        IFeature feature1 = featureCursor.NextFeature();
        IFeature feature2 = featureCursor.NextFeature();
        IRelationalOperator relationalOperator = (IRelationalOperator)feature1.Shape;
        Boolean geometriesEqual = relationalOperator.Equals(feature2.Shape);
        Console.WriteLine("Geometries are equal: {0}", geometriesEqual);
    }
}
```

Top Mistakes – Calling FindField in a Loop

- Relying on FindField as opposed to hard-coded field positions is a good practice but overusing FindField can hinder performance

```
public static void ExcessiveFindFieldCalls(IFeatureClass featureClass)
{
    using(ComReleaser comReleaser = new ComReleaser())
    {
        // Open a cursor on the feature class.
        IFeatureCursor featureCursor = featureClass.Search(null, true);
        comReleaser.ManageLifetime(featureCursor);

        // Display the NAME value from each feature.
        IFeature feature = null;
        while ((feature = featureCursor.NextFeature()) != null)
        {
            Console.WriteLine(feature.get_Value(featureClass.FindField("NAME")));
        }
    }
}
```

Top Mistakes – leaking a reference

- C# and .Net being a managed language often takes care of allocating and releasing memory for us. But the story gets a bit more complicated when we involve COM objects, which we must in order to instantiate and interact with non-managed layers like ArcObjects.

- ```
  if (myDS != null) while (Marshal.ReleaseComObject(myDS) > 0) { }
  ```

- Anything holding a reference open, often this is:
- Datasets – e.g., tables, fc, workspaces
- Cursors, features\rows
  ```

Top Mistakes – DDL in edit sessions

- Methods that trigger DDL commands, such as IFeatureWorkspace.CreateTable or IClass.AddField, should never be called inside an edit session
  - DDL commands will commit any transactions that are currently open, making it impossible to rollback any unwanted edits if an error occurs
  - E.g., a custom editing application that adds new values to a coded value domain based on a user's edits, then fails unexpectedly when the application tries to commit the edits

Top Mistakes – Calling Store in Store-triggered events

- Calling Store on the object again triggers the event model from within the model, leading to unexpected behavior
  - In some cases, this results in infinite recursion causing an application to hang, while in others, errors are returned with messages that might be difficult to interpret

Top Mistakes – OnCreateFeature event

- Modifying the shape of the feature passed into the OnCreateFeature event, leading to unexpected behavior

```
void events_OnCreateFeature(ESRI.ArcGIS.Geodatabase.IObject obj)
{
  try
  {
    IGeometry newShape;

    // While creating a new feature, don't handle its OnCreate event by modifying its shape.
    IFeature feature = (ESRI.ArcGIS.Geodatabase.IFeature) obj
    feature.Shape = newShape;

    // Do not call Store() on the feature that is being handled in the event
    feature.Store();
  }
  catch (Exception exc) {}
}
```

Top Mistakes – Using GetFeature versus GetFeatures

- For performance purposes, anytime more than one feature is being retrieved using a known object ID, always use the GetFeatures method

```
private static void GetFeatureExample(IFeatureClass featureClass, int[] oidList)
{
    int nameFieldIndex = featureClass.FindField("NAME");
    foreach (int oid in oidList)
    {
        IFeature feature = featureClass.GetFeature(oid);
        Console.WriteLine("NAME: {0}", feature.get_Value(nameFieldIndex));
    }
}
```

Top Mistakes – Relying on name objects for caching

- Calling Open on a name object does not auto cache the reference to the returned dataset
- Relying only on the name object causes the underlying dbms to open the fully table reference each time
- Cache the open dataset reference at the start of the application

Top Mistakes – Careless reuse of variables

```
private static IFields FieldSetCreation()
{
    // Create a field collection and a field.
    IFields fields = new FieldsClass();
    IFieldsEdit fieldsEdit = (IFieldsEdit)fields;
    IField field = new FieldClass();
    IFieldEdit fieldEdit = (IFieldEdit)field;

    // Add an ObjectID field.
    fieldEdit.Name_2 = "OBJECTID";
    fieldEdit.Type_2 = esriFieldType.esriFieldTypeOID;
    fieldsEdit.AddField(field);

    // Add a text field.
    fieldEdit.Name_2 = "NAME";
    fieldEdit.Type_2 = esriFieldType.esriFieldTypeString;
    fieldsEdit.AddField(field);

    return fields;
}
```

Top Mistakes – Change non-persisting schema objects

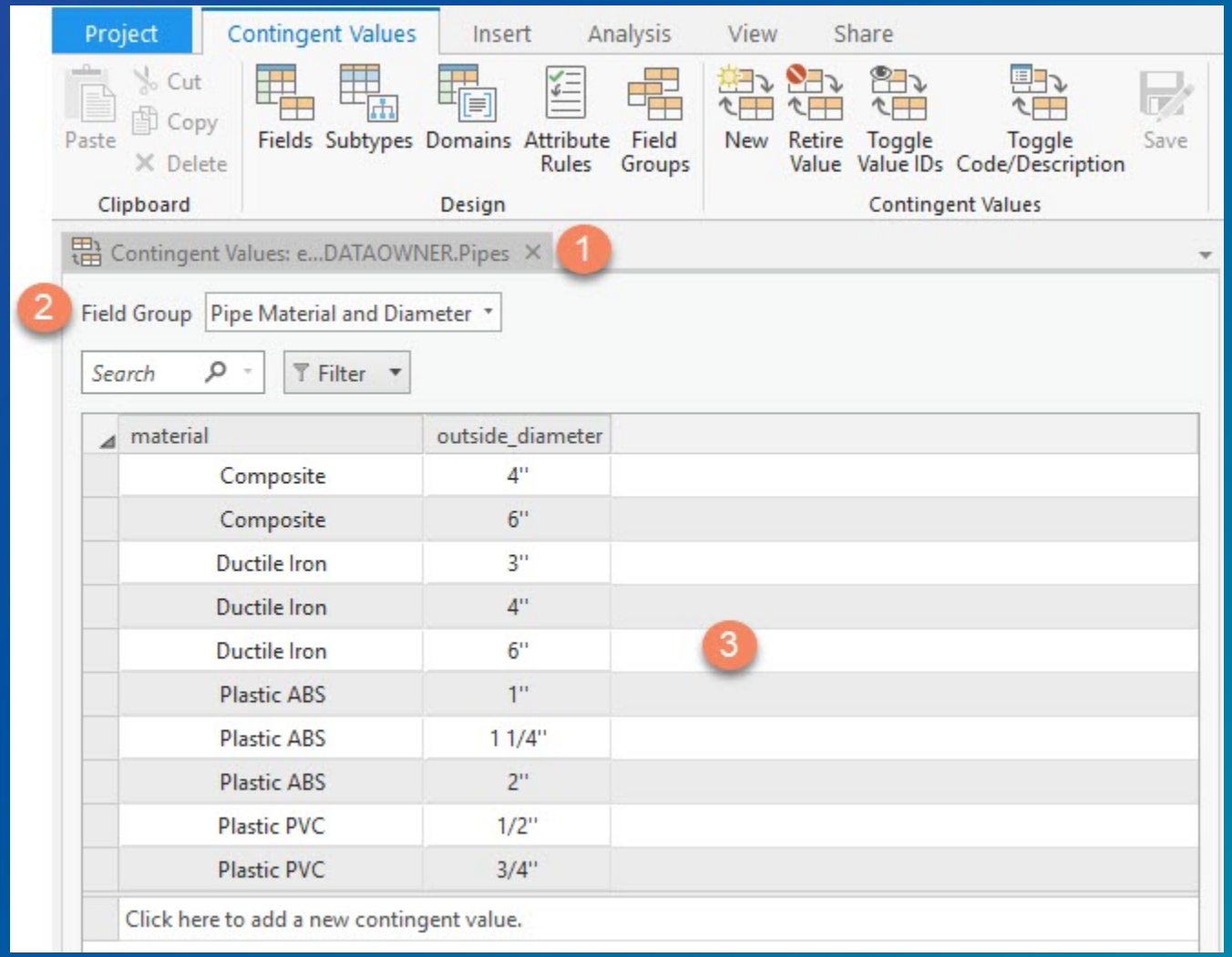
- When modifying geodatabase objects – e.g., datasets, domains, fields, etc., you should be aware that these classes fall into two categories of the following behaviors:
  - Those that automatically persist schema changes in the geodatabase, that is, tables
  - Those that do not – fields, domains, indexes

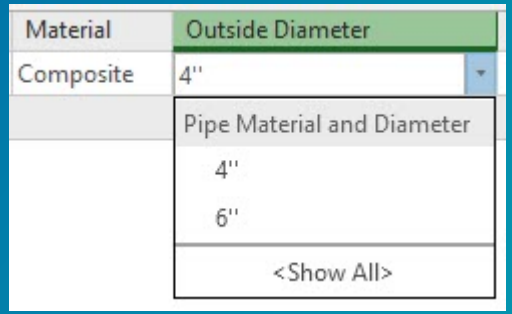
- A classic example of this are the methods, IClass.AddField and IFieldsEdit.AddField
  - When the former is called, the API adds a field to the database table
  - When the latter is called, a field is added to the field collection in memory but no change is made to the actual table

Top Mistakes – Inserts and relationship class notification

- Notification (also known as messaging) ensures the proper behavior of composite relationships, feature-linked annotation, and custom class extensions
  - This behavior is not free
  - Edits and inserts to datasets that trigger notification is noticeably slower than the same operation on datasets that do not trigger any notification
- This performance hit can be mitigated by ensuring that all notified classes are opened before any inserts taking place

Bonus – Contingent Values





Bonus – Attribute Rules

- **Are stored with the feature class\table in the geodatabase**
- **use Arcade scripts to either automatically populate fields, prevent edits or create error features for features that violate rules**

```
/* Calculation Rule Example */
// Calculate a uniqueID for each transformer getting created based on the containing
// substation and database sequence
// Trigger: insert
// Field : UniqueID
var seq = "Tx-" + NextSequenceValue("assetid")
var fsSubstation =  FeatureSetByName($datastore, "Substation", ["name"], true)
var fsIntersectingSubstation = Intersects(fsSubstation, Geometry($feature))
var substation = First (fsIntersectingSubstation)

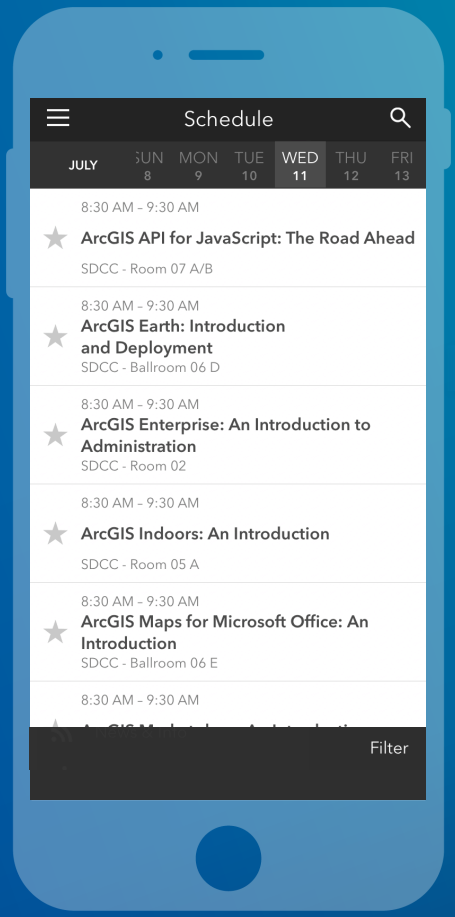
var subName  = ""
if (substation != null)
    subName =  substation.name

return subName + " - " + seq
```

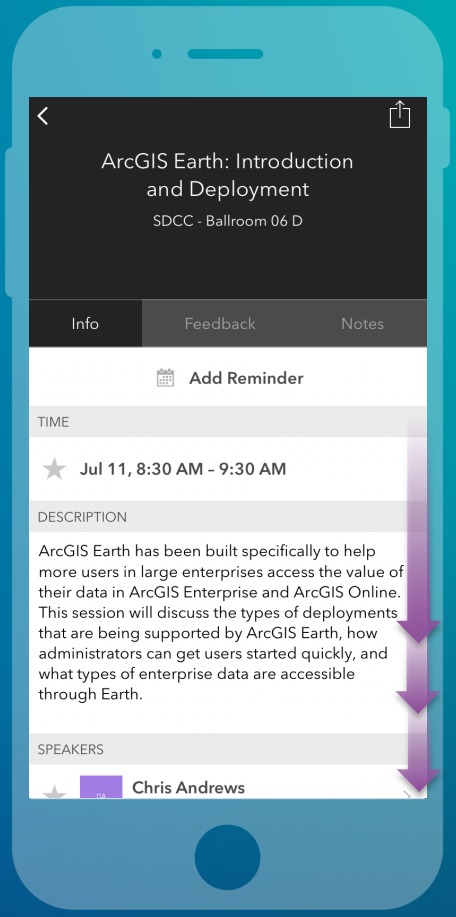
Please Take Our Survey on the App

Download the Esri Events app and find your event

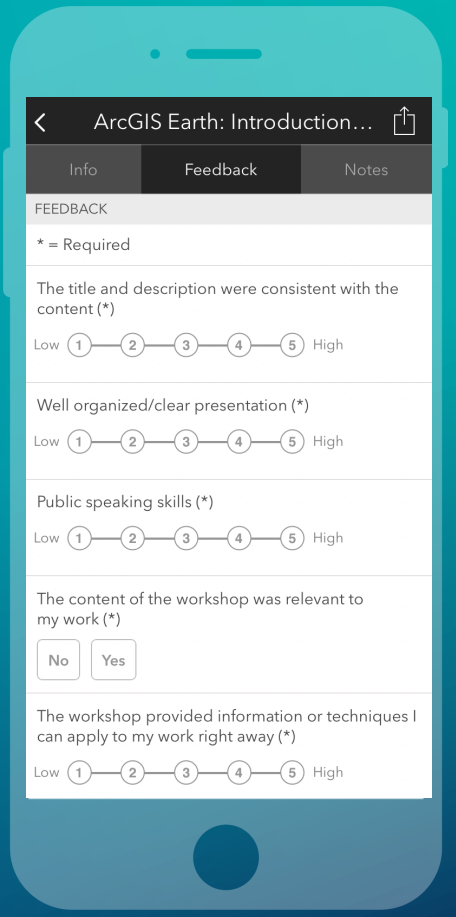
Select the session you attended



Scroll down to find the feedback section



Complete answers and select “Submit”



ArcGIS Pro SDK for .NET – Technical Sessions

Date	Time	Session	Location
Thu, Mar 07	1:00 pm - 2:00 pm	ArcGIS Pro SDK for .NET: An Overview of Geodatabase API	Mesquite G-H
	3:00 – 3:30 pm	Attribute Rules in Arcade	Demo Theater 2: Oasis 1-2
Fri, Mar 08	8:30 am – 9:30 am	Advanced Editing with Focus on Edit Operations, Transaction Types, and Events	Mesquite C
	10:00 am - 11:00 am	Demonstrating Pro Extensibility with Add-Ins	Mesquite B

ArcGIS Pro – Road Ahead Session

Date	Time	Session	Location
Thu, Mar 07	4:00 pm – 5:00 pm	ArcGIS Pro: The Road Ahead	Primrose B

esri

THE SCIENCE OF WHERE