ArcGIS Pro SDK for .NET: An Overview of the Geodatabase API

Colin Zwicker

Nghiep Quang

Sree Sreeraman

What will not be deeply discussed

- Add-in model
  - ArcGIS Pro SDK for .NET: UI Design and MVVM
- Threading model
  - ArcGIS Pro SDK for .NET: Configurations
- Editing
  - ArcGIS Pro SDK for .NET: Editing and Geodatabase Integration

Intro to Core.Data

- Geodatabase API namespace Core.Data

- ArcObjects is not exposed to external developers
- Managed .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

History

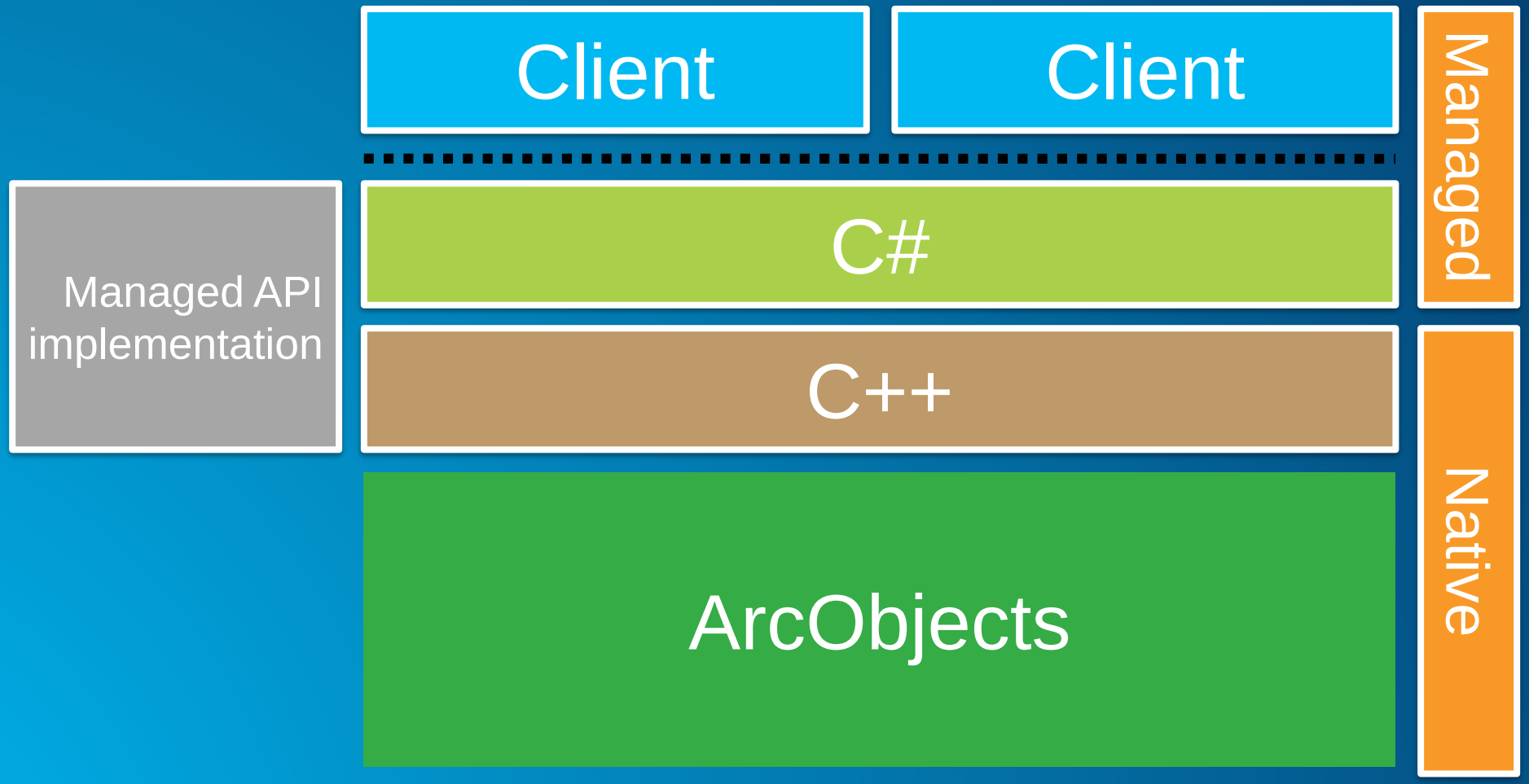
- Grows on demand to support ArcGIS Pro
- 1.1 – web geodatabase support level (fileGDB, Enterprise)
- 1.2 – versioning, and feature service support
- 1.3 – database access, queryDef(initions)
- 1.4 – joins, SQLSyntax, feature service as geodatabase

Architecture Principles

Architecture Principles

- Managed .NET API that provides access to the Geodatabase and supporting data
- It is an object-oriented API
- Aligns with modern C# practices and existing frameworks
- Adheres to the principles and architecture of the general Pro API

Core.Data


Client
Client
Managed
Managed API implementation
C#
C++
Native
ArcObjects

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

```
IReadOnlyList<string> args = Geoprocessing.MakeValueArray(@"C:\Data.gdb", "Places");
IGPResult gpResult         = await Geoprocessing.ExecuteToolAsync("CreateFeatureclass_management", args);

if (gpResult.IsFailed)
{
  // ...
}
```

Threading

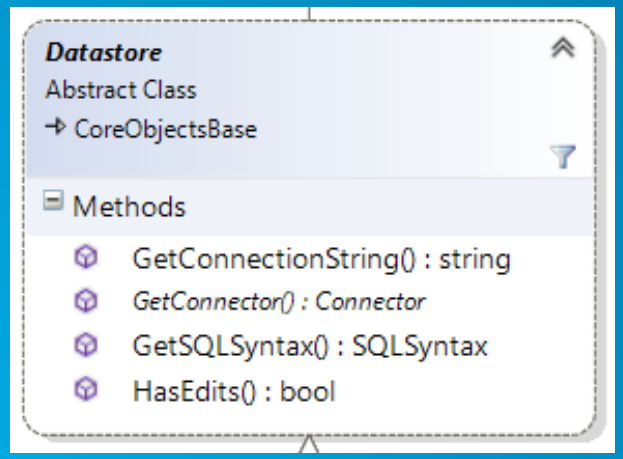
- Almost all of the methods in Core.Data API should be called on the MTC (Main CIM Thread)
  - API reference documentation on the methods that need to run on the MCT are specificed
  - These methods calls should be wrapped inside the QueuedTask.Run call
  - Failure to do so will result in CalledOnWrongThreadException or ConstructedOnWrongThreadException being thrown
- Read “Working with multithreading in ArcGIS Pro” conceptual help to learn more

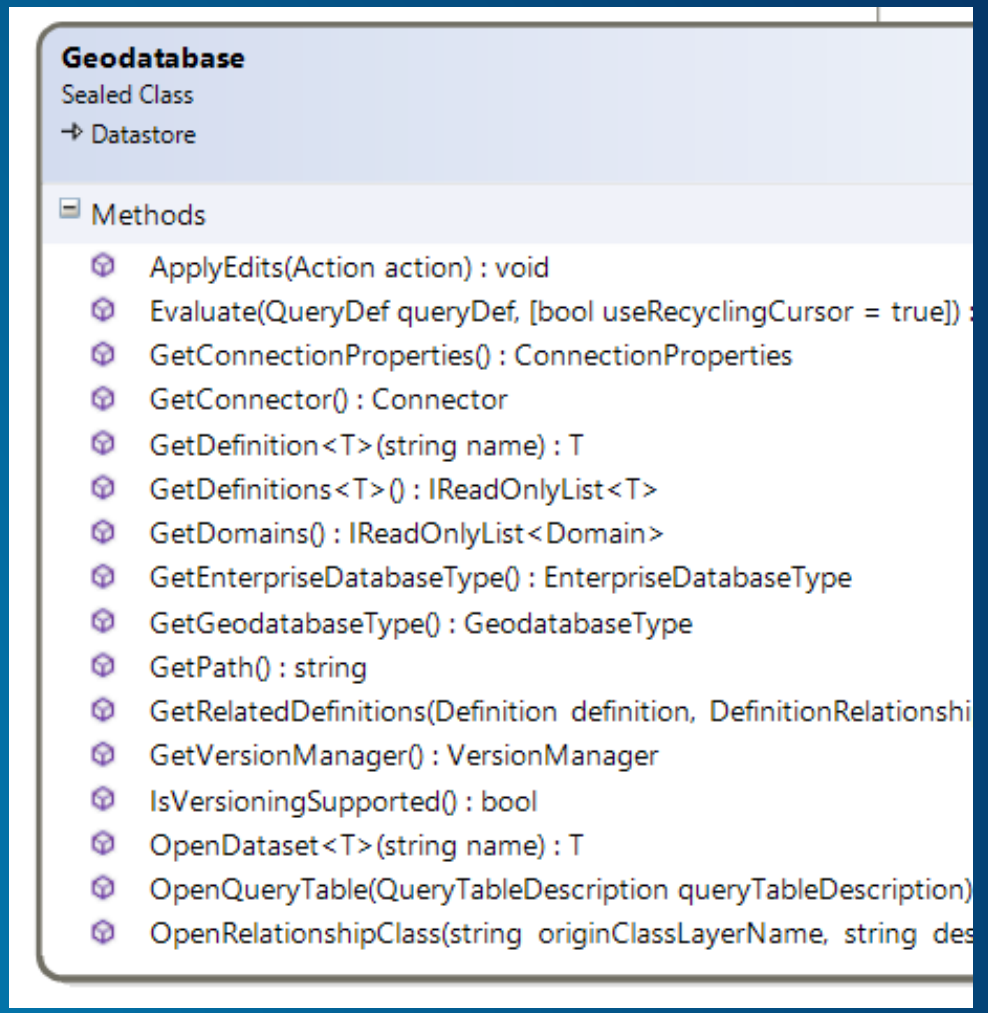
```
await QueuedTask.Run(() =>
{
  using (Geodatabase geodatabase = new Geodatabase(new DatabaseConnectionFile(new Uri(sdeFilePath))))
  {
    workspaceConnectionString = geodatabase.GetConnectionString();

    // ...
  }
});
```

Core.Data

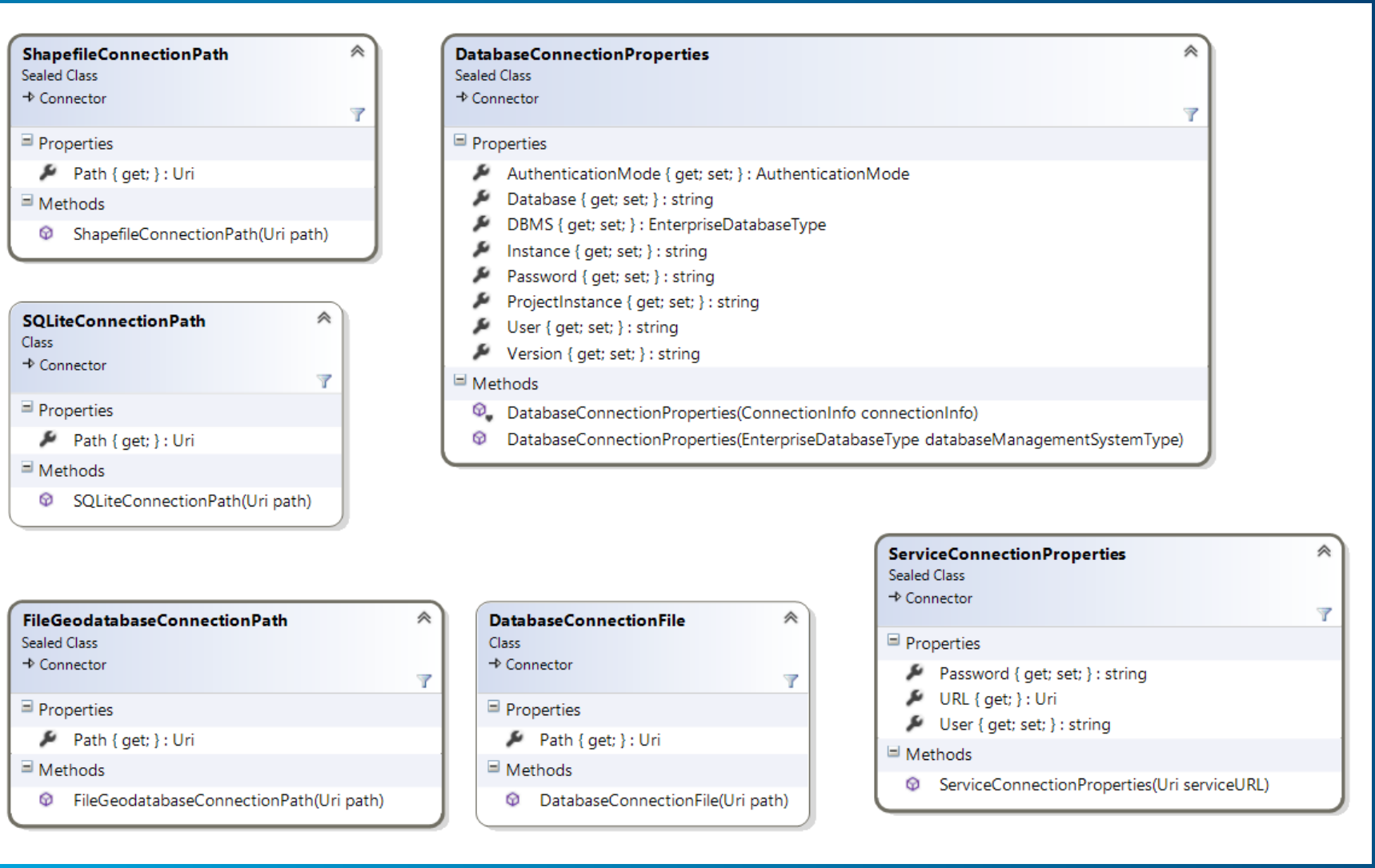
Datastore

- Container of spatial and non-spatial datasets
- Accessed by:
  - connection properties
  - uri (local file path or url)
  - ArcGIS.Core.Data.Dataset.GetDatastore()





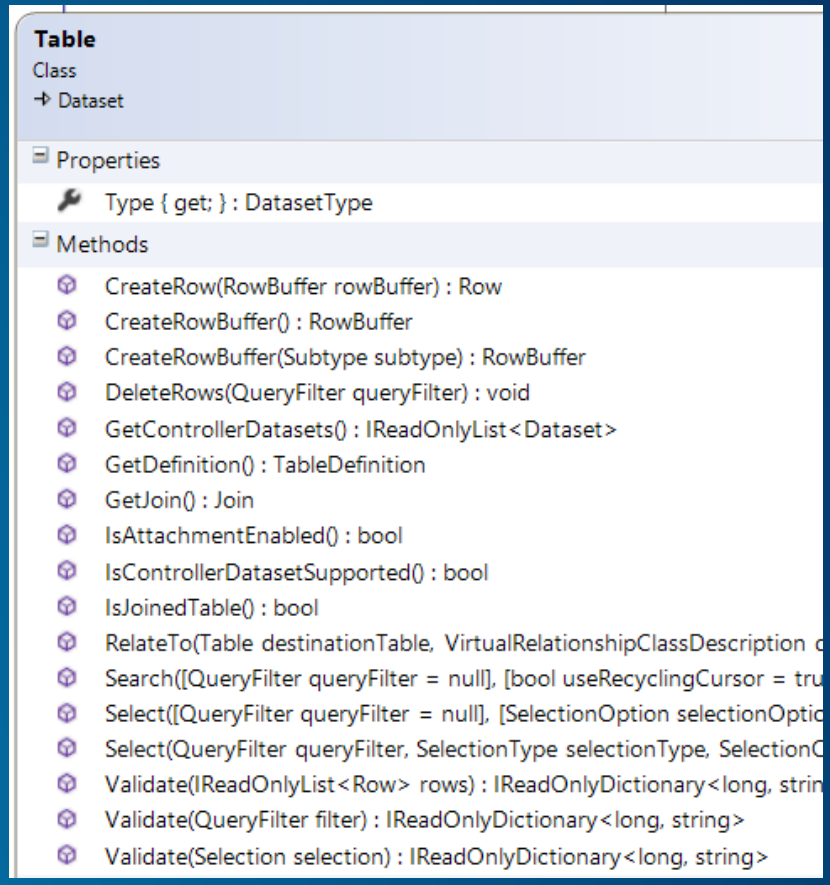
Datastore constructors



Datasets - Table

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

  - **geodatabase.OpenDataset<Core.Data.Table>(“TableName”);**
  - **Desktop.Mapping.FeatureLayer.GetTable();**
  - **Row.GetTable();**



Datasets - Feature class

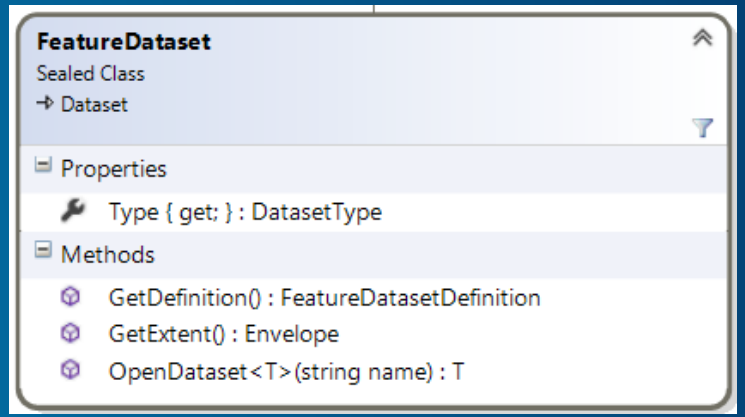
- Feature class
  - Inherit from tables
  - Tables with shape (point, line, polygon)
  - Contains 0 or more Features
  - Supports spatial queries and Selections

  - geodatabase.OpenDataset<Core.Data.FeatureClass>(“FeatureClassName”);
  - If (table is FeatureClass) Desktop.Mapping.FeatureLayer.GetTable() as FeatureClass;
  - If (row is Feature) Row.GetTable() as FeatureClass;
  - Open as table:
    - geodatabase.OpenDataset<Core.Data.Table>(“FeatureClassName”);

Datasets - Feature dataset

- **Feature Dataset**
  - **Collection of related datasets with a common coordinate system**
  - **Organize feature classes to support topology, network dataset or terrain dataset.**
  - **Acts as a container**

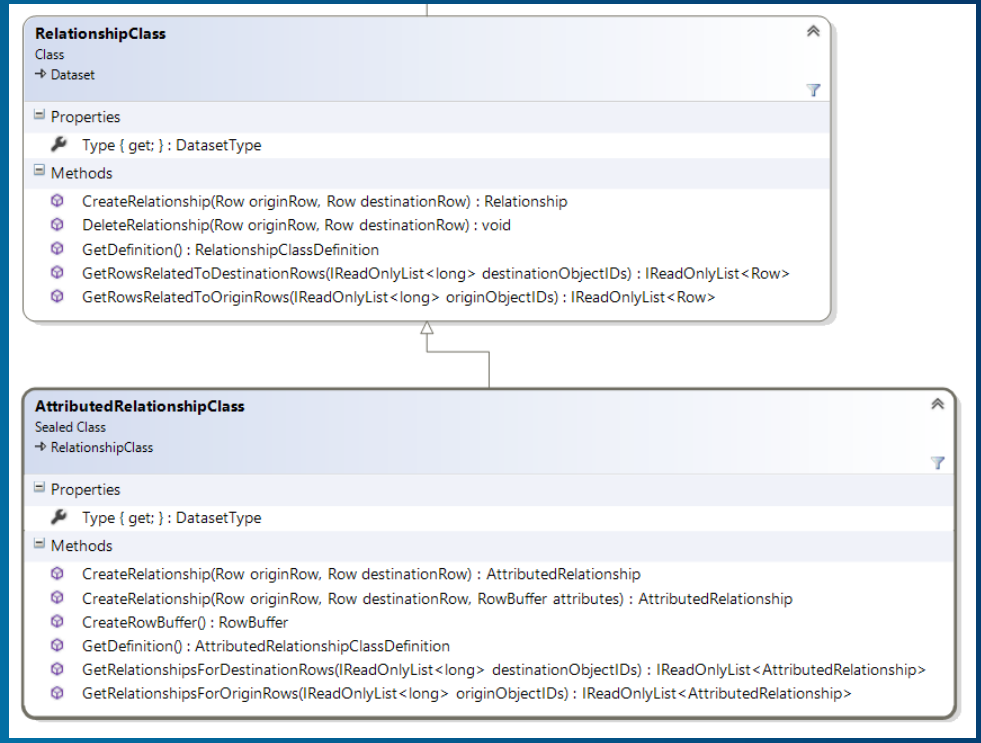
  - **geodatabase.OpenDataset<Core.Data.FeatureDataset>(“FeatureDatasetName”)**



Datasets - Relationships

- 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



Definition

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

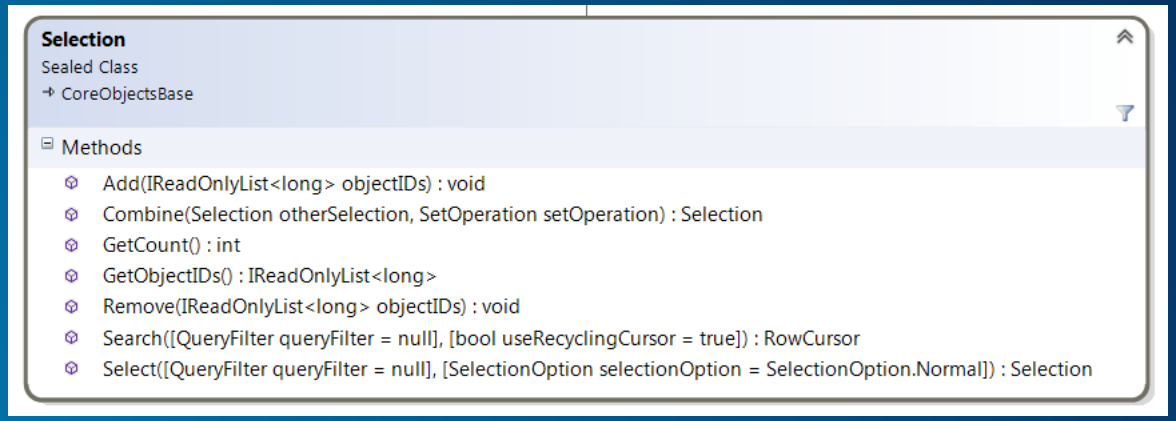
- Separated from the dataset to facilitate a lightweight mechanism of discovery
- Opening dataset comparatively expensive
- Definitions can be use to filter datasets without opening them

- table.GetDefitintion();
- geodatabase.GetDefinition<TableDefinition>(“TableName”);
- geodatabase.GetDefinitions<TableDefinition>();

Demo: Datastore & Dataset

Querying Data

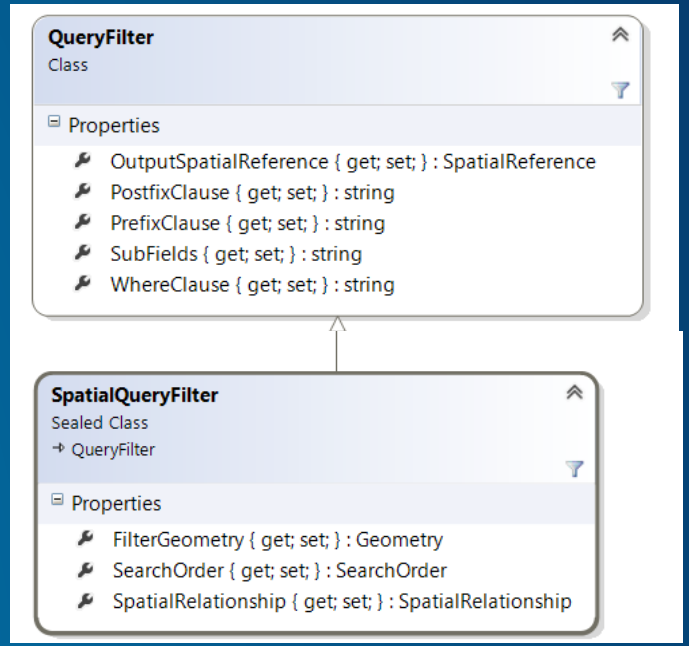
- **Selection**
  - List of Object IDs
  - Lightweight way to highlight features on map
  - Ability to combine



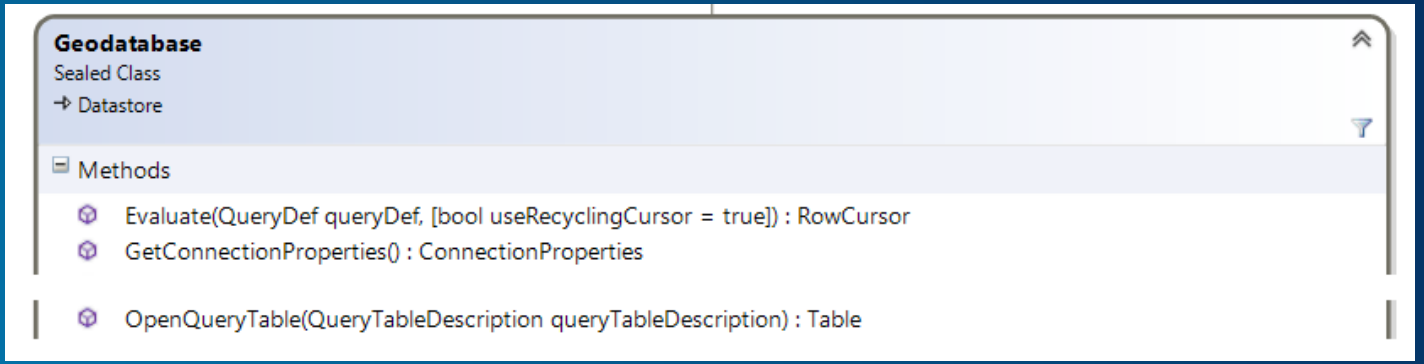
- **Search**
  - Return Rows (or subset of values) via RowCursor
  - Table bound
  - Supports Recycling

Querying Data - Filtering queries

- **QueryFilter**
  - **Used to restrict the records retrieved (Whereclause, Subfields)**
  - **Apply pre or postfix clauses (DISTINCT or ORDER BY)**
  - **Exception to MTC rule (can be created on a non-MTC thread)**

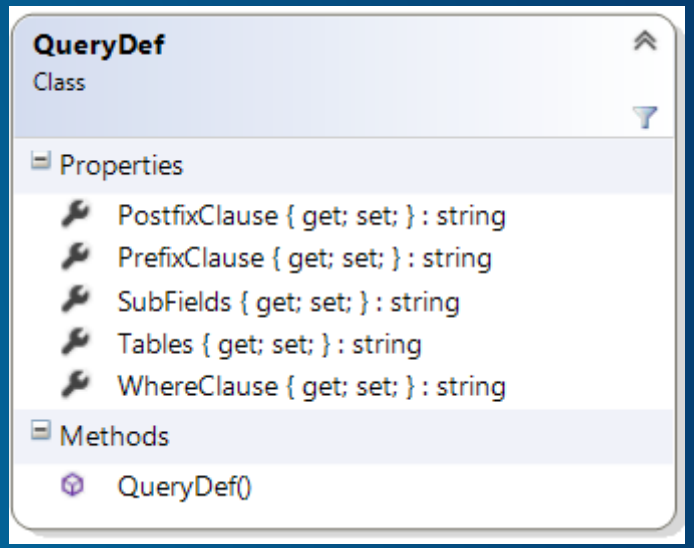


Querying Data - QueryDef

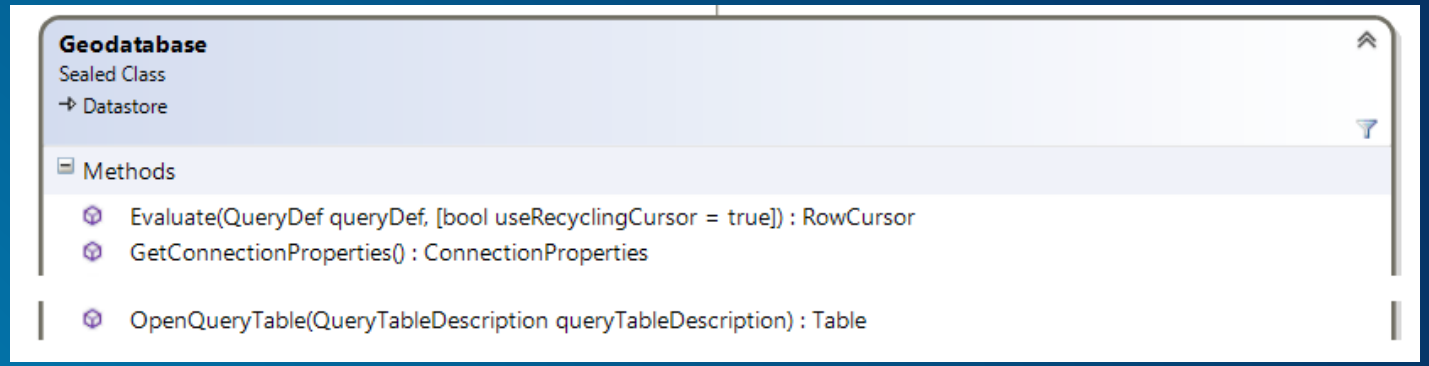


- QueryDef
  - Available from the geodatabase datastore (exception, not available with Feature Services)
  - Tables are input parameter
  - Single table query
  - Or two or more joined tables within the same datastore
  - Rows do not implement GetTable()
  - Does not support field aliases
  - ‘Left’ most shape field supported

  - geodatabase.Evaluate(queryDef, false);

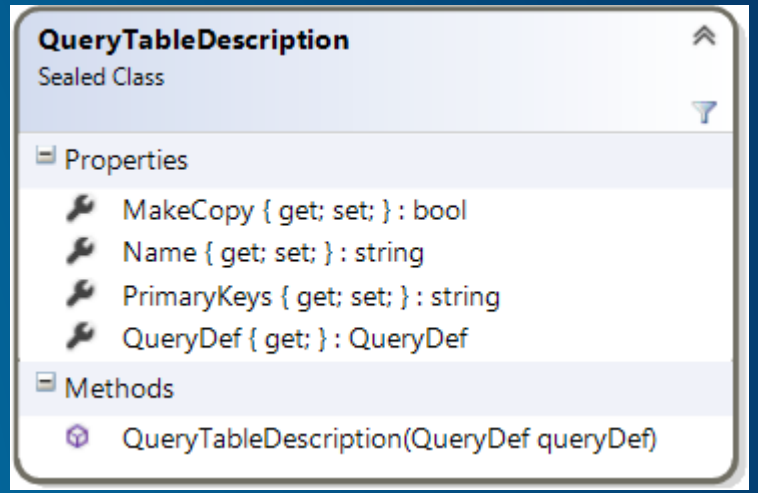


Querying Data - QueryTable



- QueryTable
  - ReadOnly
  - Virtual table (for map layer or Geoprocessing)
  - Requires a QueryTableDescription
    - Name
    - QueryDef
    - Key fields (Object ID \ Shape)

  - geodatabase.OpenQueryTable(QueryTableDescription);



Demo:
Working with rows & features

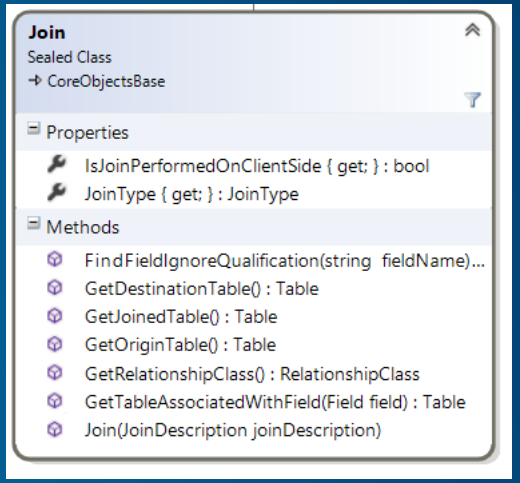
SQL Helper Method

- SQLSyntax
  - Eases the generation of SQL across multiple platforms
    - FileGDB, Oracle or SQLServer
  - Obtained from the datastore

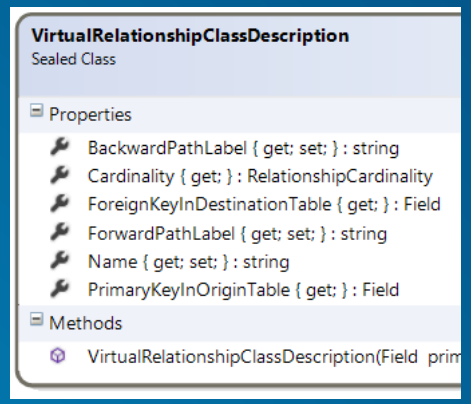
  - Fully qualify a table or field
  - Return owner name for a fully qualified table name
  - Find supported sql functions
  - Find keywords
  - Useful when creating fields or table via geoprocessing
    - Invalid characters

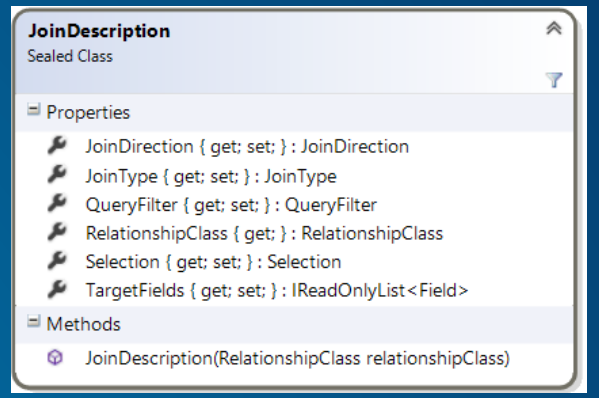
Joins

- Used to combine fields from two tables into a single table representation
- Supports table from one or two datastores
- Created using relationship classes
  - Relationship class stored in the geodatabase
  - Or Virtual relationship class
- Result is read-only, but reflect updates to underlying tables
- Do not support Many-Many cardinality
  - VirtualRelationshipClassDescription will raise an exception



Joins & Relates





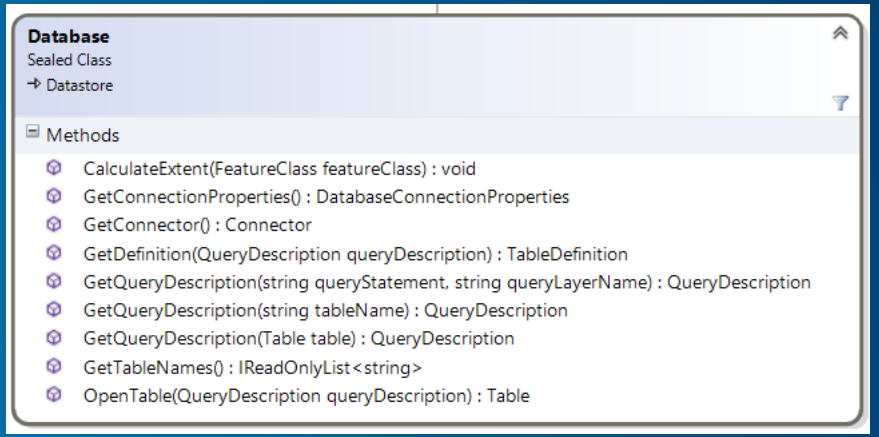
- join.GetJoinedTable()
- Definitions not supported on joined tables where tables are from multiple datastores
- Every attempt will be made to push join to database for performance
  - Within a single datastore

- Virtual relationship class can be used to support a Relate
  - Created with VirtualRelationshipClassDescription
- table.RelateTo(rightTable, virtualRelationshipClassDescription)

Database 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 string containing a sql query
- Modify QueryDescription
  - unique ID field, must have not Nulls
  - single geometry column
- Pass to Database.OpenTable

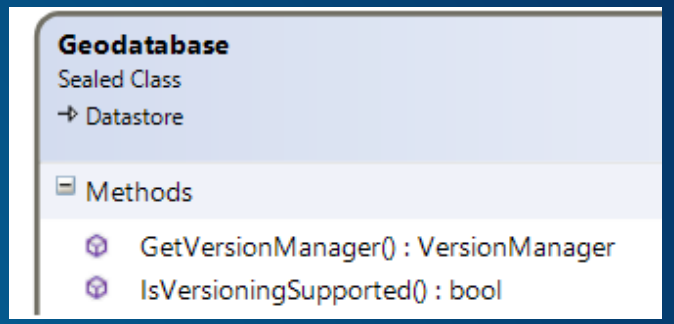


When to use difference query constructs

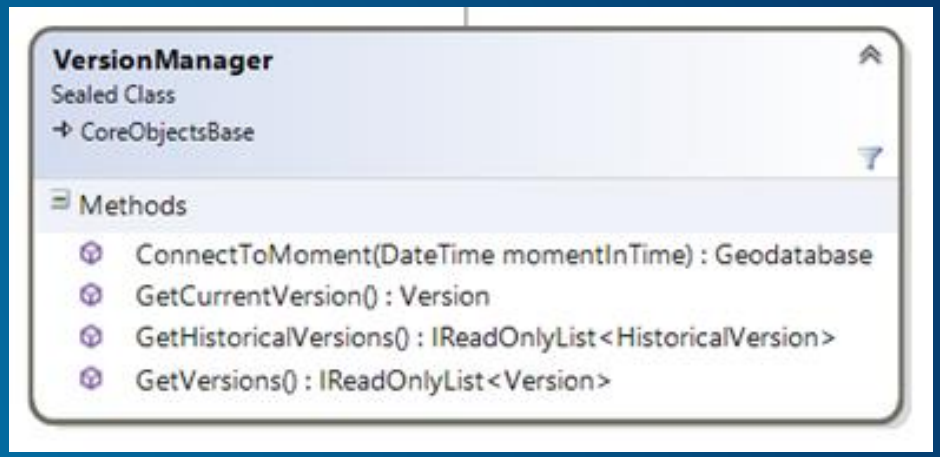
- Simple single registered table from geodatabase? -> table.Search or fc.Search
- Onetime query with join within a geodatabase? -> QueryDef
  - Don’t like sql? Join
- Reuse above query multiple times? -> QueryTable
- Use result as Layer or input to GP tool? -> QueryTable
- Join multiple tables in the same datastore? -> QueryDef
  - Join does support joining a table to an existing JoinTable
- Database simple query? -> QueryLayer
- Database join? -> QueryLayer
- Cross datastore join? -> Only possible via Join

Demo: Joins

Versioning Support



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



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

Editing

- ArcGIS.Desktop.Editing
  - EditOperation

- ArcGIS Pro SDK for .NET: Editing and Geodatabase Integration
  - 10:00 am - 11:00 am in Mesquite C

Core.Host

- Using CoreHost in ArcGIS.Core.Hosting namespace
  - Programs using the API can be run “head-less” (i.e., standalone)
  - GP tools must be run directly from Python
- ArcGIS.Desktop.Editing not possible
  - geodatabase.ApplyEdits
    - Not to be used in Add-in

Best Practices

Garbage Collection

- By default, locks and connections on database objects held by .NET objects are released at non-deterministic times (When garbage collection runs)
  - As a consequence, connections and locks on any dataset referenced by the .NET object will also be held
- Using blocks or calling Dispose on objects after use provides greater control on when connections and file system locks are released

```
FeatureClass featureClassIndeterministic = geodatabase.OpenDataset<FeatureClass>(featureClassName);

using (FeatureClass featureClassDeterministic = geodatabase.OpenDataset<FeatureClass>(featureClassName))
{
  Console.WriteLine(featureClassDeterministic.GetName());
}
```

- Locks acquired by featureClassIndeterministic are released when garbage collector runs
- Locks acquired by featureClassDeterministic are released at the end of the using block

Best Practices

- **Do explicitly dispose of objects of the following types:**
  - ***ArcGIS.Core.Data.Datastore*** **and its derived classes (e.g.,** ***Geodatabase*****)**
  - ***ArcGIS.Core.Data.Dataset*** **and its derived classes (e.g.,** ***FeatureDataset*****,** ***Table*****,** ***FeatureClass*****,** ***RelationshipClass, etc*****)**
  - ***ArcGIS.Core.Data.RowCursor***
  - ***ArcGIS.Core.Data.RowBuffer***
  - ***ArcGIS.Core.Data.Row*** **and its derived class,** ***Feature***

Best Practices (cont'd)

- **Two standard ways to explicitly dispose of objects in C#**
  - **Via the *try/finally* block:**

```
public void DoSomething()
{
  Geodatabase geodatabase = null;
  FeatureClass featureClass = null;

  try
  {
    geodatabase = new Geodatabase(connectionProperties);
    featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS);
    //…
  }
  finally
  {
    if (featureClass != null)
      featureClass.Dispose();

    if (geodatabase != null)
      geodatabase.Dispose();
  }
}
```

Best Practices (cont'd)

- **Via the *using* statement:**

```
public void DoSomething()
{
  using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
  using (FeatureClass featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS))
  {
    //…
  }
}
```

```
public void DoSomething()
{
  using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
  {
    using (FeatureClass featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS))
    {
    //…
    }
  }
}
```

Best Practices (cont'd)

- **Do remember to explicitly dispose of Row/Feature returned from RowCursor**
  - **Not explicitly disposing of Row/Feature:**

```
public static long CountRows(Table table)
{
  using (RowCursor rowCursor = table.Search())
  {
    long count = 0;

    while (rowCursor.MoveNext())
    {
      ++count;
    }

    return count;
  }
}
```

Best Practices (cont'd)

- **Explicitly disposing of Row/Feature:**

```
public static long CountRows(Table table)
{
  using (RowCursor rowCursor = table.Search())
  {
    long count = 0;

    while (rowCursor.MoveNext())
    {
      ++count;
      rowCursor.Current.Dispose(); // Need to call Dispose on the Row/Feature as well.
    }

    return count;
  }
}
```

Best Practices (cont'd)

- **Canonical form of processing a *RowCursor* :**

```
public void DoSomething()
{
  using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
  using (Table table = geodatabase.OpenDataset<FeatureClass>(SOME_TABLE))
  {
    using (RowCursor rowCursor = table.Search())
    {
      while (rowCursor.MoveNext())
      {
        using (Row row = rowCursor.Current)
        {
          //…
        }
      }
    }
  }
}
```

Best Practices (cont'd)

- **Don't create static variables for classes derived from *ArcGIS.Core.CoreObjectBase***

```
public class SomeClass
{
  private static Table table = Factory.InitializeTable();

  //…

}
```

  - **will cause *ConstructedOnWrongThreadException* because all static variables will be executed on the *GUI thread* when an add-in is first loaded by ArcGIS Pro**

Road Ahead

What is coming

- 2.0 (breaking api changes allowed)
  - Items marked Deprecated can be removed
  - Blob field support (reading and writing)
  - Create Version / Delete Version
  - Annotation support

- 2.x +
  - LongTransaction support in feature services
  - UtilityNetwork support
  - Topology feature interactions
  - Dimensions support
  - WFS datastore support

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**


esri
THE
SCIENCE
OF
WHERE