Effective Geodatabase Programming

Jochen Manegold

Gerhard Trichtl

ESRI EUROPEAN DEVELOPER SUMMIT

esri

Topic Summary

- ArcObjects and the Geodatabase API
- Unique Instancing of Objects
- Working with Cursors
- Using Caching
- Defining Schema Locks
- Top 10 developer mistakes

Purpose

- Deep insight into the architecture of the Geodatabase
- Important information for you to be an effective Geodatabase programmer
- Providing best practices, give answers to common questions
- Focus on programming patterns

Assumptions

- Familiar with ArcObjects
- Experiences in programming against the Geodatabase API
- Code examples will use C#

Agenda

- ArcObjects and the Geodatabase API
- Unique Instancing of Objects
- Working with Cursors
- Using Caching
- Defining Schema Locks
- Top 10 developer mistakes

ArcObjects and the Geodatabase API

Jochen Manegold

Data Access Objects in ArcObjects .NET 10.7 SDK (ESRI.ArcGIS.Geodatabase)

- WorkspaceFactory, Workspace
- Dataset, FeatureClass
- QueryFilter, Cursor, Feature

Data Access Objects in ArcGIS Pro 2.4 SDK (ArcGIS.Core.Data)

- Constructor, Geodatabase
- Dataset, FeatureClass, FeatureClassDefinition
- QueryFilter, RowCursor, Feature

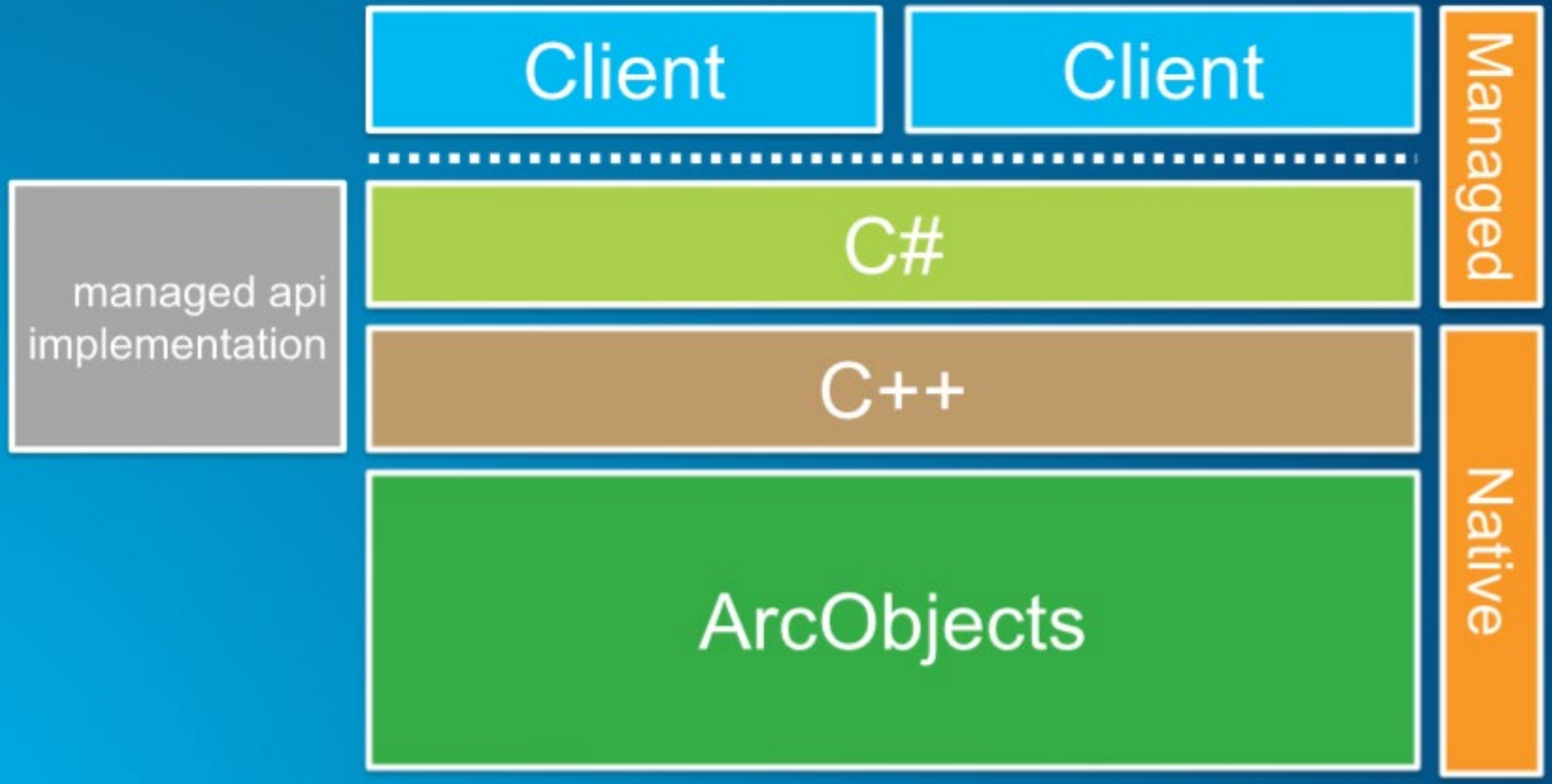
- https://github.com/Esri/arcgisprosdk/wiki
  - ProSnippets: Geodatabase
  - ProConcepts: Geodatabase

ArcGIS.Core.Data

- DML-only API (Data Manipulation Language)
- for modifying the schema the Geoprocessing API need to be performed
- uses a lot of unmanaged resources (use “using” to actively dispose and release unmanaged resources)

```
using (Geodatabase gdb = featureClass.GetDatastore() as Geodatabase)
{
    // etc.
}
```

Core.Data


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

ArcObjects SDK for Microsoft .NET Framework

- **Connecting to a file geodatabase**

```
// For example, path = @"C:\myData\myfGDB.gdb".
public static IWorkspace FileGdbWorkspaceFromPath(String path)
{
    Type factoryType = Type.GetTypeFromProgID(
        "esriDataSourcesGDB.FileGDBWorkspaceFactory");
    IWorkspaceFactory workspaceFactory = (IWorkspaceFactory)Activator.CreateInstance
        (factoryType);
    return workspaceFactory.OpenFromFile(path, 0);
}
```

ArcGIS Pro 2.4 SDK

- **Connecting to a file geodatabase**

```
public async Task OpenFileGDB()
{
  try {
    await ArcGIS.Desktop.Framework.Threading.Tasks.QueuedTask.Run(() => {
      // Opens a file geodatabase. This will open the geodatabase if the folder exists and contains a valid geodatabase.
      using (
        Geodatabase geodatabase =
          new Geodatabase(new FileGeodatabaseConnectionPath(new Uri(@"C:\Data\LocalGovernment.gdb")))) {
        // Use the geodatabase.
      }
    });
  }
  catch (GeodatabaseNotFoundOrOpenedException exception) {
    // Handle Exception.
  }

}
```

ArcObjects SDK for Microsoft .NET Framework

- **Open a Feature Class**

```
// Cast the workspace to IFeatureWorkspace and open a feature class.
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass featureClass = featureWorkspace.OpenFeatureClass("Parcels");
```

ArcGIS Pro 2.4 SDK

- **Open a Feature Class**

```
// Open a featureClass (within a feature dataset or outside a feature dataset).
using (FeatureClass featureClass = geodatabase.OpenDataset<FeatureClass>("LocalGovernment.GDB.AddressPoint")) {
}
```

Unique Instancing of Objects

Jochen Manegold

Unique Instancing of Objects

- Geodatabase Objects that will have at most one instance instantiated (similar to COM singletons)
  - WorkspaceFactory
  - Workspace, Version
  - FeatureClass
- Regardless of the API that handed out the reference, it is always the same reference
- Changes to the object affect all other references

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

- **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);
}
```

Cursor

Jochen Manegold

Cursor Types

- Class Cursor
  - ArcObjects SDK for Microsoft .NET Framework
    - Search Cursor (general queries)
    - Update Cursor (positioned updates)
    - Insert Cursor (bulk inserts)
  - ArcGIS Pro 2.4 SDK
    - RowCursor

- QueryDef Cursor
  - Evaluate Method

Differences: Rows returned by class cursors are bound to the class which created the cursor

Rows returned by class cursors are bound to the class which created the cursor

- IRow Members

Members

	Name	Description
←	Delete	Deletes the row.
■—	Fields	The fields Collection for this row buffer.
■—	HasOID	Indicates if the row has an OID.
■—	OID	The OID for the row.
←	Store	Stores the row.
■—	Table	The Table for the row.
■—■	Value	The value of the field with the specified index.

Rows returned by class cursors are bound to the class which created the cursor

- Table is bound to Row through the Search Cursor

```
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
```

- Rows from QuerDef Cursors are NOT bound to the table

```
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
```

SearchCursor

- Created by
  - ITable.Search
  - ITable.GetRow
  - ITable.GetRows
  - ISelection.Search
- When in an edit session, the query may be satisfied by a cache (spatial/feature cache, object pool)
- Rows can be modified (IRow.Store, IRow.Delete)
- When in an edit session the cursor will only flush the class' cached rows

SearchCursor

- **Following members not supported on the search cursor:**

Members

	Name	Description
←	DeleteRow	Delete the existing Row in the database corresponding to the current position of the cursor.
■←	Fields	The Fields Collection for this cursor.
←	FindField	The index of the field with the specified name.
←	Flush	Flush any outstanding buffered writes to the database.
←	InsertRow	Insert a new Row into the database using the property values in the input buffer. The object ID of the new Row, if there is one, is returned.
←	NextRow	Advance the position of the cursor by one and return the Row object at that position.
←	UpdateRow	Update the existing Row in the database corresponding to the current position of the cursor.

Update Cursor

- Created by
  - ITable.Update
  - ISelectionSWet2.Update
- Update Row by Row
  - NextRow – UpdateRow – NextRow – UpdateRow …
- Query is never be satisfied by a cache
- Resulting Rows can also be modified by ICursor.UpdateRow or ICursor.DeleteRow
  - Do not combine with IRow.Store und IRow.Delete

Update Cursor

- **Following member are not supported on the update cursor:**

Members

	Name	Description
←	DeleteRow	Delete the existing Row in the database corresponding to the current position of the cursor.
←	Fields	The Fields Collection for this cursor.
←	FindField	The index of the field with the specified name.
←	Flush	Flush any outstanding buffered writes to the database.
←	InsertRow	Insert a new Row into the database using the property values in the input buffer. The object ID of the new Row, if there is one, is returned.
←	NextRow	Advance the position of the cursor by one and return the Row object at that position.
←	UpdateRow	Update the existing Row in the database corresponding to the current position of the cursor.

Update Cursors and Store/Delete Events

- You find store events in
  - Non-simple feature types (!esriFTSimple)
  - Class participates in geometric networks or relationships that require messaging
  - Custom features (ObjectClassExtension)
- Triggered by IRow.Store or IRow.Delete
- Using an UpdateCursor with UpdateRow and DeleteRow, an internal Search Cursor is created and the methods become equivalent to IRow.Store and IRow.Delete

Best practices for Update Cursors

- 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

- Created by
  - ITable.Insert
- Primary use is for bulk inserts
- Best performance when using buffering and proper flushing while in an edit session

Insert Cursors

- **Following member are not supported on the insert cursor:**

Members

	Name	Description
←	DeleteRow	Delete the existing Row in the database corresponding to the current position of the cursor.
■←	Fields	The Fields Collection for this cursor.
←	FindField	The index of the field with the specified name.
←	Flush	Flush any outstanding buffered writes to the database.
←	InsertRow	Insert a new Row into the database using the property values in the input buffer. The object ID of the new Row, if there is one, is returned.
←	NextRow	Advance the position of the cursor by one and return the Row object at that position.
←	UpdateRow	Update the existing Row in the database corresponding to the current position of the cursor.

Insert Cursor and Store events

- Similar to Update Cursor: ICursor.InsertRow triggers ITable.CreateRow and IRow.Store

Insert Cursor and Buffering

- Call the ITable.Insert method with the argument “useBuffering” set to “True”
- Periodically call Flush
  - After 1000 rows is a good starting number
  - The higher the flush interval the less network traffic
- Try to ensure that the class has no spatial cache
  - Extra processing is required to keep the cache in sync
- Proper error handling for both InsertRow and Flush, because both can result in rows written to the database

QueryDef Cursor

- Remember: returned Rows not bound to class
- Executes a user defined query
- 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 do not support APIs to modify the row
  - Store/Delete not supported

Recycling Cursor

- 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 reused
  - 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
-

Examples

- **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
    }
  }
```

Examples

```
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
    }
  }
```

Using Recycling Cursors

- 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

Using Recycling Cursors

```
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
}
```

Non-Recycling Cursor

- 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
- Geodatabase APIs which support the creation of nonrecycling cursors have a Boolean argument
  - recycling = false creates a nonrecycling cursor

Using Non-Recycling Cursors

- 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

Documentation to the Use of Cursors

- https://desktop.arcgis.com/en/arcobjects/latest/net/webframe.htm#GdbApiMistakes.htm
-

Questions and Answers

- Q: When should I release a reference to a cursor?
- A: Do not hold cursor references if they are not needed
  - Release ASAP after fetching is completed
  - Release after application is done with the cursor

Questions and Answers

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

    // Create a sum of each geometry's area.
    IFeature feature=null;
    double totalShapeArea=0;
    while ((feature=featureCursor.NextFeature()) != null)
    {
        IArea shapeArea=(IArea)feature.Shape;
        totalShapeArea += shapeArea.Area;
    }

    Console.WriteLine("Total shape area: {0}", totalShapeArea);
}
```

Questions and Answers

- Q: If I need to use a cursor inside an edit session, where should I create the cursor?
- A: Inside the edit session, scope to edit operation

Questions and Answers

```
// 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();
```

Questions and Answers

- Q: Should I use a search cursor to update rows?
- A: Definitively YES. In fact, using search cursors within an edit session is the recommended way to update rows

Questions and Answers

```
using(ComReleaser comReleaser=new ComReleaser())
{
    // Use IFeatureClass.Search to create a search cursor.
    IFeatureCursor searchCursor=featureClass.Search(null, false);
    comReleaser.ManageLifetime(searchCursor);

    // Find the positions of the fields used to get and set values.
    int laneFieldIndex=featureClass.FindField("LANE_COUNT");
    int speedFieldIndex=featureClass.FindField("SPEED_LIMIT");
    IFeature feature=null;
    while ((feature=searchCursor.NextFeature()) != null)
    {
        // Check the lane count of the feature.
        int laneCount=Convert.ToInt32(feature.get_Value(laneFieldIndex));

        // Set the speed limit based on the lane count.
        int speedLimit=(laneCount == 2) ? 50 : 80;
        feature.set_Value(speedFieldIndex, speedLimit);
        feature.Store();
    }
}
```

Questions and Answers

- Q: If I am editing existing data, what type of cursor should I use?
- A: A Search Cursor. ArcMap/Pro – possibility that an active cache can satisfy the query and no DBMS query is required

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

Caching Geodatabase Data

Gerhard Trichtl

Spatial Cache

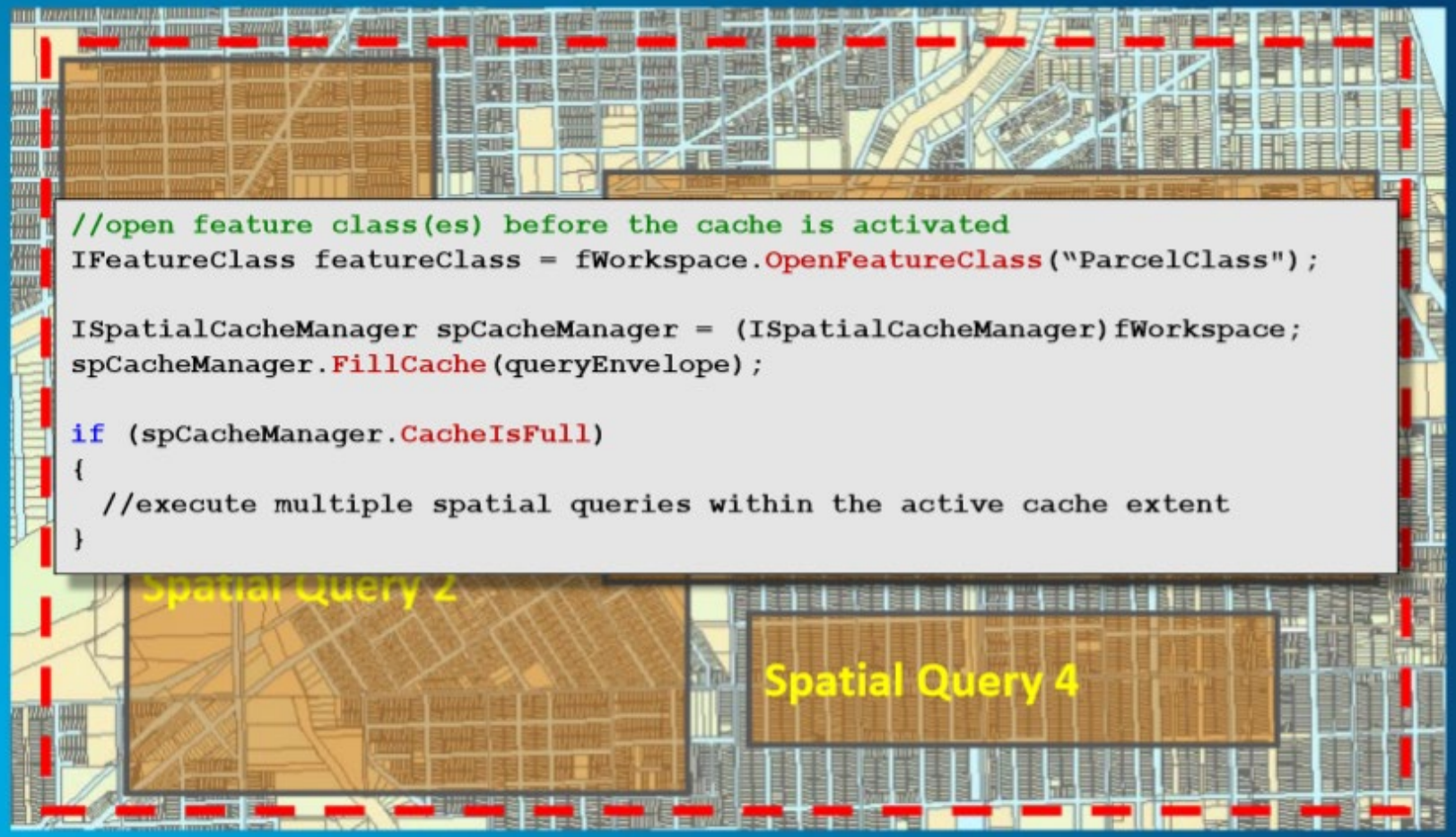
What is a Spatial Cache?

- A client-side caching of feature values over a given spatial extent
- Can speed up queries
  - Reduces roundtrips to the database
- When to use?
  - If making many spatial queries within a common extent

```
//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 a Spatial Cache?

ArcGIS Developer Help

ISpatialCacheManager Interface

Provides access to members that control the Spatial Cache Management. **Note:** the ISpatialCacheManager interface has been superseded by ISpatialCacheManager3. 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

	Name	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.

Classes that implement ISpatialCacheManager

Classes	Description
VersionedWorkspace	VersionedWorkspace Object.
Workspace	Workspace Object.

Schema Cache

What is a Schema Cache?

- A cached snapshot of the Geodatabase schema
- Can improve performance when opening datasets by reducing database round trips
- Requires a static data model
- ArcMap use it during OpenDocument and Reconcile

When to use?

- 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

Documentation

- https://desktop.arcgis.com/en/arcobjects/latest/net/webframe.htm#LeverageTheSchemaCache.htm

ArcGIS Developer Help

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

	Name	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.

Classes that implement IWorkspaceFactorySchemaCache

Classes	Description
SdeWorkspaceFactory (esriDataSourcesGDB)	Esri SDE Workspace Factory.

Using Schema Caching

- 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)

```
IWorkspaceFactorySchemaCache workspaceFactorySchemaCache=
    (IWorkspaceFactorySchemaCache)workspaceFactory;

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

workspaceFactorySchemaCache.EnableSchemaCache(workspace);
```

Using Schema Caching

- Cache needs to be “fresh”
  - If in dynamic environments schema changes will not be visible until cache is refreshed

```
 //Spatial cache refresh if stale.
if (workspaceFactorySchemaCache.IsSchemaCacheStale(workspace))
{
    workspaceFactorySchemaCache.RefreshSchemaCache(workspace);
}
```

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

```
workspaceFactorySchemaCache.DisableSchemaCache(workspace);
```

Schema Locks

Gerhard Trichtl

Schema Locks

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

When to use Schema Locks?

- 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

Using 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

Top 10 Developer Mistakes

Jochen Manegold

Top 10 Developer Mistakes

1. Misusing recycling cursors
2. Calling FindField in a Loop
3. Using Cursors across edit operations
4. Calling DLL commands in an edit session
5. Calling Store in a Store-triggered event
6. Using GetFeature versus GetFeatures
7. Relying on name objects for caching
8. Careless reuse of variables
9. Changes to non-persisting schema objects
10. Making Inserts with relationship class notification

http://resources.esri.com/help/9.3/ArcGISEngine/dotnet/09bd8059-f031-4b88-bac8-3b4b73dccb05.htm

1. Misusing recycling cursors

- If recycling is enabled, a cursor only allocates memory for a single row regardless of how many rows are returned from the cursor
  - Performance benefits in terms of memory usage and running time, but has drawbacks for certain workflows

1. Misusing recycling cursors

```
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);
}
```

What is the right value for enableRecycling: True or False?

2. 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**
- **Avoid:**

```
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")));
  }
}
```

2. Calling FindField in a Loop

- **Use:**

```
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;
  int nameIndex = featureClass.FindField("NAME");

  // Make sure the FindField result is valid.
  if (nameIndex ==  - 1)
  {
    throw new ArgumentException("The NAME field could not be found.");
  }

  while ((feature = featureCursor.NextFeature()) != null)
  {
    Console.WriteLine(feature.get_Value(nameIndex));
  }
}
```

3. Using Cursors across edit operations

- **Avoid this:**

```
// Start an edit session and edit operation.
using(ComReleaser comReleaser = new ComReleaser())
{
  IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
  workspaceEdit.StartEditing(true);
  workspaceEdit.StartEditOperation();

  // Create a new insert cursor and feature buffer.
  IFeatureCursor featureCursor = featureClass.Insert(true);
  comReleaser.ManageLifetime(featureCursor);
  IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
  comReleaser.ManageLifetime(featureBuffer);

  // Insert a feature.
  featureBuffer.Shape = new PointClass
  {
    X = 5, Y = 10
  };
  featureCursor.InsertFeature(featureBuffer);

  // Abort the edit operation and start a new one.
  workspaceEdit.AbortEditOperation();
  workspaceEdit.StartEditOperation();

  // Insert another feature.
  featureBuffer.Shape = new PointClass
  {
    X = 10, Y = 5
  };
  featureCursor.InsertFeature(featureBuffer);

  // Commit the edit operation and stop editing.
  workspaceEdit.StopEditOperation();
  workspaceEdit.StopEditing(true);
}
```

Blocked since 10.0

3. Using Cursors across edit operations

- **Use:**

```
// Start an edit session and edit operation.
IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
workspaceEdit.StartEditing(true);
workspaceEdit.StartEditOperation();

// Create a new insert cursor and feature buffer.
using(ComReleaser comReleaser = new ComReleaser())
{
  IFeatureCursor featureCursor = featureClass.Insert(true);
  comReleaser.ManageLifetime(featureCursor);
  IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
  comReleaser.ManageLifetime(featureBuffer);

  // Insert a feature.
  featureBuffer.Shape = new PointClass
  {
    X = 5, Y = 10
  };
  featureCursor.InsertFeature(featureBuffer);
}

// Abort the edit operation and start a new one.
workspaceEdit.AbortEditOperation();
workspaceEdit.StartEditOperation();
```

```
// Create another insert cursor and feature buffer.
using(ComReleaser comReleaser = new ComReleaser())
{
  IFeatureCursor featureCursor = featureClass.Insert(true);
  comReleaser.ManageLifetime(featureCursor);
  IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
  comReleaser.ManageLifetime(featureBuffer);

  // Insert another feature.
  featureBuffer.Shape = new PointClass
  {
    X = 10, Y = 5
  };
  featureCursor.InsertFeature(featureBuffer);
}

// Commit the edit operation and stop editing.
workspaceEdit.StopEditOperation();
workspaceEdit.StopEditing(true);
```

4. Calling DDL commands in an edit session

- Methods that trigger DDL commands, such as IFeatureWorkspace.CreateTableor 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
  - also modifications to domains behave like true DDL commands

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

5. Calling Store in a Store-triggered event

- 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
- Example

```
private static void EventHandlerInitialization(IFeatureClass featureClass)
{
  IObjectClassEvents_Event objectClassEvents = (IObjectClassEvents_Event)
    featureClass;
  objectClassEvents.OnCreate += new IObjectClassEvents_OnCreateEventHandler
    (OnCreateHandler);
}

private static void OnCreateHandler(IObject obj)
{
  obj.set_Value(NAME_INDEX, Environment.UserName);
  obj.Store(); // Do NOT do this!
}
```

6. 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
- Avoid this:

```
int nameFieldIndex = featureClass.FindField("NAME");
foreach (int oid in oidList)
{
  IFeature feature = featureClass.GetFeature(oid);
  Console.WriteLine("NAME: {0}", feature.get_Value(nameFieldIndex));
}
```

6. Using GetFeature versus GetFeatures

- **Use GetFeatures from the GeoDatabaseHelperClass Class**

```
private static void GetFeaturesExample(IFeatureClass featureClass, int[]
  oidList)
{
  int nameFieldIndex = featureClass.FindField("FIPSSTCO");
  using(ComReleaser comReleaser = new ComReleaser())
  {
    IGeoDatabaseBridge geodatabaseBridge = new GeoDatabaseHelperClass();
    IFeatureCursor featureCursor = geodatabaseBridge.GetFeatures(featureClass,
      ref oidList, true);
    comReleaser.ManageLifetime(featureCursor);

    IFeature feature = null;
    while ((feature = featureCursor.NextFeature()) != null)
    {
      Console.WriteLine("NAME: {0}", feature.get_Value(nameFieldIndex));
    }
  }
}
```

7. 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

8. Careless reuse of variables

```
// Create a new 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;
```

8. Careless reuse of variables

```
// Create a new 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;
```

- In this case you try to add to identical fields
- Possible solutions:
  - Reassign the field and fieldEdit variables to a newly created field object after each field is added
  - Use a separate set of variables for each field that will be added to the collection, i.e. "oidField" and "oidFieldEdit"

8. Careless reuse of variables

```
// Execute a query...
IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
IFeature feature = null;
while ((feature = featureCursor.NextFeature()) != null)
{
  // Do something with the feature...
}

// Re-execute the query...
featureCursor = featureClass.Search(queryFilter, true);
feature = null;
while ((feature = featureCursor.NextFeature()) != null)
{
  // Do something with the feature...
}

// Release the cursor.
Marshal.ReleaseComObject(featureCursor);
```

- **In this case you lose all references to objects that should be explicitly released using the ComReleaser class or the Marshal.ReleaseComObject method (the first feature cursor)**
- **Lifetime management is object-specific, not variable-specific.**

8. Careless reuse of variables

```
using(ComReleaser comReleaser = new ComReleaser())
{
  // Execute a query...
  IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
  comReleaser.ManageLifetime(featureCursor);
  IFeature feature = null;
  while ((feature = featureCursor.NextFeature()) != null)
  {
    // Do something with the feature...
  }

  // Re-execute the query...
  featureCursor = featureClass.Search(queryFilter, true);
  feature = null;
  while ((feature = featureCursor.NextFeature()) != null)
  {
    // Do something with the feature...
  }
}
```

- in this example, only the first cursor is properly managed

9. 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, f.e. table
  - 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

10. Inserts and relationship class notification

- Notification (also known as messaging) ensures the proper behavior of composite relationships, featurelinked 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

10. Inserts and relationship class notification

```
// Open the class that will be edited.
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;
IFeatureClass featureClass = featureWorkspace.OpenFeatureClass("PARCELS");
ITable table = featureWorkspace.OpenTable("OWNERS");

// Begin an edit session and operation.
IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
workspaceEdit.StartEditing(true);
workspaceEdit.StartEditOperation();

// Create a search cursor.
using(ComReleaser comReleaser = new ComReleaser())
{
  IFeatureCursor featureCursor = featureClass.Insert(true);
  comReleaser.ManageLifetime(featureCursor);
  IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
  comReleaser.ManageLifetime(featureBuffer);

  for (int i = 0; i < 1000; i++)
  {
    featureBuffer.Shape = CreateRandomPolygon();
    featureCursor.InsertFeature(featureBuffer);
  }

  featureCursor.Flush();
}

// Commit the edits.
workspaceEdit.AbortEditOperation();
workspaceEdit.StopEditing(false);
```

Top 10 Developer Mistakes

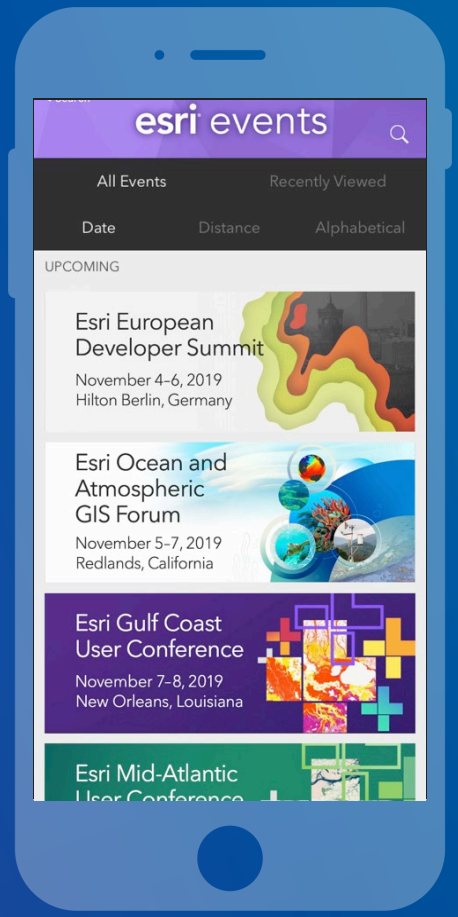
1. Misusing recycling cursors
2. Calling FindField in a Loop
3. Using Cursors across edit operations
4. Calling DLL commands in an edit session
5. Calling Store in a Store-triggered event
6. Using GetFeature versus GetFeatures
7. Relying on name objects for caching
8. Careless reuse of variables
9. Changes to non-persisting schema objects
10. Making Inserts with relationship class notification

http://resources.esri.com/help/9.3/ArcGISEngine/dotnet/09bd8059-f031-4b88-bac8-3b4b73dccb05.htm

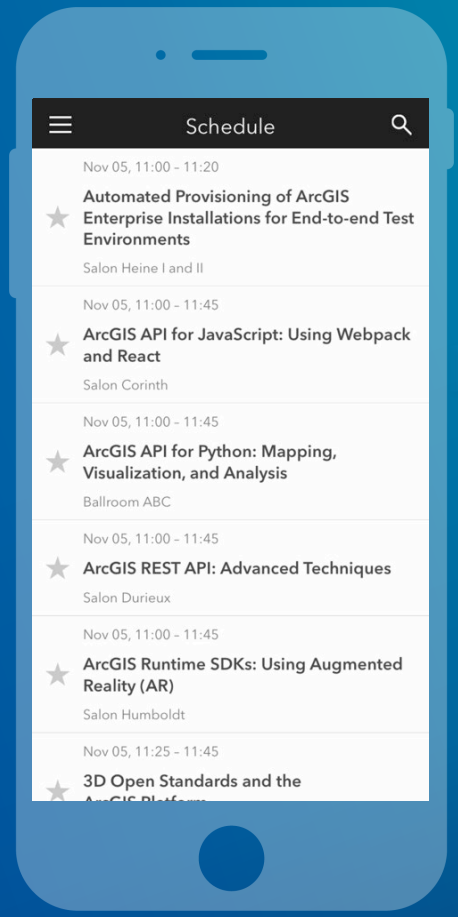
Questions?

Please Take Our Survey on the App

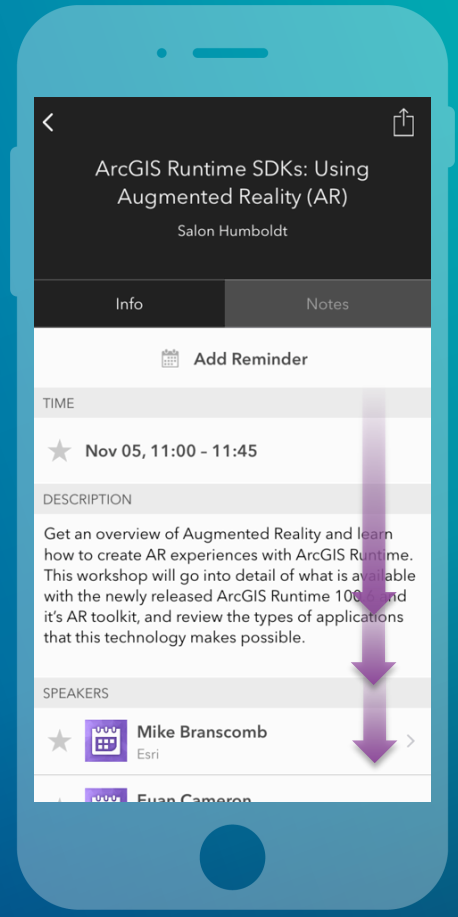
Download the Esri Events app and find your event



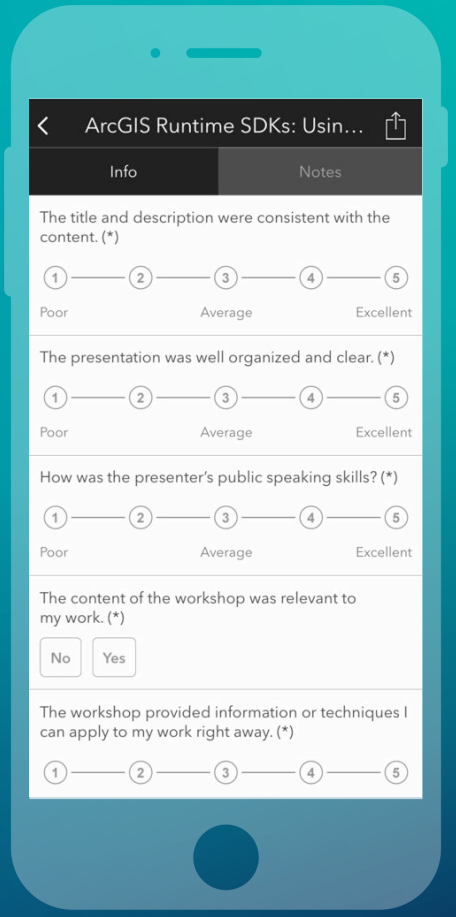
Select the session you attended



Scroll down to find the feedback section



Complete answers and select "Submit"



esri

THE SCIENCE OF WHERE