



Turbocharged Geodatabase Programming

Erik Hoel and Mike Muller

The Assumptions

- **Good working knowledge of the Geodatabase**
- **Experience in programming against the GDB API**
- **Code examples will use Java, C#, or C++**
 - Readily translatable to other languages (e.g., VB)
- **Way too much content, very little time**
 - Slides are a priority queue
 - Talk fast, skip explanations
 - Save questions for the end or for the follow-on Tech-Talk session in the Oasis room
- **Slides intended to be later used by you as a reference**

The 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 GDB development team) conceptualize the architecture**
- **General focus areas:**
 - **Architectural**
 - **Performance and scalability**

The Outline

- **Implementation Architecture**
- **Object Class Properties**
- **Validating Data**
- **Dataset Extensions**
- **Event Model**
- **Cursors**
- **Selection Sets**
- **Unique Instancing of Objects**
- **Junk Yard**



Implementation Architecture

Implementation Architecture

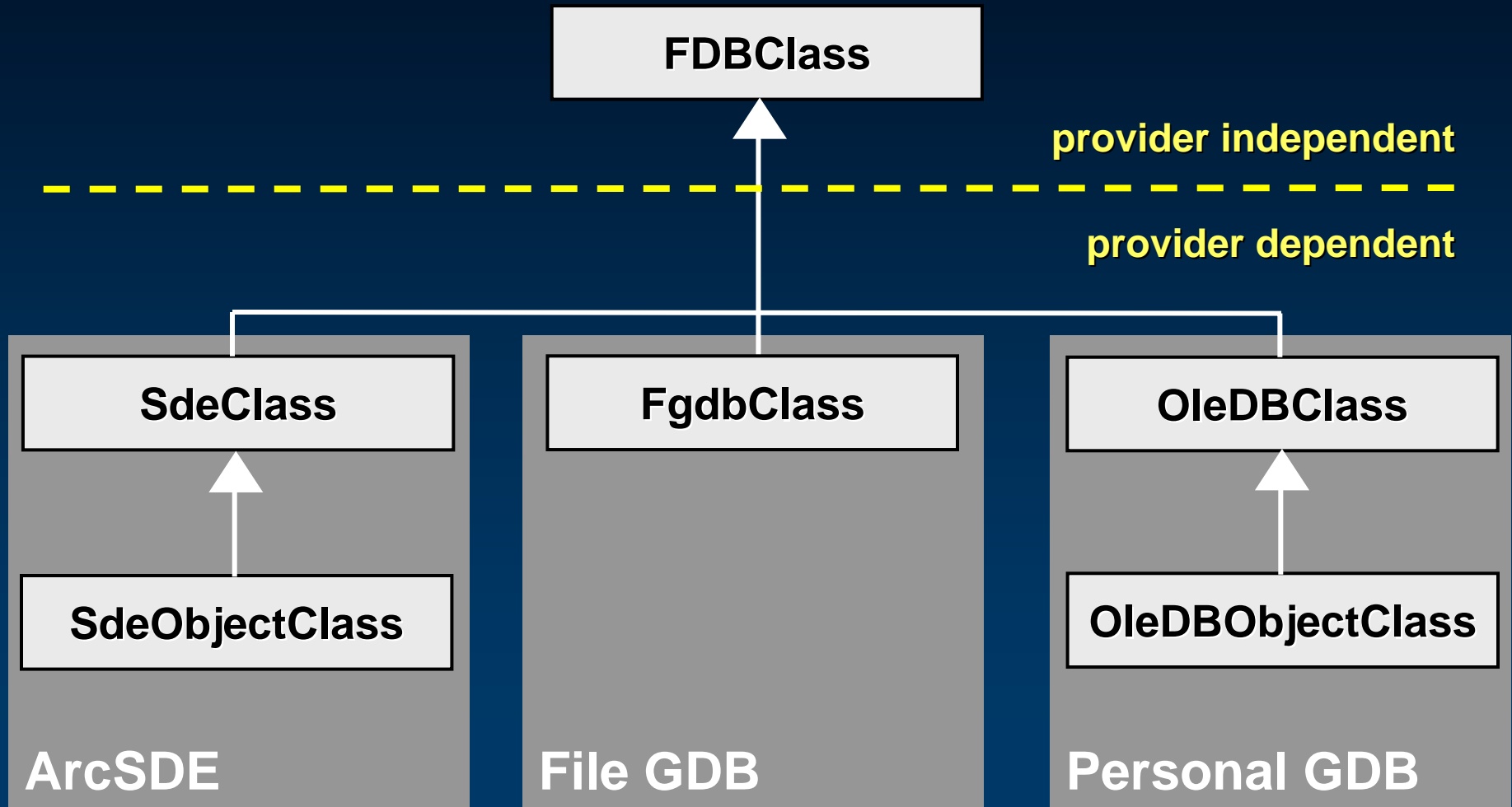
- **Geodatabase is composed of many DLLs**
- **At the lowest level, it utilizes three providers:**
 - ArcSDE Geodatabases - SDE C-API
 - Personal Geodatabases - Microsoft Jet (via DAO)
 - File Geodatabases -
- **Components are implemented differently depending upon knowledge required of the underlying provider**
- **Items that are not shared between providers include:**
 - Cursors, QueryDefs

Implementation Architecture

- **Many classes that implement datasets or other key abstractions are partially shared across data provider implementations:**
 - Workspaces, Feature Datasets, Feature Classes, Relationship Classes
- **Other datasets and abstractions are implemented in a manner that is completely provider independent:**
 - Geometric Networks, Topologies, Network Datasets
 - Domains, Rules, Data Elements, Names, Logical Network, Topology Graph

Implementation Architecture

Example Hierarchy: Feature Classes





Object Class Properties

Object Class Properties

Overview

- There are four properties of an object class that control its behavior
 - requiresStoreMethod
 - requiresEditSession
 - requiresLongEditSession
 - canEditWithProjection
- The **requiresStoreMethod** property
 - The Store() method is required for polymorphism
 - Examples (complex features):
 - Network features
 - Dimension features
 - Annotation features

Object Class Properties

Overview

- The **requiresEditSession** property
 - Edit sessions are required for the running object table and long transactions
 - In the absence of an edit session applications would have to use short transactions and discard all object state across short transaction boundaries
 - Edit sessions are required for correct handling of composite relationships during update and for relationship message propagation
 - Examples:
 - Features participating in a topology
 - Features participating in a composite relationship or a relationship with notification
 - Network features

Object Class Properties

Overview

- The **requiresLongEditSession** property
 - If true, then the behavior of this class requires that all changes to it be made within a long transaction (i.e., high isolation) edit session
 - Examples:
 - Features in a topology
 - Network features
- The **canEditWithProjection** property
 - Whether the class can be edited in a spatial reference different from the class
 - Examples (where property is **false**):
 - Network features

Object Class Properties

Setting

- **These properties are set by**
 - Examining the feature type during class initialization
 - Probing the optional class extension (r.e., *IObjectClassInfo2*)
- **If the feature class participates in a topology, then it will require an edit session for all updates**
 - edits outside of an edit session are disallowed as we would not be able to recover from a crash in a bulk update operation outside of an edit session (i.e., dirty areas would not be created)
- **Network features always require both the store method and edit sessions**
- **If the object class participates in composite relationships or relationships with messaging then both the store method and edit sessions are required**

Object Class Properties

Class Extensions

- **A class extension may tighten the default policies** by implementing *IObjectClassInfo2* or *IFeatureClassEdit*
- **A class extension may not relax the default policies** (e.g., a custom network feature cannot choose to not require an edit session)
- The *IObjectClassInfo2* interface is implemented to indicate whether a class required `Store()` to be called when inserting or updating features, even when insert or update cursors are used
 - `CanBypassStoreMethod(VARIANT_BOOL* pCanBypass)`
 - `CanBypassEditSession(VARIANT_BOOL* pCanBypass)`
- The *IFeatureClassEdit* interface is used for specifying advanced editing configuration
 - You can control whether the class can be edited in a spatial reference different from the class definition



Validating Data

Validating Data

- **Why we allow invalid data when loading?**
 - Do not want data to disappear when loading
 - Some storage representations allowed invalid geometries
- **What is the best way to validate data**
 - Ephemeral topologies
 - Validation rules
 - Class extensions
 - Editor events
- **When do you want to validate your data**
 - After loading, prior to versioning
 - After loading, prior to users creating new versions off **DEFAULT**
 - Before posting edits to **DEFAULT** (r.e., QA version)

Validating Data

- **Validate*() methods**
 - Found on object classes in the *IValidation* interface
 - Validate by entire object class, selection set, query filter, or set of features
 - Validation of domains, relationship rules, and connectivity rules
 - Quick abort logic
- **Validation order:**
 - Subtype
 - Attribute rules
 - Network connectivity
 - Custom rules (r.e., class extension)
 - Relationship rules

Validating Data

- **Custom validation of Objects**
 - Users may augment this by creating class extensions supporting the *IObjectClassValidation* interface
 - After successfully completing native validation of subtypes, attribute and connectivity rules, the `ValidateRow()` method is called



Dataset Extensions

Dataset Extensions

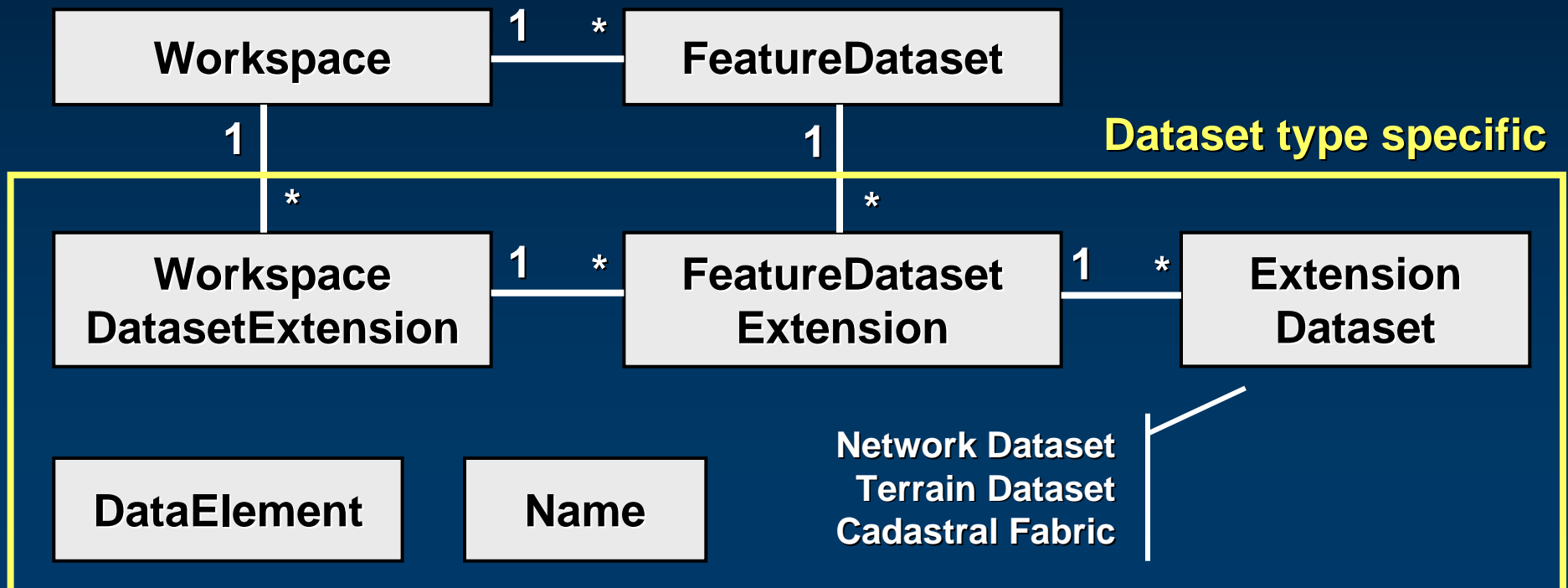
Overview

- Existing public Geodatabase extension mechanisms
 - Class Extensions
 - Workspace Extensions
- Introduced at 9.1 with the Network Dataset
- Internal architectural pattern for adding new dataset types to the Geodatabase
 - No need to touch the GDB kernel code
 - Four newest datasets implemented using this mechanism
 - Network datasets
 - Terrain datasets
 - Representation classes
 - Cadastral fabrics
 - More to follow

Dataset Extensions

Basic Model

- Utilizes an enhanced workspace extension operating with a new dataset, feature dataset extension (optional), data element, and name object



Dataset Extensions

Navigation

- Simplifies end user access patterns
- Navigation becomes generic, no need to use dataset type specific interfaces; e.g.,
 - IObjectClassContainer
 - IFeatureClassContainer
 - IRelationshipClassContainer
 - INetworkCollection2
 - ITopologyContainer
- Instead, use small group of polymorphic interfaces
 - IFeatureDatasetExtensionContainer on the FD
 - IDatasetContainer3 on the FDX

Example: Get a Network Dataset by Name

```
HRESULT GetNetworkByName(IWorkspace* pWorkspace, BSTR fdName, BSTR ndName, IDataset** ppDataset)
{
    // For the workspace and named feature dataset, return the network dataset with the specified name.

    if (!ppDataset)
        return E_POINTER;

    *ppDataset = 0;

    // Get the feature dataset extension containing the network.

    HRESULT hr;
    IFeatureDatasetPtr ipFD;
    IFeatureDatasetExtensionPtr ipFDX;

    1 if (FAILED(hr = ((IFeatureWorkspacePtr) pWorkspace)->OpenFeatureDataset(fdName, &ipFD)))
        return hr;
    2 if (FAILED(hr = ((IFeatureDatasetExtensionContainerPtr) ipFD)->FindExtension(esriDTNetworkDataset, &ipFDX)))
        return hr;

    // Return the network dataset with the specified name.

    3 return ((IDatasetContainer2Ptr) ipFDX)->get_DatasetByName(esriDTNetworkDataset, ndName, ppDataset);
}
```

Dataset Extensions

Creation and Schema Updates

- **Creating a dataset or modifying the schema use a polymorphic interface in addition to a data element**
- **Data elements are key concept to new datasets**
 - Used to completely specify the definition of the dataset
 - Also used when modifying the schema of a dataset
- **Standard creation pattern:**
 1. **Cocreate appropriate data element**
 2. **Configure data element properties**
 3. **Navigate to appropriate feature dataset extension**
 4. **Call `IDatasetContainer3::CreateDataset()` passing in the data element; an `IDataset` reference is returned**
- **Reliance upon this pattern will grow**



Event Model

Event Model

Overview

- **Types of events**
 - Feature events
 - Object class events
 - Class extension events
 - Workspace events
- **Feature events and class extension events only apply to implementers of custom features and class extensions**
- **Object class and workspace events may be listened to by client code**

Event Model

Overview

- **Some events may either be listened to or may be supported in object class extensions; e.g.,**
 - **IObjectClassEvents**
 - **ITopologyClassEvents**

Event Model

Custom Features - IRowEvents

- Allows custom Row objects to take special action in response to changes made to the state of a Row
- Methods:
 - **OnChanged** – first functionality in Store() on an existing Row (before call to IObjectClassEvents::OnChange())
 - **OnDelete** – first functionality in Delete(), before Row is logically deleted
 - **OnInitialize** - after hydrating and setting the field values of a new Row object but before handing the Row to the client
 - This is an opportunity for the Row object to initialize further state and derived member variables
 - **OnNew** – first functionality in Store() on a newly created Row (before any updates to the spatial cache, etc.)
 - **OnValidate** - unsupported (deprecated)

Event Model

Detailed Example 1: Store() on a new Feature

1. Feature marked as not being new
2. Custom features (incl. dimensions, annotation)
 - IRowEvents::OnNew
3. Class extensions
 - IObjectClassEvents::OnCreate
4. Listeners to IObjectClassEvents
 - IObjectClassEvents::OnCreate
5. Class extensions (if relationship class, etc.)
 - IRelatedObjectClassEvents::RelatedObjectCreated
6. Update the spatial cache if in edit session

Event Model

Detailed Example 2: Delete() on a Feature

1. Custom features (incl. dimensions, annotation)
 - IRowEvents::OnDelete
2. Class extensions
 - IObjectClassEvents::OnDelete
3. Listeners to IObjectClassEvents
 - IObjectClassEvents::OnDelete
4. Delete part objects if in composite relationship
5. Mark the Feature as deleted

Event Model

Custom Features - IRelatedObjectEvents

- For custom Objects; allows Object to respond events on related Objects
 - e.g., if a change in the attribute of a related Object needs to trigger a change in the attributes of this Object
- Methods:
 - **RelatedObjectChanged** – allows a feature in a composite relationship to update its position when related feature changes position
 - RelatedObjectMoved
 - RelatedObjectRotated
 - RelatedObjectSetMoved
 - RelatedObjectSetRotated
- Set also IConfirmRelatedObjectEvents

Event Model

Custom Features - IFeatureEvents

- For custom Features; an optional interface (IRowEvents complements this interface)
- Methods:
 - **InitShape** – unsupported (deprecated)
 - **OnMerge** – unsupported
 - **OnSplit** - called when a Feature is split, before
 - the feature's geometry is updated, and
 - The feature is either deleted (delete-create-create model) or updated (update-create model)

Event Model

Custom Features

- **INetworkFeatureEvents**
 - **OnConnect** - first functionality in **Connect()** on an existing **NetworkFeature** (unsupported)
 - **OnDisconnect** - first functionality in **Disconnect()** on an existing **NetworkFeature** (unsupported)

Event Model

Object Classes – IObjectClass[Schema]Events

- Listenable event interfaces on the object class
- IObjectClassEvents
 - Used for monitoring changes to the Objects in the ObjectClass
 - Also supported on class extensions
 - Class extensions messaged first
 - Methods are called before notifying other related and external objects
 - OnCreate
 - OnChange
 - OnDelete
- IObjectClassSchemaEvents (post events)
 - OnAddField
 - OnDeleteField
 - OnBehaviorChanged

Example: Listening to Events

```
using ESRI.ArcGIS.Geodatabase;
```

1

```
class EventListener  
{  
    public IVersionEvents_OnReconcileEventHandler onReconcileEvent;
```

2

```
    public void SetupVersionEvents(IVersion version)  
    {  
        IVersionEvents_Event versionEvent = (IVersionEvents_Event) version;  
        //Wire OnReconcile event.  
        onReconcileEvent = new IVersionEvents_OnReconcileEventHandler(versionEvent_OnReconcile);  
        versionEvent.OnReconcile += onReconcileEvent;  
    }
```

3

```
    //Define event handler method.  
    private void versionEvent_OnReconcile(string targetVersionName, bool HasConflicts)  
    {  
        System.Windows.Forms.MessageBox.Show(" Reconcile against " + targetVersionName);  
    }  
}
```

From the the driver application, start listening to the version event by creating a new EventListener object and calling the SetupVersionEvents() method passing in a reference to the IVersion object.

4

```
EventListener eventListener = new EventListener();  
eventListener.SetupVersionEvents(version);
```

Event Model

Class Extensions – IRelatedObjectClassEvents[2]

- Interfaces on class extensions that receive messages about objects in related classes
 - Can create relationship instances, etc.
- Object and relationship class passed as arguments
- Methods:
 - RelatedObjectCreated
 - RelatedObjectChanged
 - RelatedObjectMoved
 - RelatedObjectRotated
 - RelatedObjectSetMoved
 - RelatedObjectSetRotated

Event Model

Class Extensions – **IConfirmSendRelatedObjectEvents**

- Interface on class extensions that is used to confirm the messaging of related Objects to optimize editor
 - When an Object that participates in a Relationship is modified, its related Objects will be messaged if relationship notification is set in that direction (r.e., *IRelatedObjectEvents*)
 - Typically, a related Object is only interested in certain changes on an Object (e.g., a particular field is modified)
 - The properties of this interface allow an *ObjectClassExtension* to prevent or confirm that messages should be sent

Event Model

Feature Classes - ITopologyClassEvents

- Listenable interface providing access to the `OnValidate()` event which is fired each time a dirty area is validated in the topology in which the class is participating
- **Also supported on class extensions**
 - Class extensions messaged first
- **Methods:**
 - **OnValidate** – last functionality in `ValidateTopology()` that returns an *IGeometry* object corresponding the area that was validated
- An alternative is to catch the *ITopologyExtensionEvents* event interface
 - This is dependent upon ArcMap and the Topology extension

Event Model

Relationship Classes - IRelationshipClassEvents

- A listenable event interface on both the RelationshipClass and the AttributedRelationshipClass
- This interface provides information as to when two objects are related or unrelated and when attributes on an attributed relationship are modified
- Methods:
 - OnChange
 - OnCreate
 - OnDelete
- Note: if a user creates or deletes a relationship instance via a put_Value() call on the foreign key in the destination object, no events are fired

Event Model

Workspace Events - IWorkspaceEditEvents

- **Listenable event interfaces on the workspace**
- **Methods:**
 - **OnStartEditing**
 - **OnStopEditing**
 - **OnStartEditOperation**
 - **OnStopEditOperation**
 - **OnAbortEditOperation**
 - **OnUndoEditOperation**
 - **OnRedoEditOperation**
- **It is important that in response to these events, clients should discard or refresh cached row objects within the application (r.e., maintaining undo/redo stack)**

Event Model

Workspace Events – IWorkspace[Replica]Events

- Listenable event interfaces on the workspace
- IWorkspaceEvents
 - OnCreateDataset
 - OnRenameDataset
 - OnDeleteDataset
- IWorkspaceReplicaEvents are fired after extracting data or schema in a checkout or a child replica
 - AfterCreateChildReplica – after the replica operation in the master GDB ends
 - BeforeCreateChildReplica – after the replica operation begins

Event Model

Workspace Events – IVersionEvents

- **Listenable event on the VersionedWorkspace**
- **Methods:**
 - **OnConflictsDetected** - fired during reconciliation, after conflicts are detected; it can be used to filter found conflicts
 - **OnReconcile** - fired after the version is reconciled, associating it with a new database state
 - **OnRefreshVersion** - fired after the version is refreshed, associating it with a new database state
 - **OnRedefineVersion** - fired after the version is changed in place to represent a different version, associating it with a new database state

Event Model

Workspace Events – IVersionEvents2

- **Listenable event on the VersionedWorkspace**
- **Augments the collection found in IVersionEvents**
- **Methods:**
 - **OnArchiveUpdated** - fired after the historical archive has been updated with changes saved or posted to the DEFAULT version
 - **OnBeginReconcile** - fired before a version is reconciled (compliments OnReconcile())
 - **OnPost** - fired after a version is posted
 - **OnDeleteVersion** - fired before a version is deleted



Cursors

Cursors

Cursor Types

- **Three Class Cursors**
 - Search (general query cursor)
 - Update (positioned update cursor)
 - Insert (bulk inserts)
- **One QueryDef Cursor**
 - User defined query (e.g. `IQueryDef.Evaluate`)
- **What's the difference?**
 - Rows created by class cursors are bound to the class which created the cursor
 - Rows created by querydef cursors are not bound to a class

Cursors

Example: Cursor Types

```
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
    }
}
```

Cursors

Class Cursors: Search

- 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 cache, object pool)
- Use within an edit session will only flush the class' cached rows
- Resulting rows can be modified
 - Store supported
 - Delete supported

Cursors

Class Cursors: Update

- Positional update cursor
 - NextRow->UpdateRow ... 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 ICursor.DeleteRow (should not be combined with Store and Delete)

Cursors

Class Cursors: Update (continued)

- If the Class supports Store events, an internal object cursor is created and UpdateRow and DeleteRow become equivalent to Row.Store and Row.Delete
 - Non-simple feature types (! esriFTSimple)
 - Class participates in Geometric Network
 - Class participates in Relationships that require messaging
 - Custom Features
 - ObjectClassExtension overrides
- UpdateRow Method Behaviors
 - Call with a row that was not retrieved from the update cursor – undefined (e.g. some datasources ignore, some error)
 - Call with a row from the update cursor but not at the current position – undefined (most datasources error)

Cursors

Class Cursors: Insert

- 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, an internal object cursor is created and `InsertRow` becomes equivalent to `Table.CreateRow` and `Row.Store` (same rules as Update)

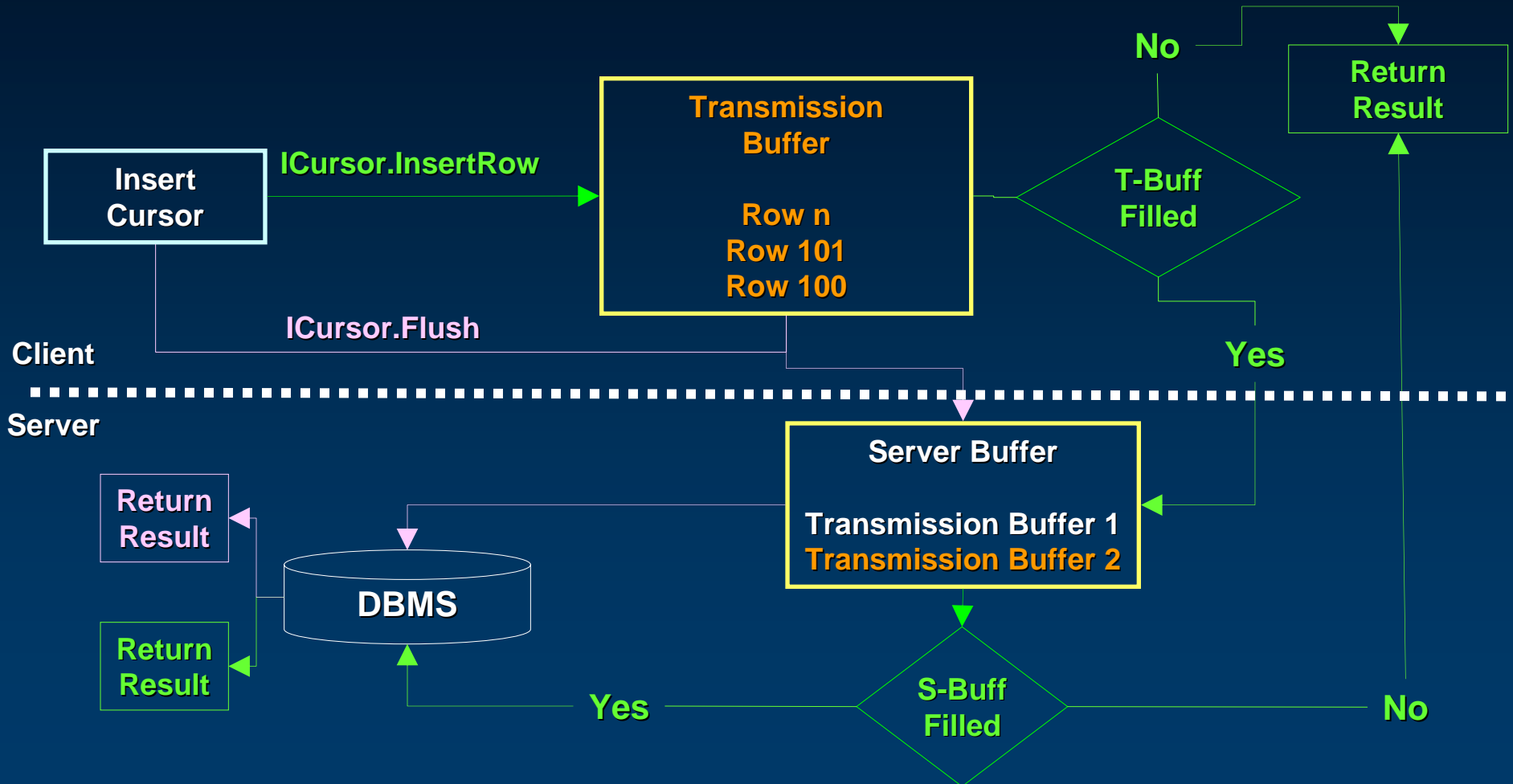
Cursors

Class Cursors: Insert - 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 – use profiling tools if performance is poor
 - Can depend on how much re-insertion is acceptable if an error is encountered
- Crucial that calls to `InsertRow` and `Flush` have the proper error handling since both can result in rows written to the database
- Try to ensure that the class has no spatial cache – extra processing is required to keep the cache in synch

Cursors

Class Cursors: Insert – Buffering (Enterprise)



Cursors

QueryDef Cursors

- **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 modify the row**
 - **Store not supported**
 - **Delete not supported**



Recycling Cursors

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 method argument
 - recycling = true creates a “recycling cursor”

Recycling Cursors

Common interfaces with methods that can create recycling cursors

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

Recycling Cursors

You don't always get what you want

- Recycling argument is a suggestion and may be ignored
 - Input filter is spatial and requires client-side topo engine filtering
 - Active spatial cache and the query can be completely satisfied by the cache
- Developers should always assume they are working with a recycling cursor if one was requested

Recycling Cursors

All calls to NextRow/NextFeature 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

Row values which are objects may also be the same instance

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

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

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

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

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

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

Recycling Cursors

Use Cases

- Use recycling cursors when references to the current row and its values do not need to be kept beyond the next call to NextRow/NextFeature
- Isolate use of references to the local method which created the recycling cursor to minimize potential bugs (i.e. do not pass the references around as some other method may decide to hold it)
- Proper use within an edit session can dramatically reduce resource consumption

Recycling Cursors

Use Cases (cont.): Example of edit session 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(
        esriMultiuserEditMode.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 72 fields
 - run(workspace, true) ~60MB memory
 - run(workspace, false) ~185MB memory (**ObjectPool**)



Non-Recycling Cursors

Non-Recycling Cursors

What are they?

- A “non-recycling cursor” is 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 method argument
 - recycling = false creates a “non-recycling cursor”

Non-Recycling Cursors

Use Cases

- Use non-recycling cursors when references to the current row and its values are needed beyond the next call to NextRow/NextFeature
- Commonly used to cache sets of rows (long lived references)
- Some Geodatabase APIs require sets of rows – should be retrieved as non-recycled rows



Non-Recycling Cursors, SubFields, and Editing

Non-Recycling Cursors, SubFields, and Editing

Developer Considerations

- The system may override the SubFields of a QueryFilter used to create a non-recycling cursor resulting in all Fields being fetched
 - The class is being edited
 - The class has an active spatial cache
- Developers can avoid complex logic for building a SubFields QueryFilter property if it is known the class will be editing

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Not Overriden

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

    IQueryFilter filter = new QueryFilter();
    filter.setSubFields("PARCEL_ID, PARCELKEY"); <-- filter requests 2 fields

    ITable testTable = workspace.openTable("parcels");
    ICursor cursor = testTable.ITable_search(filter, false); <<--non-recycling

    IFields fieldSet = cursor.getFields();
    int fieldCount = fieldSet.getFieldCount();

    System.out.println("There are " + fieldCount + " fields.");

    IRow firstRow = cursor.nextRow();

    for (int i = 0; i < fieldCount; i++) {
        Object obj = firstRow.getValue(i);

        if (obj != null) {
            System.out.println("Field value " + i + " has a class type of " + obj.getClass().toString());
        }
    }
}
```

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Not Overriden

- The filter requested 2 fields
- The workspace was not editing
- The class was not being edited
- 2 fields were fetched

Output:

```
There are 7 fields.
```

```
Field value 1 has a class type of class java.lang.String
```

```
Field value 2 has a class type of class java.lang.Integer
```

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Overriden

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

    IQueryFilter filter = new QueryFilter();
    filter.setSubFields("PARCEL_ID, PARCELKEY"); <<-- filter requests 2 fields

    ITable testTable = workspace.openTable("parcels");
    IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
    workspaceEdit.startMultiuserEditing(esriMultiuserEditMode.esriMESMNonVersioned); <<-- editing

    ICursor cursor = testTable.ITable_search(filter, false); <<--non-recycling
    IFields fieldSet = cursor.getFields();
    int fieldCount = fieldSet.getFieldCount();
    System.out.println("There are " + fieldCount + " fields.");
    IRow firstRow = cursor.nextRow();

    for (int i = 0; i < fieldCount; i++) {
        Object obj = firstRow.getValue(i);
        if (obj != null) {
            System.out.println("Field value " + i + " has a class type of " + obj.getClass().toString());
        }
    }
    workspace.stopEditing(false);
}
```

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Overriden

- The filter requested 2 fields
- The workspace was editing
- The class was being edited
- All fields were fetched

Output:

There are 7 fields.

Field value 0 has a class type of class java.lang.Integer

Field value 1 has a class type of class java.lang.String

Field value 2 has a class type of class java.lang.Integer

Field value 3 has a class type of class java.util.Date

Field value 4 has a class type of class com.esri.arcgis.interop.NativeObjRef

Field value 5 has a class type of class java.lang.Double

Field value 6 has a class type of class java.lang.Double

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Not Overriden

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

    IQueryFilter filter = new QueryFilter();
    filter.setSubFields("PARCEL_ID, PARCELKEY"); <<-- filter requests 2 fields

    ITable testTable = workspace.openTable("parcels");
    IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
    workspaceEdit.startMultiuserEditing(esriMultiuserEditMode.esriMESMVersioned); <<-- editing

    ICursor cursor = testTable.ITable_search(filter, false); <<--non-recycling
    IFields fieldSet = cursor.getFields();
    int fieldCount = fieldSet.getFieldCount();
    System.out.println("There are " + fieldCount + " fields.");

    IRow firstRow = cursor.nextRow();
    for (int i = 0; i < fieldCount; i++) {
        Object obj = firstRow.getValue(i);

        if (obj != null) {
            System.out.println("Field value " + i + " has a class type of " + obj.getClass().toString());
        }
    }
    workspace.stopEditing(false);
}
```

Non-Recycling Cursors, SubFields, and Editing

Example: SubFields Not Overriden

- The filter requested 2 fields
- The workspace was editing
- The class was not being edited (edit session mode was versioned and the class is not registered)
- 2 fields were fetched

Output:

```
There are 7 fields.
```

```
Field value 1 has a class type of class java.lang.String
```

```
Field value 2 has a class type of class java.lang.Integer
```

Cursor Scope

- Do not hold cursor references if they are not needed.
 - Release ASAP after fetching is completed
- Usage within edit sessions
 - Cursors should be scoped to edit operations – not edit sessions

```
// e.g. 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 Facts

- **Fetching non-recycled rows from an update cursor and calling Store on them will invoke an entirely new update outside the context of the update cursor – should be avoided**
- **A cursor will fetch the Fields needed to satisfy the query, regardless of the SubFields specified in the QueryFilter**
 - Object ID Field
 - Geometry Field if a SpatialFilter is used
- **GetRows should be favored over GetRow in a loop – GetRows can optimize the process – GetRow is always 1 query per row**



Cursor FAQs

Cursor FAQs

When should I release a reference to a cursor?

- Cursors require resources so they should be released as soon as possible – typically this is immediately after the final Row has been fetched or the algorithm determines no more Rows need to be fetched

Cursor FAQs

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

- **Covered earlier. Cursor use should be scoped to edit operations that are within the edit session – inside.**

Cursor FAQs

Should I use a Search Cursor to update rows?

- Yes. In fact, using Search Cursors within an edit session is the recommended way to update rows (see next slide).

Cursor FAQs

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

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

- **Why?**
- **ArcMap – possibility that an active cache can satisfy the query and no DBMS query is required**
- **Engine Simple Data**
 - IES – take advantage of batched updates for edit operations
 - OES – performance, can handle errors on a per row basis
- **Engine Complex Data – the system will emulate Search regardless**



Geodatabase Selection Sets

Geodatabase Selection Sets

What are they?

- A Selection Set is an object that references a set of rows by object id.
- Like a Class Cursor, a Selection Set is bound to the class which created it
- A Selection Set supports the manual adding and removing of object ids
- A Selection Set can be queried – result can be a cursor of rows or a new selection set
- Selection Sets can be combined to produce new a Selection Set
 - Combine Ops - `esriSetUnion`, `esriSetIntersection`, `esriSetDifference`, `esriSetSymDifference`

Geodatabase Selection Sets

Creating Selection Sets

- **Methods that create Selection Sets**
 - **ITable.Select** – creates a new selection based on the class
 - **ISelectionSet.Select** – creates a new selection from an existing selection based on a QueryFilter (possibly a subset)
- **esriSelectionType(s)**
 - **esriSelectionTypeIDSet** – object id references not guaranteed to remain in-memory (e.g. ArcSDE logfiles)
 - **esriSelectionTypeSnapshot** – object id references guaranteed to remain in-memory
 - **esriSelectionTypeHybrid** – object id references transient (initially Snapshot but can become IDSet)

Geodatabase Selection Sets

Creating Selection Sets

- **esriSelectionOption**
 - **esriSelectionOptionNormal** – default, create the selection based on the input QueryFilter
 - **esriSelectionOptionOnlyOne** – use the input QueryFilter but only select the first matching row (single object id)
 - **esriSelectionOptionEmpty** – ignore the input QueryFilter and create a Selection Set which is initially empty
- **Notes about the Selection Container argument**
 - **IWorkspace** argument intended to provide alternative object id data storage
 - Ignored by all 9.2 DataSources – consider it deprecated

Geodatabase Selection Sets

Best Practices

- If the intended use of the Selection Set is for Combine operations, create it as IDSet – avoids the creation of a temporary IDSet (ArcSDE Geodatabases)
- Extracting object ids only – use an ID Enumerator over a Search Cursor (IDs property vs. Search method)
 - Performance
 - ID Enumerators can be re-iterated making them good candidates for reuse, cursors cannot

Geodatabase Selection Sets

Best Practices: ID Enumerators

```
// Only need object ids, use an IEnumIDs

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

    ITable testTable = workspace.openTable("parcels");

    IQueryFilter queryFilter = new QueryFilter();
    queryFilter.setWhereClause("parcel_id like 'P%'");

    ISelectionSet sel = testTable.select(queryFilter, esriSelectionType.esriSelectionTypeSnapshot,
        esriSelectionOption.esriSelectionOptionNormal, null);

    IEnumIDs idEnum = sel.getIDs();

    int oid;

    for (;;) {
        oid = idEnum.next();
        if (oid == -1)
            break;
        System.out.println(oid);
    }
}
```

Geodatabase Selection Sets

Best Practices: ID Enumerators

```
// POOR EXAMPLE of using a cursor just to fetch object ids

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

    ITable testTable = workspace.openTable("parcels");

    IQueryFilter queryFilter = new QueryFilter();
    queryFilter.setWhereClause("parcel_id like 'P%'");

    ISelectionSet sel = testTable.select(queryFilter, esriSelectionType.esriSelectionTypeSnapshot,
        esriSelectionOption.esriSelectionOptionNormal, null);

    ICursor[] cursor = new ICursor[1];
    sel.search(null, true, cursor); <<-- WHY NOT JUST USE AN IEnumIDs?

    int oid;
    IRow row = cursor[0].nextRow();

    while (row != null) {
        System.out.println(row.getOID());
        row = cursor[0].nextRow();
    }
}
```

Geodatabase Selection Sets

Best Practices

- **Avoid the use of `esriSelectionTypeHybrid` if you know the Selection Set size is above or below the selection threshold**
 - Selection threshold is the size at which a hybrid Selection Set transitions from Snapshot to IDSet
 - Registry Key - default selection threshold is 100
 - Using hybrid for Selection Sets that are always above the selection threshold results in 2 queries per Selection Set
 - Always use Snapshot for small Selection Sets (unless the intent is to use them in Combine operations)
 - Hybrid should only be used when the size is an unknown with some expectation that it could grow beyond the selection threshold

Geodatabase Selection Sets

Best Practices

- Favor AddList over Add
 - Adding an array of object ids is always more efficient than single adds in a loop
- If the object ids are known, create an empty Selection Set and use AddList – no need to build a QueryFilter with an “IN” list of object ids
- ISelectionSet2.Update – updates based on a large set of object ids (less complicated than “chunked IN list” searches)
 - Create an empty Selection Set
 - AddList
 - ISelectionSet2.Update
 - Update using the update cursor



Unique Instancing of Objects

Unique Instancing of Objects

What is it?

- **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)
 - Versions
- **Regardless of the API that handed out the reference, it is the same reference**
- **Changes to the object effect (obviously) all holders of the reference**

Unique Instancing of Objects

Examples

```
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 Instanting of Objects

Examples: Special Case

Rows 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(esriMultiuserEditMode.esriMESMNonVersioned);  
  
    IFeature fe1 = fc1.getFeature(1);  
    IFeature fe2 = fc1.getFeature(1);  
    System.out.println(fe2.equals(fe1)); <<- true  
  
    workspace.stopEditing(false);  
}
```



Junk Yard

Junk Yard

Miscellaneous Thoughts

- Prototyping
- Feature classes in a feature dataset
- Performance
- Practical limits

Further Questions

- **TECH-TALK**

- **What:** Opportunity to ask questions and discuss concerns with Erik, Mike, and other GDB team members
- **Where:** One of the five Tech Rooms
- **When:** During the next 30 minutes

- **Meet the Teams**

- **When:** Wednesday, 11:00am, Message Center and Lounge

- **ESRI Developers Network (EDN) website**

- <http://edn.esri.com>

Tech Talk Map

