Effective Geodatabase Programming

Erik Hoel and Brent Pierce

Purpose

- Cover material that is important to master in order for you to be an effective Geodatabase programmer
- Provide additional insight regarding how we (the Geodatabase development team) conceptualize the architecture
- General focus areas:
  - Architecture
  - Performance and scalability

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)
- Lots of technical content, very little time (75 minutes)
  - Please 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
  - Extra bonus material in the downloadable version

Outline

- Implementation Architecture
- Object Class Properties
- Validating Data
- Dataset Extensions
- Data Elements and Geodatabase XML
- Cursors
  - Class Cursors
  - QueryDef Cursor
- Selection Sets
- Unique Instancing of Objects

Implementation Architecture

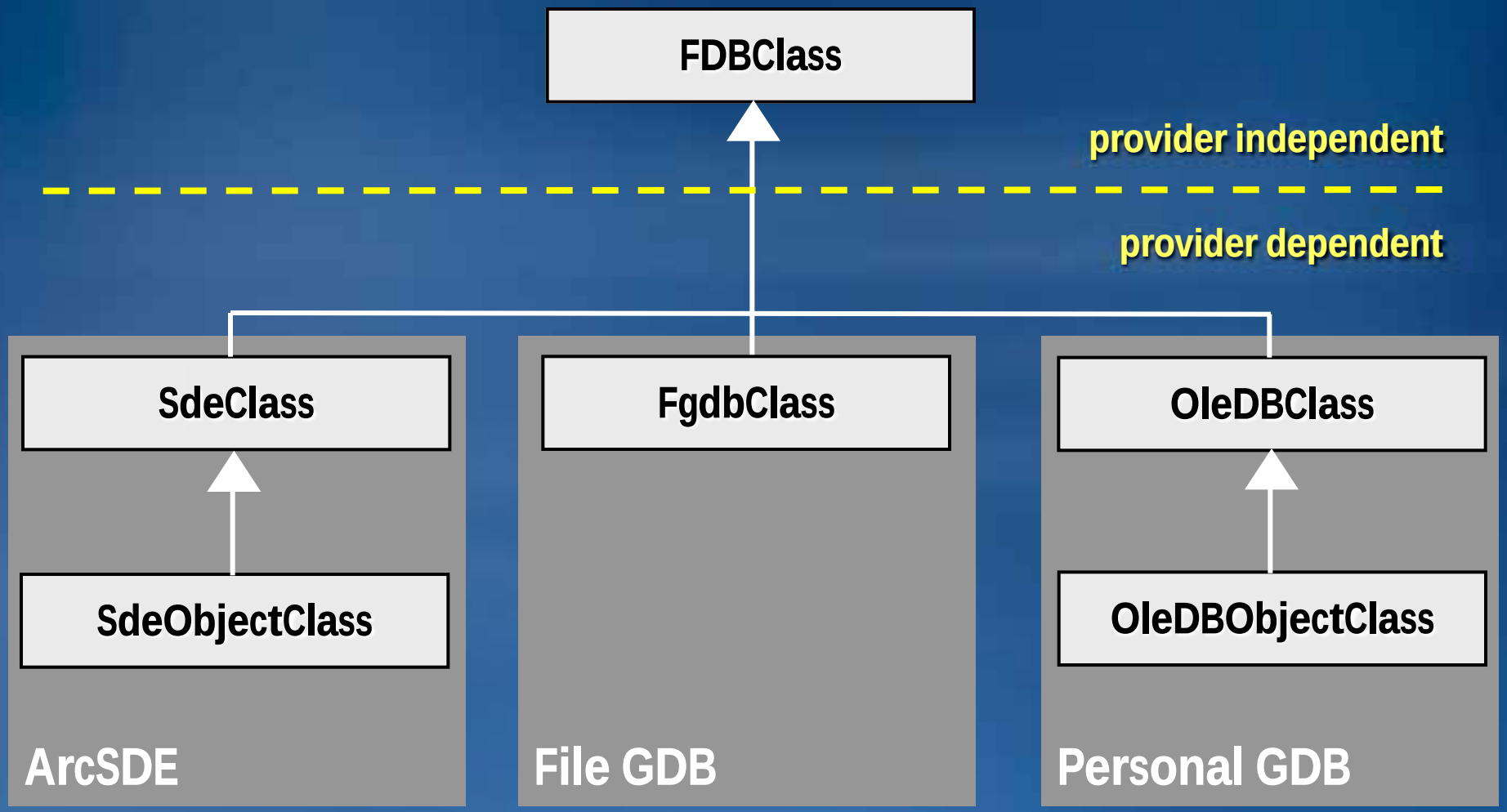
Implementation Architecture

- Geodatabase is composed of many DLLs (~45)
- At the lowest level, it utilizes three providers:
  - ArcSDE Geodatabases - SDE C-API
  - File Geodatabases – implemented internally
  - Personal Geodatabases - Microsoft Jet (via DAO)
- 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 three public properties of an object class that control its behavior
  - requiresStoreMethod
  - requiresEditSession
  - 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 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)

Object Class Properties
Setting

- 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

- 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
  - Can the class 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 (e.g., a 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 (e.g., class extension)
  - Relationship rules

Validating Data

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

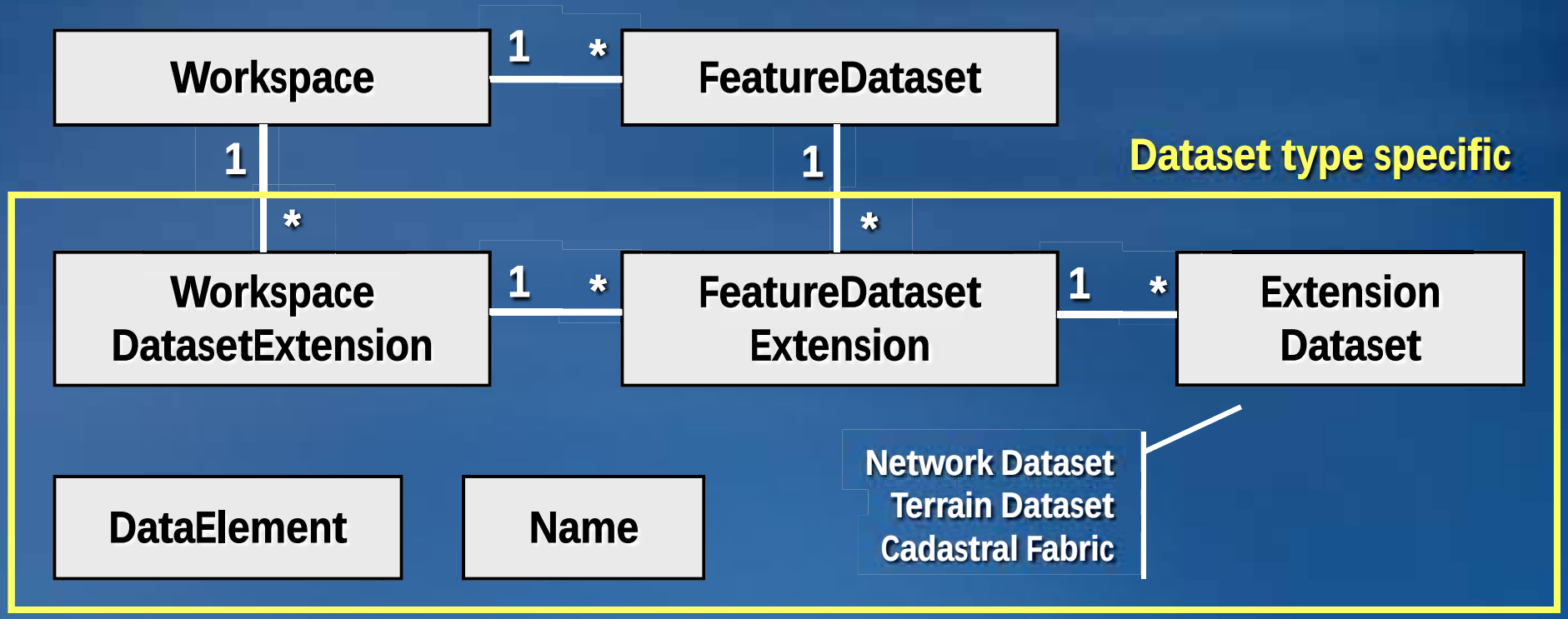
Dataset Extensions

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

Dataset Extensions

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);
}
```

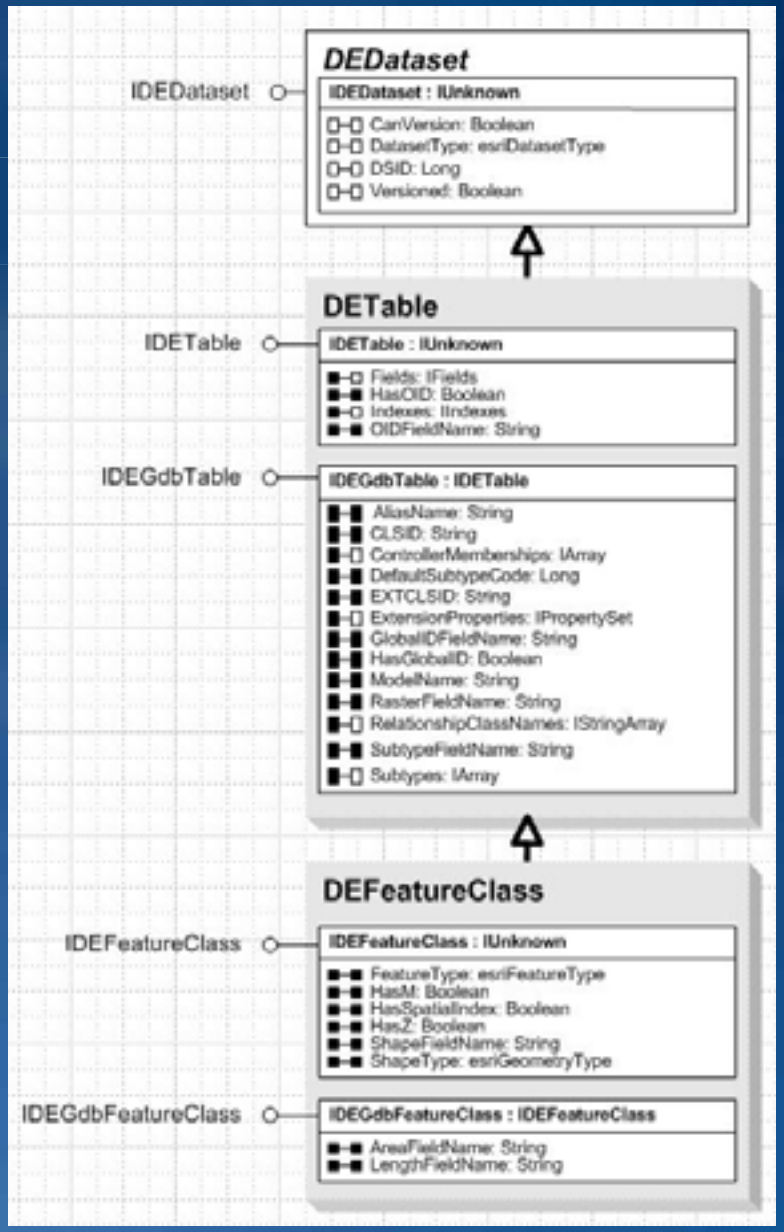
Dataset Extensions
Creation and Schema Updates

- 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

Data Elements and Geodatabase XML

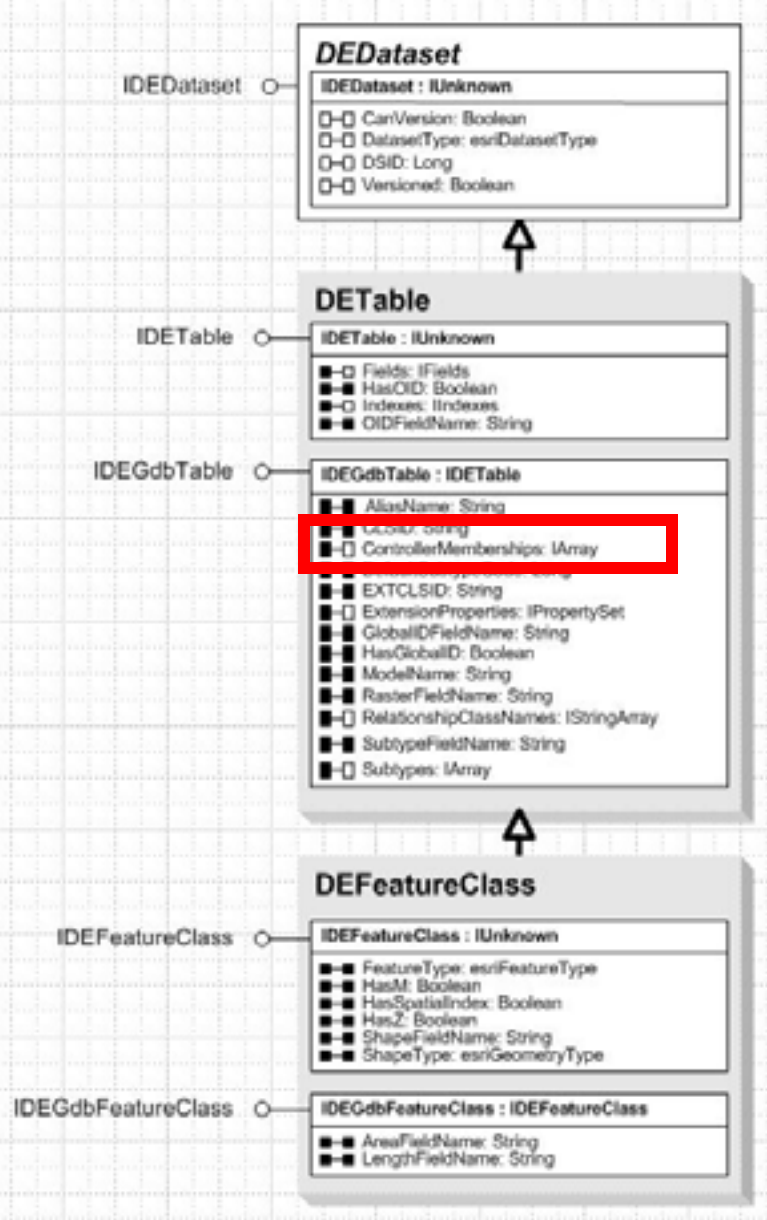
Data Elements
Overview

- Introduced at ArcGIS 9.0
- Purpose is to provide a value-based object that contains all metadata related to a dataset instance
  - No active behavior
  - Different focus from Name objects
- Heavily utilized by Geoprocessing
- Used for new controller datasets beginning at ArcGIS 9.1
  - Network datasets
  - Terrain datasets
  - Cadastral fabrics
  - Representation classes



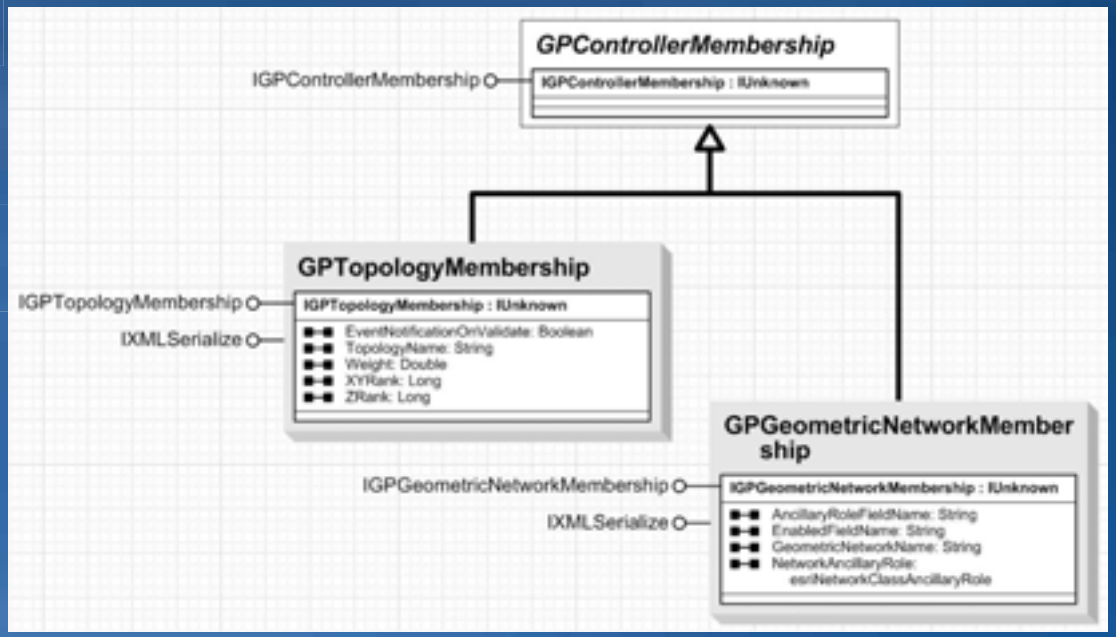
Data Elements
Overview

- Introduced at ArcGIS 9.0
- Purpose is to provide a value-based object that contains all metadata related to a dataset instance
  - No active behavior
  - Different focus from Name objects
- Heavily utilized by Geoprocessing
- Used for new controller datasets beginning at ArcGIS 9.1
  - Network datasets
  - Terrain datasets
  - Cadastral fabrics
  - Representation classes



Data Elements
Controller Memberships

- The association between a feature class and controller dataset have extra attributes, e.g.,
  - the rank of the feature class in a topology, or
  - the enabled field of a feature class in a geometric network
- These are attributes of the feature class' membership in a controlling entity, they wouldn't exist if the feature class were not a member of the controller
- The table data element has an array of controller memberships



Data Elements
Creation

- Data elements support IXMLSerialize and IPersistStream
  - Can be serialized in XML or binary form
- Data elements can be created in several ways
  - They are cocreatable, and application developers can create new instances of a data element then fill its properties with appropriate values
  - In ArcCatalog, they can be obtained from a GxObject using the GetDataElement() method in the IGxDataElement interface
  - Workspace can implement the IWorkspaceDataElements interface, which can be used to request a data element for the whole workspace, or a data element for a particular dataset.

Data Elements

Getting a Data Element using a Name object

```
// Assume workspace has a connection to the Geodatabase.

IEnumDatasetName datasetNames =
           workspace.get_DatasetNames(esriDatasetType.esriDTFeatureClass);

datasetNames.Reset();

IDatasetName datasetName = datasetNames.Next();

// When using a name object, you must request the base properties.

IDEBrowseOptions options = new DEBrowseOptionsClass();

options.RetrieveFullProperties = false;  // When using name objects
options.ExpandType = esriDEExpandType.esriDEExpandNone;

IWorkspaceDataElements workspaceDE = (IWorkspaceDataElements)workspace;

IDEFeatureClass deFeatureClass =
    (IDEFeatureClass)workspaceDE.GetDatasetDataElement(datasetName, options);
```

1

2

3

Data Elements

Getting a Data Element using a Dataset

```
// Assume workspace has a connection to the Geodatabase.

IEnumDatasetName datasetNames =
           workspace.get_DatasetNames(esriDatasetType.esriDTFeatureClass);

datasetNames.Reset();

IDatasetName datasetName = datasetNames.Next();
IName name = (IName)datasetName;
IDataset dataset = (IDataset)name.Open();

// When using a dataset object, you must request the full properties.

IDEBrowseOptions options = new DEBrowseOptionsClass();

options.RetrieveFullProperties = true;

IWorkspaceDataElements workspaceDE = (IWorkspaceDataElements)workspace;

IDEFeatureClass deFeatureClass =
    (IDEFeatureClass)workspaceDE.GetDatasetDataElement(datasetName, options);
```

1

2

3

4

Geodatabase XML
Overview

- Geodatabase XML is ESRI's open mechanism for information interchange between geodatabases and other external systems
- ESRI openly publishes and maintains the complete geodatabase schema and content as an XML specification
- The specification encompasses all geodatabase data types


```
- <esri:Workspace xmlns:esri="http://www.esri.com/schemas/ArcGIS/9.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://
  - <WorkspaceDefinition xsi:type="esri:WorkspaceDefinition">
      <WorkspaceType>esriLocalDatabaseWorkspace</WorkspaceType>
      <Version />
    + <Domains xsi:type="esri:ArrayOfDomain">
    - <DatasetDefinitions xsi:type="esri:ArrayOfDataElement">
      - <DataElement xsi:type="esri:DEFeatureClass">
          <CatalogPath>/FC=ARIRWParcels</CatalogPath>
          <Name>ARIRWParcels</Name>
          <DatasetType>esriDTFeatureClass</DatasetType>
          <DSID>31</DSID>
          <Versioned>false</Versioned>
          <CanVersion>false</CanVersion>
          <HasOID>true</HasOID>
          <OIDFieldName>OBJECTID</OIDFieldName>
        + <Fields xsi:type="esri:Fields">
        + <Indexes xsi:type="esri:Indexes">
          <CLSID>{52353152-891A-11D0-BEC6-00805F7C4268}</CLSID>
          <EXTCLSID />
        + <RelationshipClassNames xsi:type="esri:Names">
          <AliasName>ARIRWParcels</AliasName>
          <ModelName />
          <HasGlobalID>false</HasGlobalID>
          <GlobalIDFieldName />
          <RasterFieldName />
        + <ExtensionProperties xsi:type="esri:PropertySet">
          <ControllerMemberships xsi:type="esri:ArrayOfControllerMembership" />
          <FeatureType>esriFTSimple</FeatureType>
          <ShapeType>esriGeometryPolygon</ShapeType>
          <ShapeFieldName>SHAPE</ShapeFieldName>
          <HasM>false</HasM>
          <HasZ>false</HasZ>
          <HasSpatialIndex>true</HasSpatialIndex>
          <AreaFieldName>SHAPE_Area</AreaFieldName>
          <LengthFieldName>SHAPE_Length</LengthFieldName>
        + <Extent xsi:type="esri:EnvelopeN">
        + <SpatialReference xsi:type="esri:ProjectedCoordinateSystem">
        </DataElement>
      + <DataElement xsi:type="esri:DETable">
      + <DataElement xsi:type="esri:DERelationshipClass">
      </DatasetDefinitions>
    </WorkspaceDefinition>
    <WorkspaceData xsi:type="esri:WorkspaceData" />
  </esri:Workspace>
```

Geodatabase XML
Overview

- Allows applications to send and receive XML data streams:
  - Exchange of complete lossless data sets
  - Exchange of change-only (delta) record sets to pass updates and changes
  - Exchange and sharing of full or partial geodatabase schemas
  - Interchange of simple feature sets (much like shapefile interchange)

- Users can export all or any part of a geodatabase, such as individual feature data sets, feature classes, and tables, to an export file in XML format

Data Elements and XML
The Future

- With ArcGIS 9.4, the Geodatabase schema is being revamped
  - Simpler structure (four GDB_* tables rather than 35+)
  - Faster search capabilities
  - Open schema (no binary serialization)
  - Well documented
    - *XML Schema of the Geodatabase* white paper
  - Heavy reliance upon data elements and XML
    - Somewhat similar to the newer controller datasets (e.g., GDB_ExtensionDatasets table)

- Critical that your applications do not directly query the GDB system tables

- You will not have to change your AO code!
  - All geodatabase APIs will continue to work seamlessly with new schema

Data Elements and XML
The Future

- We are working on a low-level (non ArcObjects-based) API for the file geodatabase
- File geodatabase can model the full ArcGIS information model: entity relationships, 3D objects, annotation, dimensions, topology, networks, raster data, point clouds, etc.
- The geodatabase schema will be exposed using Geodatabase XML to represent the schema
- There will be a light-weight API for interacting with the rows of the individual datasets and the schema of the Geodatabase

- We will be publishing more information on what we are planning with this API later this year and we welcome your input on this design as it becomes available

Cursors

Cursors

Cursor Types

- Class Cursors (three)
  - Search (general query cursor)
  - Update (positioned update cursor)
  - Insert (bulk inserts)
- QueryDef Cursor (one)
  - 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

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

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
  - Could result in a database write
- 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
- As a developer how do you know?
  - Non-simple feature types (!esriFTSimple)
  - Class participates in Geometric Network, Relationships that require messaging
  - Custom Features (ObjectClassExtension overrides)
- UpdateRow Method Behaviors
  - Error if called with a row that was not retrieved from the update cursor
  - Error if called with a row not at the current position

Cursors
Class Cursors: Update Best Practices

- Do not use an update cursor across edit operations when editing
- Very important when editing versioned data
- Update cursors will point to a specific state in the database
  - This state could get trimmed during an edit sessions
  - 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

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)

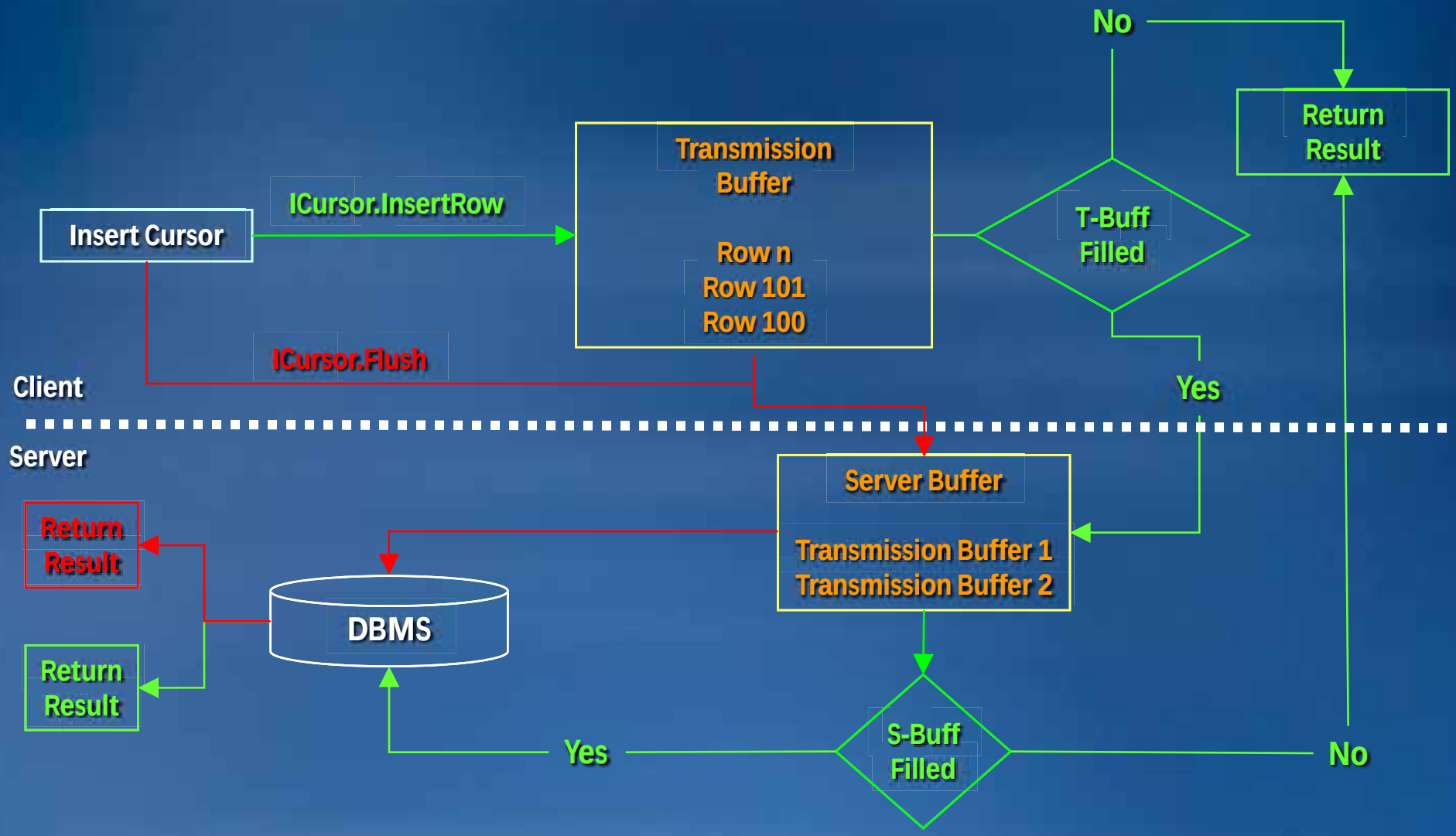
Cursors

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
  - The higher the flush interval the less network traffic (but more re-insertion)
- Try to ensure that the class has no spatial cache – extra processing is required to keep the cache in synch
- Crucial that calls to InsertRow and Flush have the proper error handling since both can result in rows written to the database

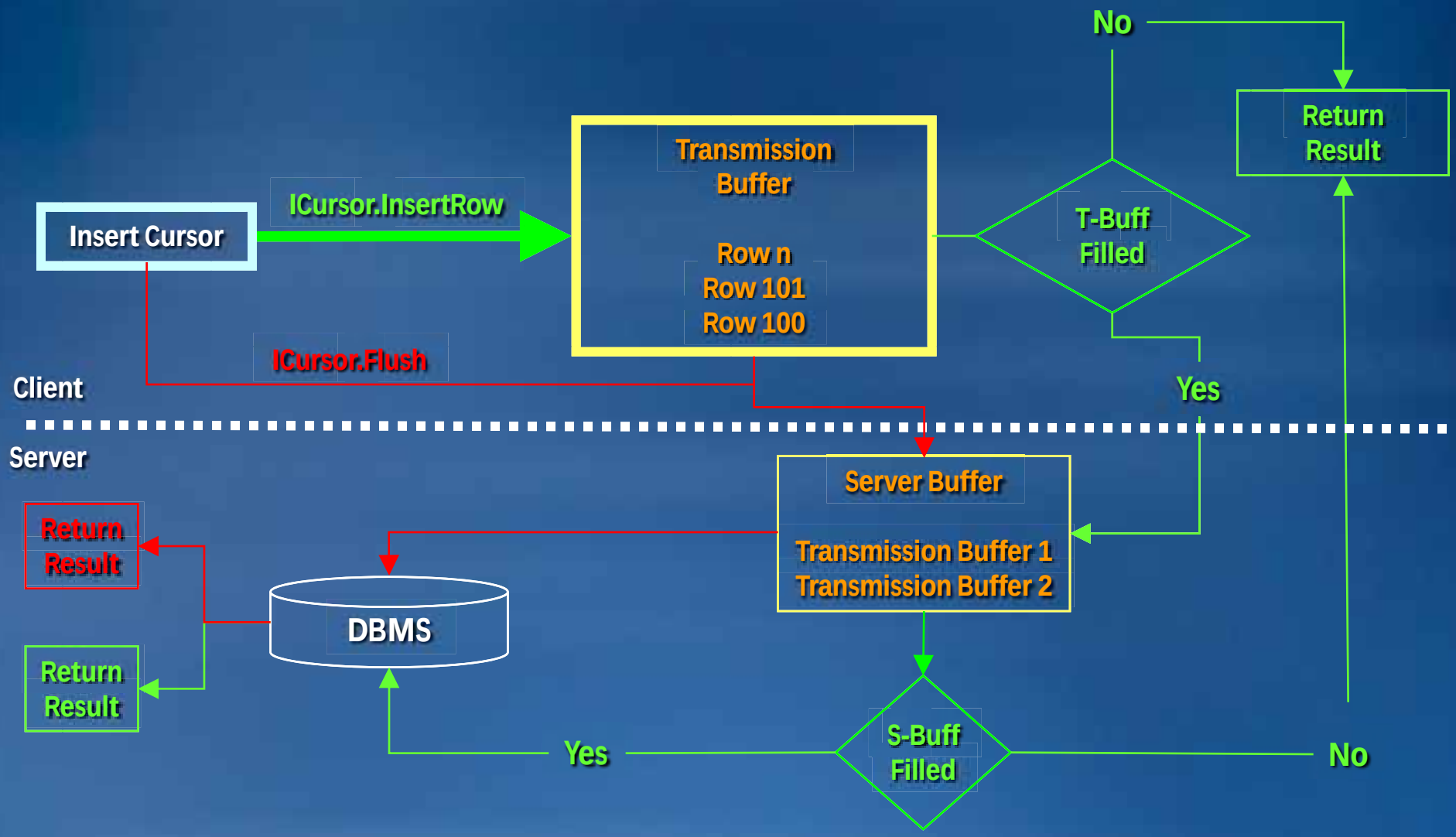
Cursors

Class Cursors: Insert – Buffering (Enterprise)



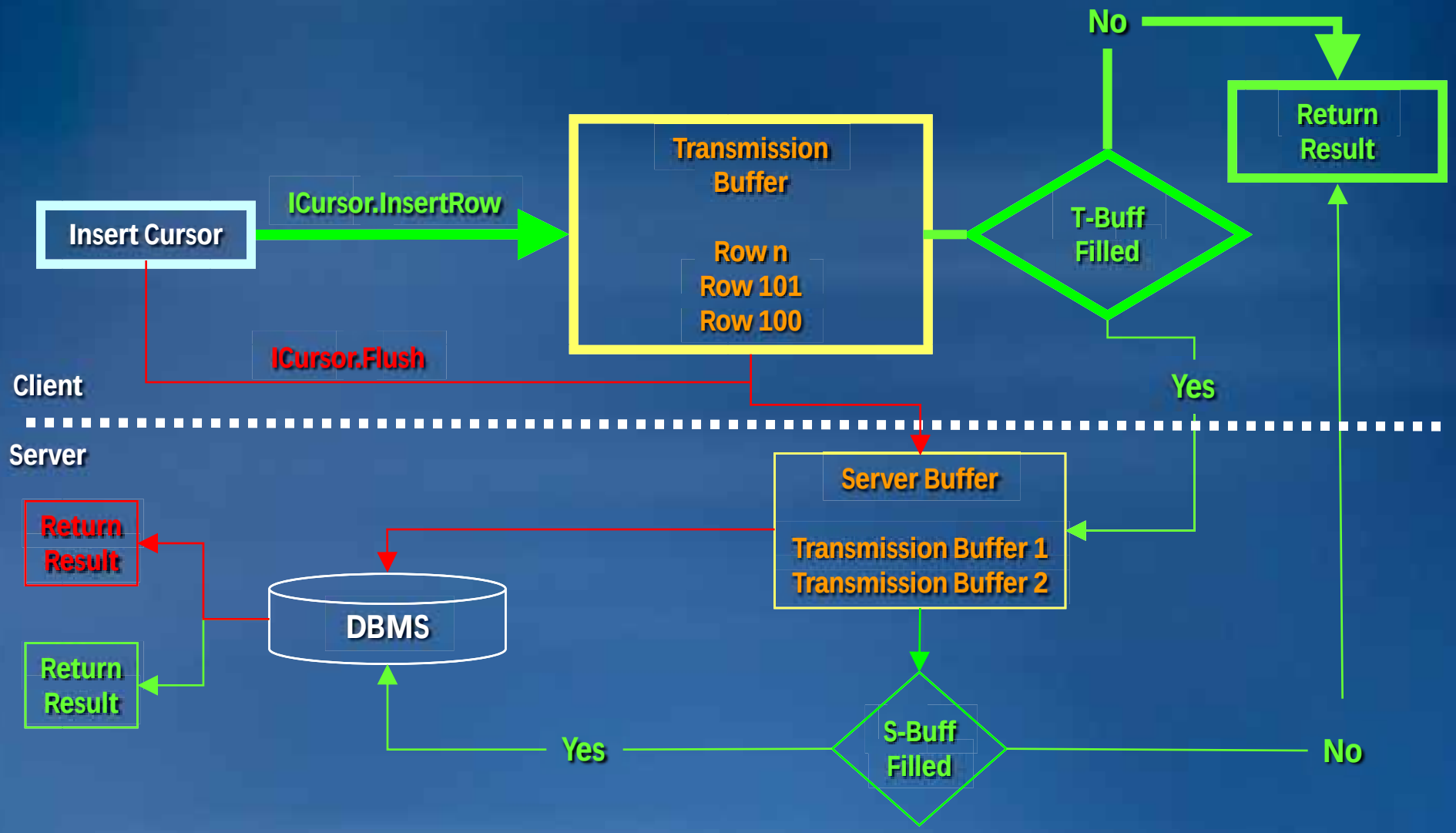
Cursors

Class Cursors: Insert – Buffering (Enterprise)



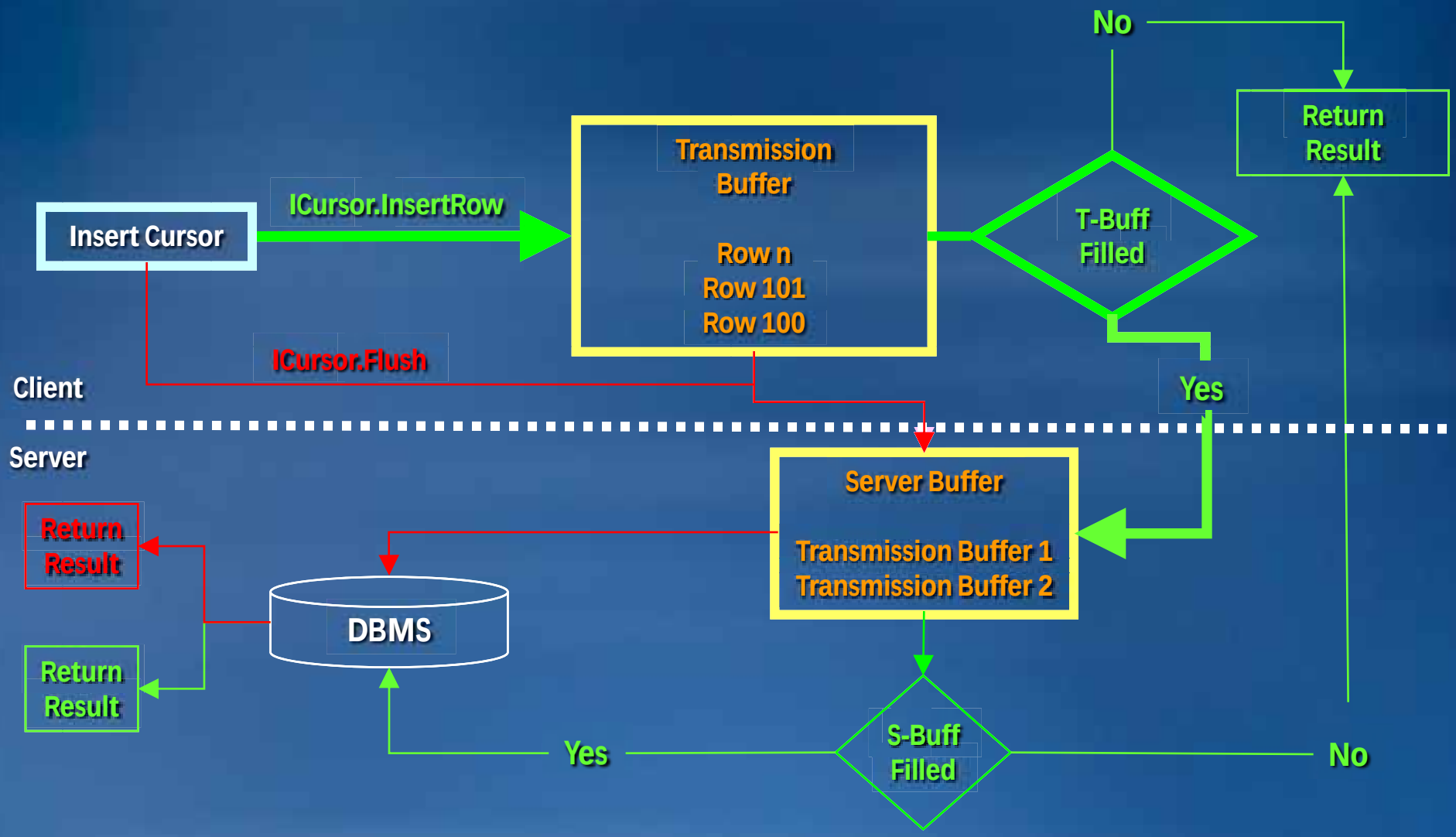
Cursors

Class Cursors: Insert – Buffering (Enterprise)



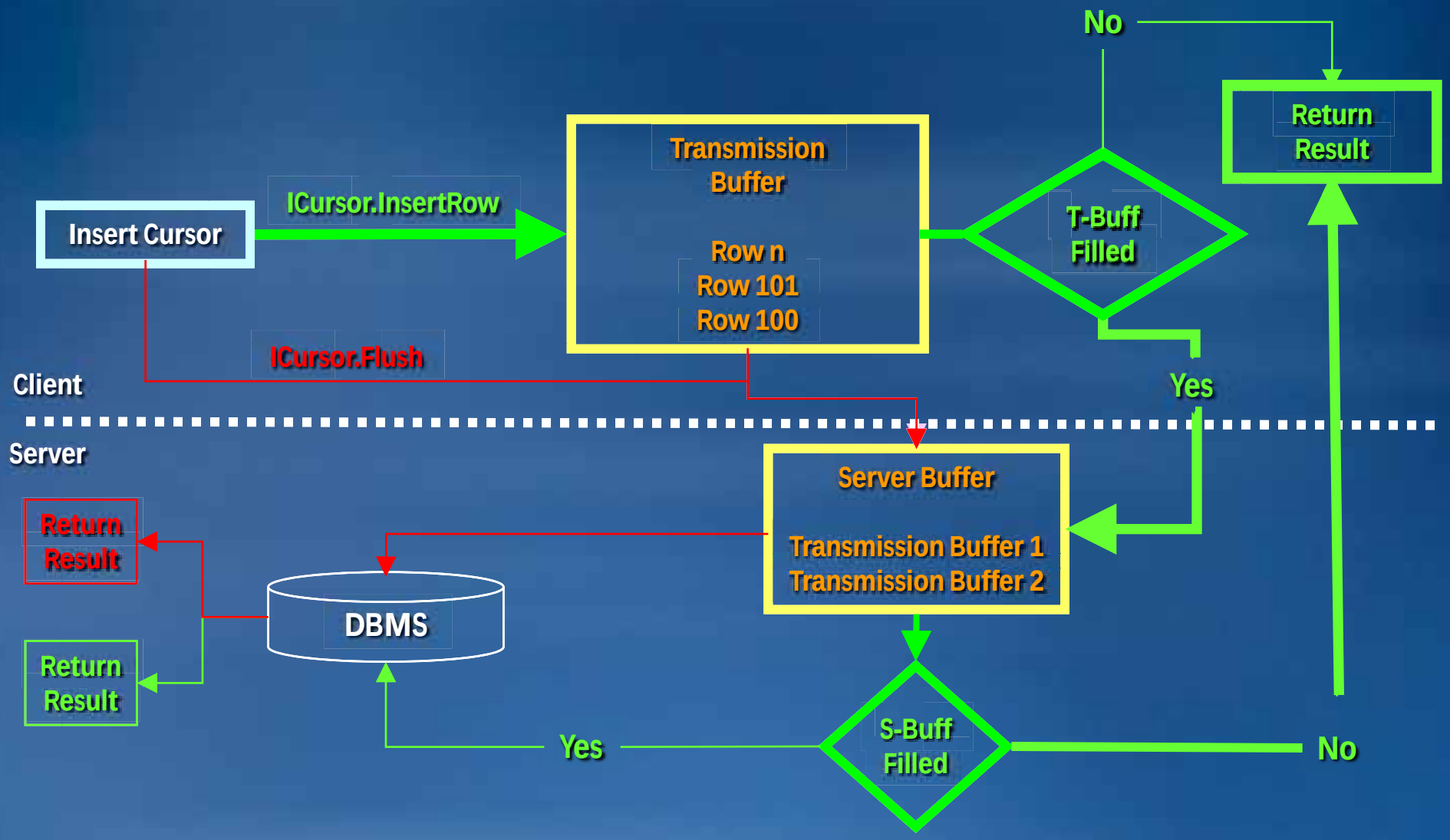
Cursors

Class Cursors: Insert – Buffering (Enterprise)



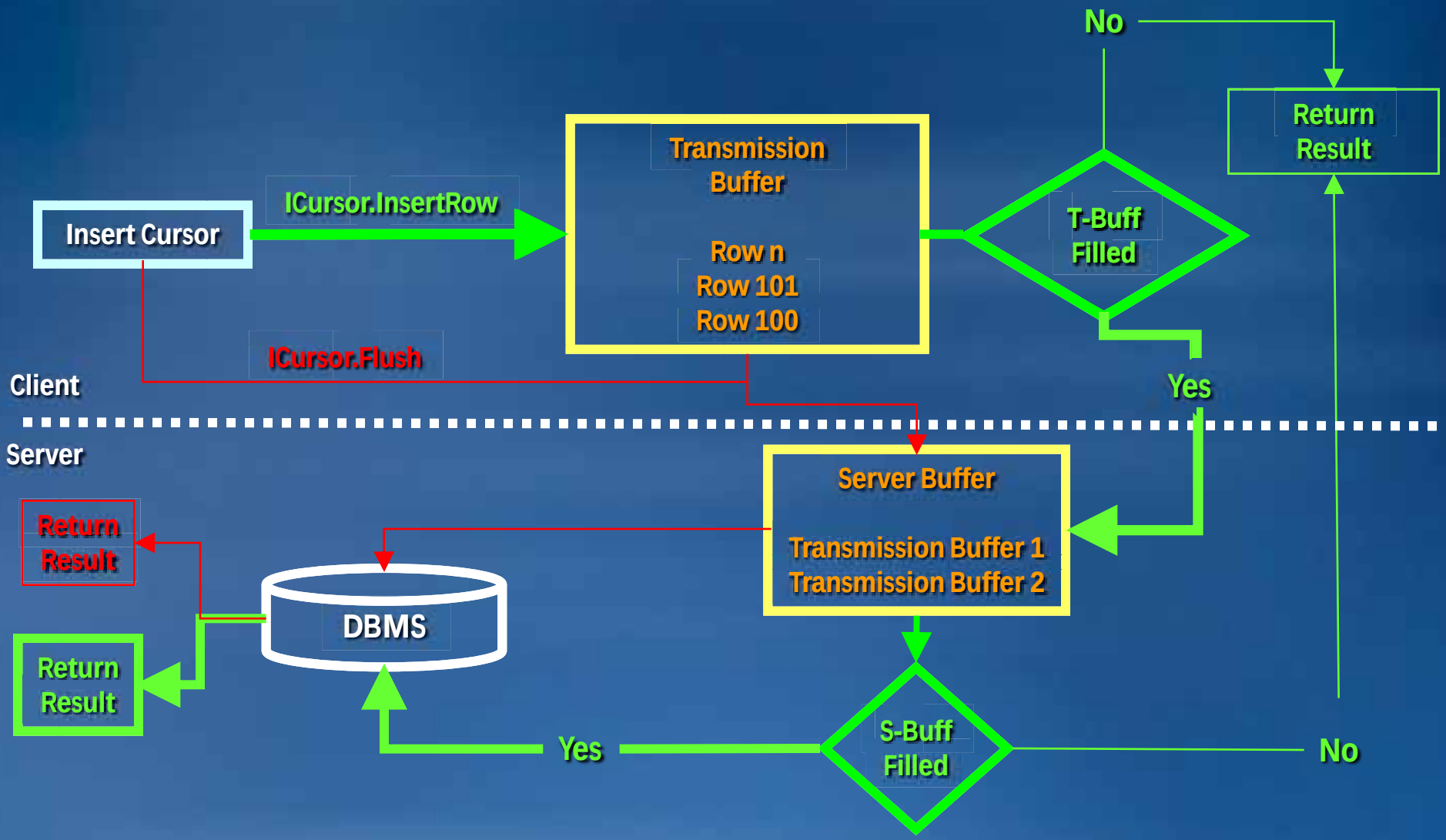
Cursors

Class Cursors: Insert – Buffering (Enterprise)



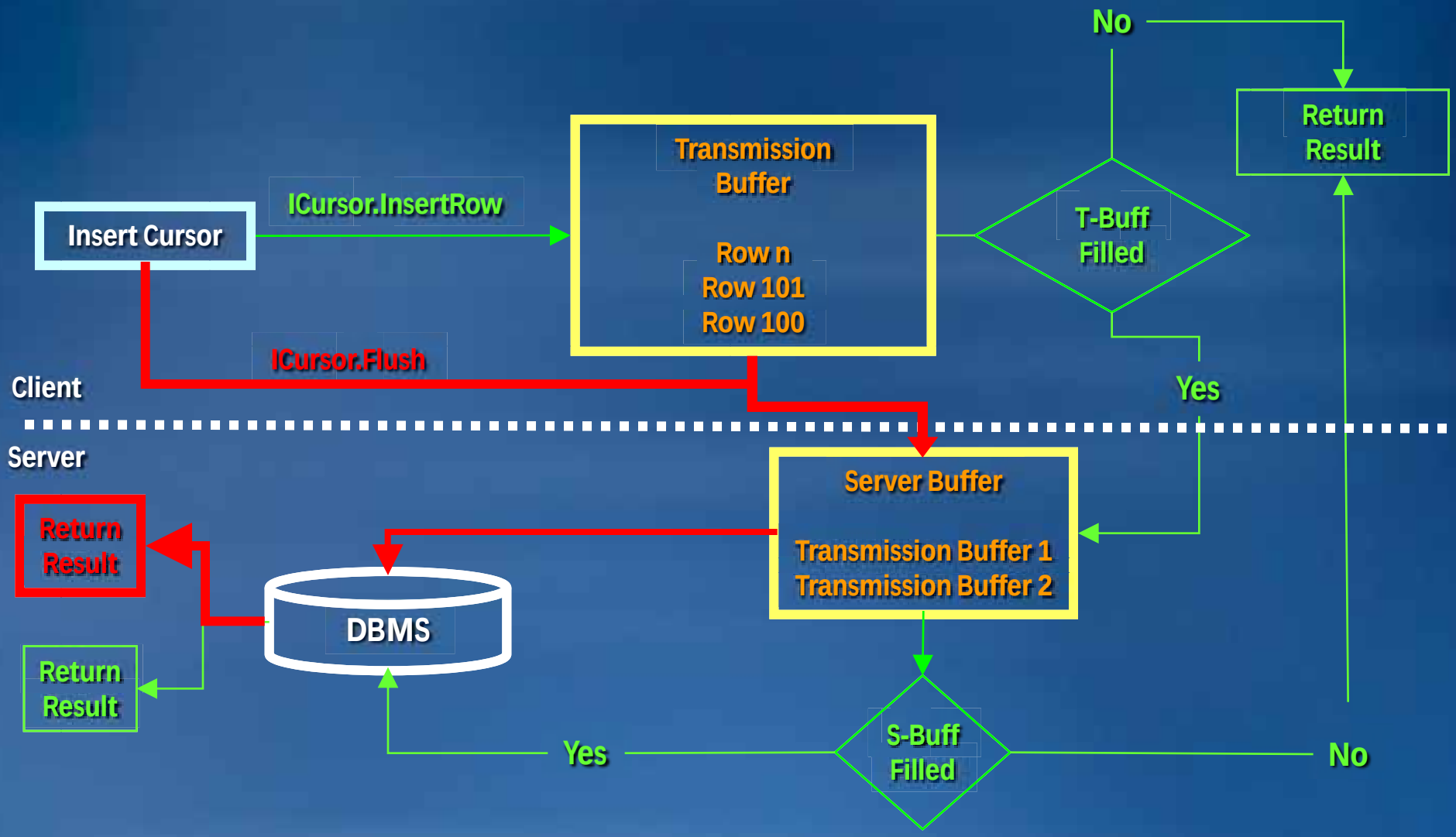
Cursors

Class Cursors: Insert – Buffering (Enterprise)



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

Cursors

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

- Examples of when the recycling property is 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

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)); <<-- 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)); <<-- 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 a call to NextRow / NextFeature
- Don't Pass it around
  - 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
- Never directly edit a recycled row

Recycling Cursors

Use Cases: 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(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)

Cursors

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
- Always edit non-recycled rows

Cursors

Non-Recycling, Sub Fields 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 two fields
- The workspace was not editing
- The class was not being edited
- Two 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

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

IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
workspaceEdit.startMultiuserEditing(esriMESMNonVersioned); <<-- Non-Versioned editing

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 Overriden

- The filter requested two 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

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

IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
workspaceEdit.startMultiuserEditing(esriMESMVersioned); <<-- Versioned Editing

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 two fields
- The workspace was editing
- The class was **not being edited**
  - edit session mode was versioned
  - the class is not registered as versioned
- Two 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
```

Cursors

Frequently Asked Questions

Cursor FAQs

When should I release a reference to a cursor?

Answer:

Do not hold cursor references if they are not needed.

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

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?

Answer:

Create cursors inside the edit session

- Scope to edit operations

```
// 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 FAQs

Should I use a Search Cursor to update rows?

Answer:

Yes. In fact, using Search Cursors within an edit session is the recommended way to update rows (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
  - Inside ES – take advantage of batched updates for edit operations
  - Outside ES – performance, can handle errors on a per row basis
- Engine Complex Data – the system will emulate Search regardless

Cursors
Best Practices

- Don't call Store on rows returned from a Update cursor
  - will invoke an entirely new update outside the context of the update cursor
- 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 one query per row

Geodatabase Selection Sets

Geodatabase Selection Sets
What are they?

- An object that references a set of rows by object id

- Like a class cursor, they are bound to the class which created them

- Capabilities:
  - Support the manual adding and removing of object ids
  - Can be queried – result can be a Cursor of Rows or a new SelectionSet
  - Two can be combined to produce new a SelectionSet
    - esriSetUnion – boolean OR
    - esriSetIntersection – boolean AND
    - esriSetDifference – boolean INHIBITION
    - esriSetSymDifference – boolean XOR

Geodatabase Selection Sets
Creating Selection Sets

- Methods that create SelectionSets
  - 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)

- esriSelectionTypes
  - 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)

  - Rule of Thumb - If you know the selection set will be big use an 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 SelectionContainer argument
  - IWorkspace argument intended to provide alternative object id data storage
  - Ignored by all post-9.1 data sources – consider it deprecated

Geodatabase Selection Sets
Best Practices

- If the intended use of the SelectionSet 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 (ISelectionSet.IDs vs. IFeatureClass.Search)

- Why?
  - Performance
  - ID Enumerators can be reiterated making them good candidates for reuse, while Cursors cannot

Geodatabase Selection Sets

Worst Practices: Cursor to fetch object ids

```
// 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: ID Enumerators to fetch object ids

```
// 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

- Avoid the use of esriSelectionTypeHybrid if you know the SelectionSet size is above or below the selection threshold
  - Selection threshold is the size at which a Hybrid SelectionSet transitions from Snapshot to IDSet
  - Registry Key - default selection threshold is 100
  - Using Hybrid for SelectionSets that are always above the selection threshold results in two queries per SelectionSet
  - Hybrid should only be used when the size is an unknown with some expectation that it could grow beyond the selection threshold

- Always use Snapshot for small SelectionSets (unless the intent is to use them in Combine operations)

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 SelectionSet 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 SelectionSet
  - AddList
  - ISelectionSet2.Update
  - Update using the update cursor

Geodatabase Selection Sets

Head's Up

- Using AddList for Java or .NET
  - Add list makes use of a C-Style conformant array
  - Because of this this can have strange side effects in managed languages like Java or .NET
- Use IGeodatabaseBridge2.AddList instead of ISelectionSet.AddList
- Blog Post
  - http://blogs.esri.com/Dev/blogs/geodatabase/archive/2008/06/06/Developer-Tips-_2D00_-Using-geodatabase-methods-with-conformant-array-parameters-in-.NET-and-Java.aspx

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)
    - Workspaces and Versions
- Regardless of the API that handed out the reference, it is the same reference
- Changes to the object affect all holders of the reference

Unique Instancing of Objects
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 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 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 Instancing of Objects
Examples: Special Case

Rows\Features uniquely instanced only within an edit session

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

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

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

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

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

  workspace.stopEditing(false);
}
```

Unique Instancing of Objects
Examples: Special Case

Rows\Features uniquely instanced only within an edit session

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

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

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

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

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

  workspace.stopEditing(false);
}
```

Unique Instancing of Objects
Examples: Special Case

Rows\Features uniquely instanced only within an edit session

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

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

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

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

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

  workspace.stopEditing(false);
}
```

Conclusion

- Implementation Architecture
- Object Class Properties
- Validating Data
- Dataset Extensions
- Data Elements and Geodatabase XML
- Cursors
  - Class Cursors
  - QueryDef Cursor
- Selection Sets
- Unique Instancing of Objects

Developer Resources

- ESRI Blogs (http://blogs.esri.com)
  - Inside the Geodatabase:
    - http://blogs.esri.com/Dev/blogs/geodatabase/default.aspx
- Geodatabase "How To" Articles
  - Developers Guide to the Geodatabase
    - http://resources.esri.com/help/9.3/ArcGISDesktop/dotnet/concepts_start.htm#1f0e8ebb-c59a-49be-998e-aa0b840845e6.htm
  - Many more articles in Geodatabase SDK
    - Working with ArcGIS Components > Geodatabase Management
- Instructor lead training (http://www.esri.com/training)

Other Recommended Sessions

- Developing with Rasters in ArcGIS
  - Wednesday 1:00 – 2:15
- Distributed Geodatabase Development
  - Wednesday 2:45 – 4:00
- Implementing Enterprise Applications with the Geodatabase
  - Wednesday 4:30 – 5:45
- Working with the Geodatabase Effectively using SQL
  - Thursday 8:30 – 9:45

Final Notes

- Thanks for attending!
- Please fill out session surveys
  - Helps us improve the session
- All sessions are recorded and will be available on Resource Centers
  - Slides and code will also be available
- Still have questions?
  - Server Island in the Showcase
  - Tech talk (directly after session)
  - Meet the Team