Developer's Guide to the Geodatabase (Best Practices)

Craig Gillgrass

Colin Zwicker

James MacKay

Schedule

- 4 hours with break 2:15 – 2:45 pm
- Cell phones and pagers

- Please complete the session survey – we take your feedback very seriously!

Seminar Roadmap

- We'll go over basic Geodatabase concepts
  - Might be a review for some of you
- Basic programming skills
  - C# demos
  - Code will be available with slides
  - Ability to read OMDs
  - ArcObjects supports: .Net, Java, C++, etc…

Seminar Roadmap …

- Seminar focus on:
  - Overview of the Geodatabase
  - How to work with the Geodatabase API
  - Tip\Tricks for using the API
- We'll take questions throughout
  - May want to hold off until we get to the end of each topic
  - Hold questions to the end of demos
- Available at the break
- Island area over the next few days

Presentation Outline

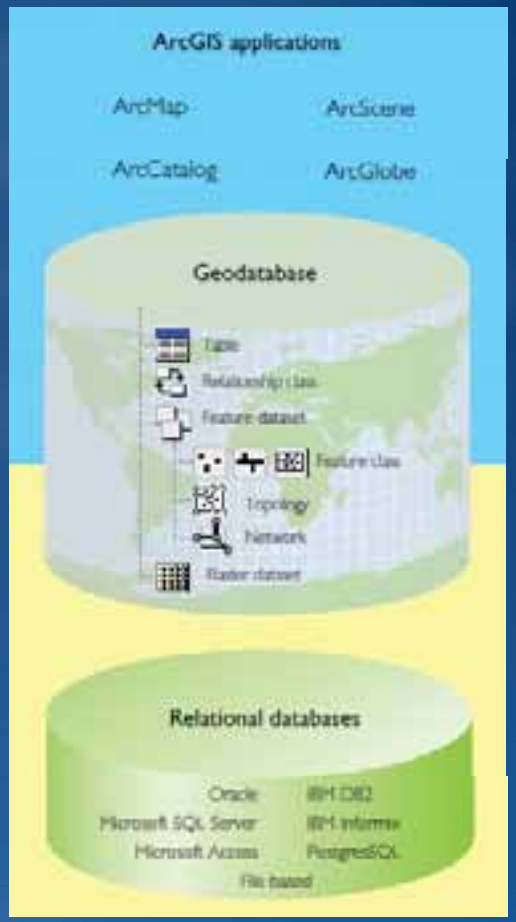
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

What is the geodatabase?

- Core ArcGIS data model
  - A comprehensive model for representing and managing GIS data
- A physical store of geographic data
  - Scalable storage model supported on different platforms
- A transactional model for managing GIS workflows
- Set of COM components for accessing data

Geodatabase Data Management Approach

- The geodatabase is built on an extended relational database.
  - Relational integrity
  - Base short transaction model
  - Reliability, Flexibility, Scalability
  - Supports continuous, large datasets
- Simple features + logic
  - All geographic data stored as tables in a DBMS
  - Extend functionality and data integrity
  - Functionality is consistent across DBMS'
- Application logic (software)
  - Works on standard DBMS tables
  - Implements GIS integrity and behavior
  - Business rules, topology, networks



Geodatabase Data Management Approach …

- Editing and data compilation
  - Rich set of editing tools
  - Maintain spatial and attribute integrity
  - Undo and redo edits
  - Multiple users editing the same data
- Versioning work flows
  - Multiple users editing over long periods of time
  - Archiving
  - Distributed data management
- Components for accessing data
  - Robust, customizable framework
  - Build and manage your own specific GIS solution

Inside the Geodatabase

- A geodatabase contains datasets.
  - Datasets represent collections of information with a real-world interpretation.
- Types of geographic datasets:
  - Tables
  - Object classes, feature classes, relationship classes
  - Feature datasets
  - Networks, Topologies, Raster and cadastral datasets
- Datasets have associated information to help manage integrity, behavior, and interpretation
  - Domains, Relational integrity, Topology, Metadata

Inside the Geodatabase…

Table

Feature dataset

Feature class

- Polygon
- Line
- Point
- Annotation
- Dimension
- Route

Relationship class

Topology

Geometric network

Network dataset

Terrain

Raster dataset

Raster catalog

Schematic dataset

Survey dataset

- Project folder
- Project

Toolbox

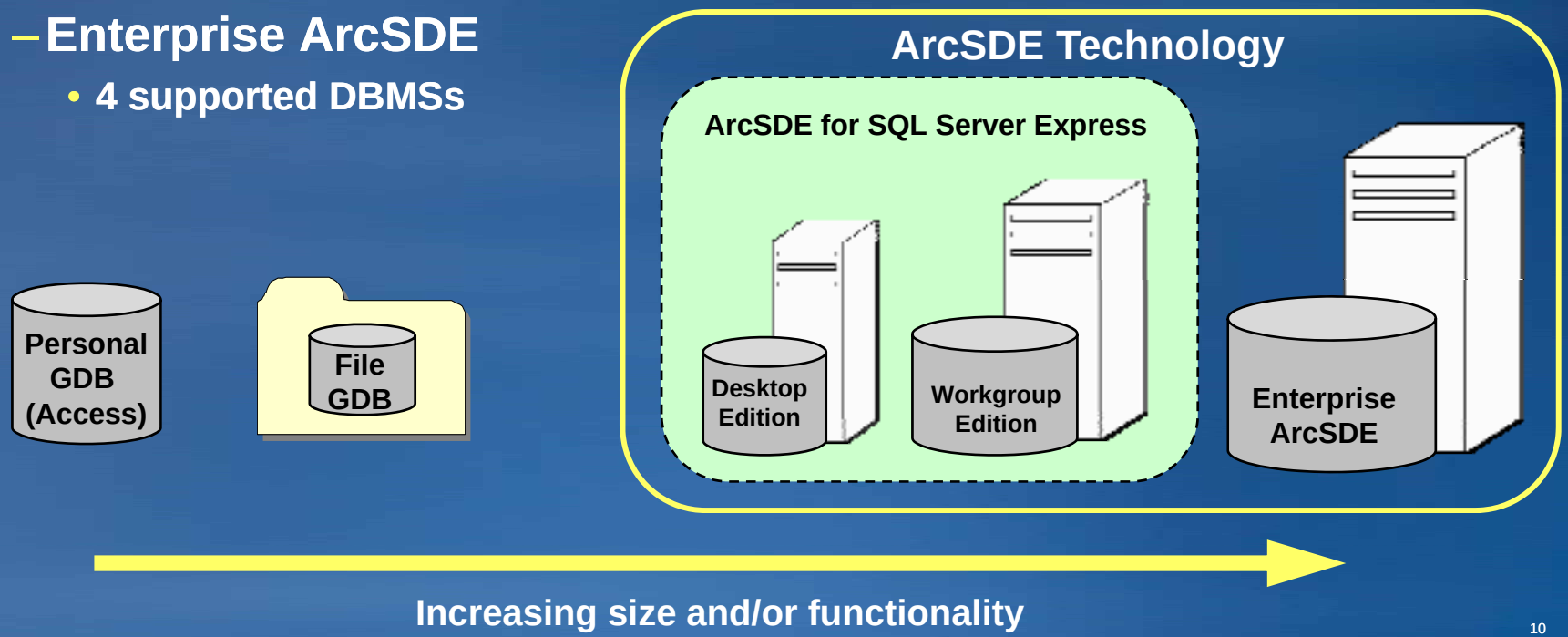
- Tool
- Model
- Script

Behavior

- Attribute domains
- Attribute defaults
- Split/merge policy
- Relationship rules
- Connectivity rules
- Topology rules

Geodatabase …

- Types of Geodatabases
- Geodatabases can be stored different ways
  - Personal geodatabase (Access mdb file)
  - File geodatabase
    - Directory of binary files
  - ArcSDE for SQL Server Express
  - Enterprise ArcSDE
    - 4 supported DBMSs



3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

* By default; option to have 256 TB per table

3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

* By default; option to have 256 TB per table

3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

* By default; option to have 256 TB per table

Demo

- Tour around a Geodatabase – Part 1
- Tour around a geodatabase – Part 2
- Create a file geodatabase (.gdb)
- Create a feature dataset and feature class
- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

Presentation Outline

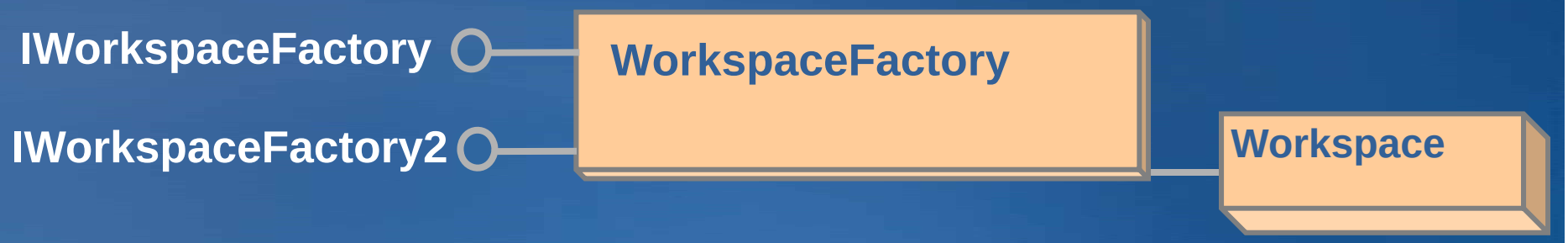
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Licensing

- Required to initialize licensing to correct level when working with geodatabase in Engine environment
  - If not, get an error on call to open the workspace
- Configure license at application start time
  - esriLicenseProductCodeEngineGeoDB
- Enterprise geodatabase editing application:
  - ArcGIS Engine Geodatabase Editing license
  - ArcEditor license
  - ArcInfo license

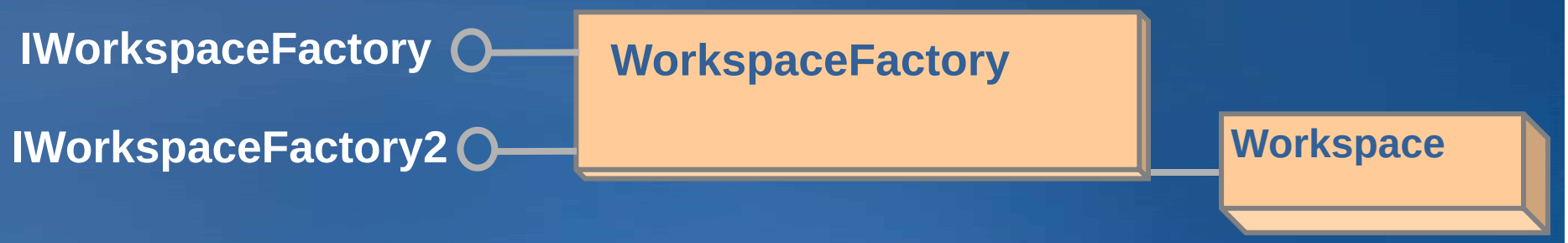
Workspace Factory and Workspaces

- Workspace Factory is a dispenser of workspaces
  - Personal, File, ArcSDE workspaces
- Create a factory from a workspace factory coclass
- Workspace factories are singleton objects
  - One instance created per Component Object Model (COM) apartment
  - Further calls return a reference to the existing object



Workspace Factory and Workspaces

- *A workspace is a container of datasets*.
  - Geodatabase, coverage workspace, folder of shapefiles
- Workspaces are not singleton objects
  - But, requesting a workspace with the same properties as an existing instance returns a reference to it
  - The Geodatabase guarantees unique instancing



Making a connection

- Create a workspace from a factory
  - Path to data and window handle (app ID)
  - Propertyset or string containing connection properties
- Connecting using a property set

```
IWorkspaceFactory sdeWkspFact = new SdeWorkspaceFactoryClass();
IPropertySet propset = new PropertySetClass();

propset.SetProperty("SERVER",“crimsontide");
propset.SetProperty("INSTANCE","5151");
propset.SetProperty("USER","brent");
propset.SetProperty("PASSWORD","brent");
propset.SetProperty("DATABASE","null");
propset.SetProperty("VERSION","SDE.DEFAULT");
propset.SetProperty("AUTHENTICATION_MODE","DBMS");

IWorkspace workspace = sdeWkspFact.Open(propset, 0);
```

- Connecting using a connection file

```
string nameOfFile = "D:\\data\\redarrow.sde”;
IWorkspace workspace = workspaceFactory.OpenFromFile(nameOfFile, 0);
```

Geoatabase specific information

- Some aspects of Geodatabases differ from one another
- Don't assume field names remain constant
  - Can differ between geodatabases
  - IFeatureClass::ShapeFieldName property
- ISQLSyntax interface can be used to determine SQL predicates, clauses, and other database specific constraints are supported by a workspace
  - ISQLSyntax.GetSpecialCharacter(esriSQL_DelimitedIdentifierPrefix)
  - ISQLSyntax.GetFunctionName(esriSQL_UPPER)
  - Important for queries
    - Can determine supported clauses for a data source
- IWorkspaceProperties

Creating Datasets

- Dataset model
  - Open methods on IFeatureWorkspace, IFeatureClassContainer, …
- Dataset Extensibility model
  - Open methods on IDatasetContainer2
- Use a name – "Streets“
  - Use ISQLSyntax::QualifyTableName for SDE
  - Owner not required to qualify name
  - IWorkspace2::NameExists
- Use a ClassID
  - ClassIDs are unique and sequential within geodatabase
- Other methods; Index, Enumerators, etc

Creating Datasets …

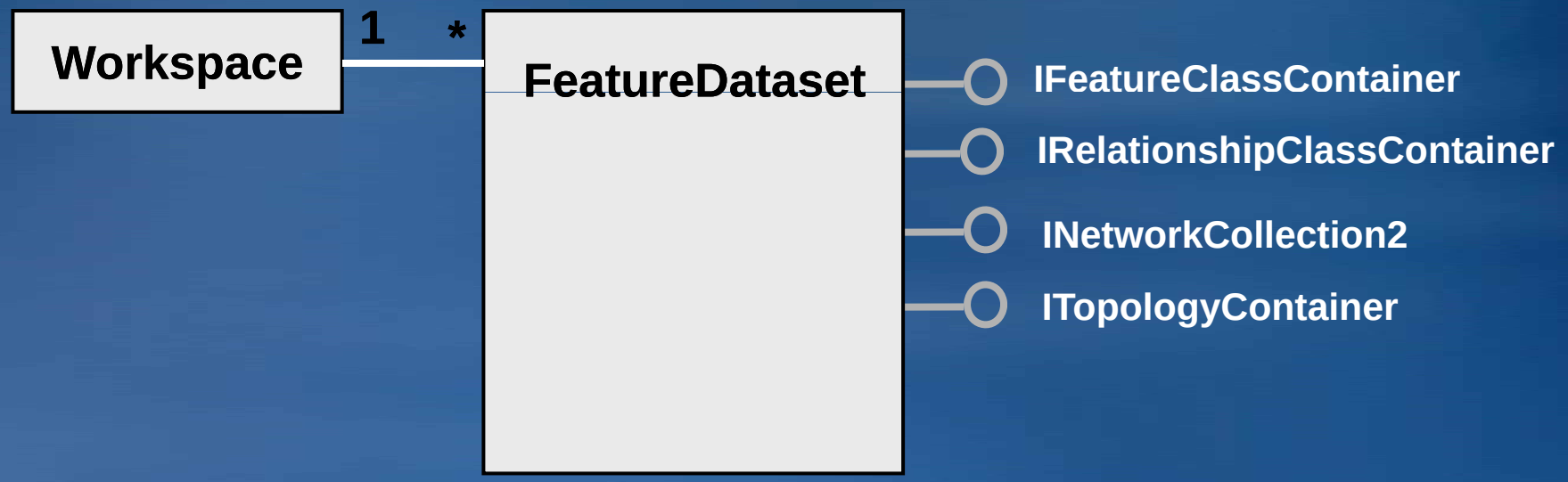
- Create methods to match the open methods
  - FeatureClass
  - FeatureDataset
  - Table
  - ...
- Dataset model
  - Properties of dataset to be created arguments to Create method
- Dataset Extensibility model
  - Use a Data Element that has been pre-populated
  - For Network Datasets, Terrains, Representations, Cadastral Fabrics

Creating Datasets …

- Creation of datasets is a type of DDL command
  - Data definition language - database commands that modify the schema of a database
  - Should never be called inside of an edit session
  - Commit any transactions that are currently open, making it impossible to rollback any unwanted edits should an error occur

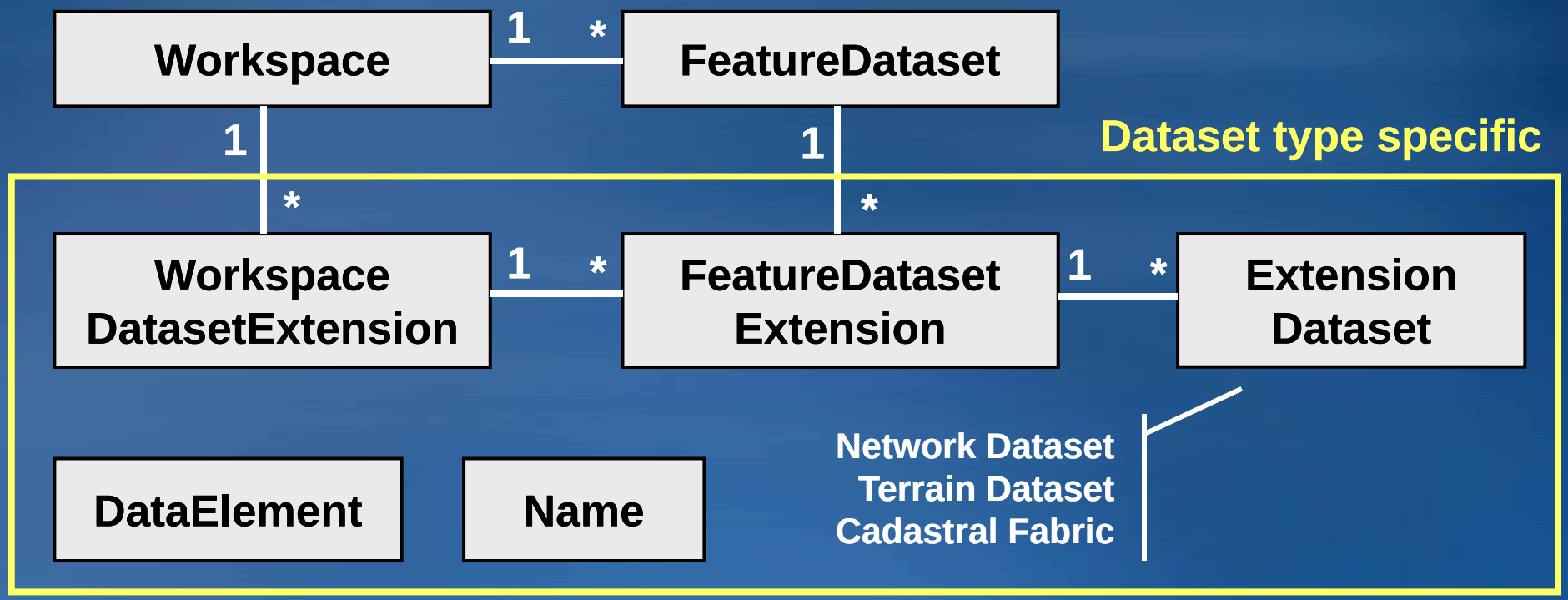
Dataset Model

- Extend existing objects with new interfaces for each specific dataset



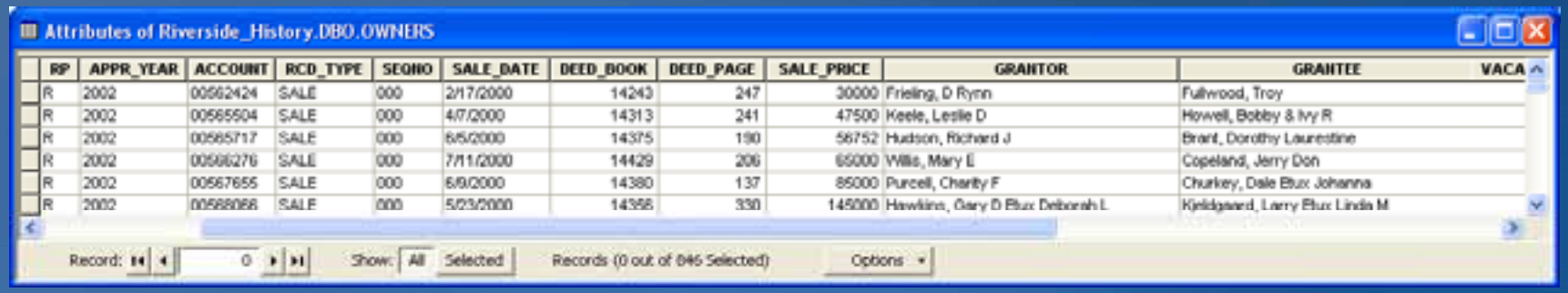
Dataset Extensions Model

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



Rows and Tables

- **Type of Dataset in the Geodatabase**
  - Containing zero to many rows
  - One to many columns
  - All rows in the table have the same schema
- **ITable interface provides Table management abilities**
  - View and modify schema
  - Add and remove rows
  - Perform queries



Attributes of Riverside_History.DBO.OWNERS

RP	APPR_YEAR	ACCOUNT	RCD_TYPE	SEQNO	SALE_DATE	DEED_BOOK	DEED_PAGE	SALE_PRICE	GRANTOR	GRANTEE	VACA
R	2002	00562424	SALE	000	2/17/2000	14243	247	30000	Frieling, D Rynn	Fullwood, Troy	
R	2002	00565504	SALE	000	4/7/2000	14313	241	47500	Keele, Leslie D	Howell, Bobby & Ivy R	
R	2002	00565717	SALE	000	6/5/2000	14375	190	56752	Hudson, Richard J	Brant, Dorothy Laurestine	
R	2002	00566276	SALE	000	7/11/2000	14429	206	65000	Willis, Mary E	Copeland, Jerry Don	
R	2002	00567655	SALE	000	6/9/2000	14380	137	85000	Purcell, Charity F	Churkey, Dale Blux Johanna	
R	2002	00568066	SALE	000	5/23/2000	14356	330	145000	Hawkins, Gary D Blux Deborah L	Kjeldgaard, Larry Blux Linda M	

Record: 0 Show: All Selected Records (0 out of 846 Selected) Options

Objects and Object Classes

- Objects are entities with properties and behavior
  - Extend the Row concept
- An Object Class is a Table with Geodatabase behavior
  - Registered with the Geodatabase
  - An object is an instance of an object class
  - All objects in an object class have the same properties and behavior
  - An object can be related to other objects via relationships.

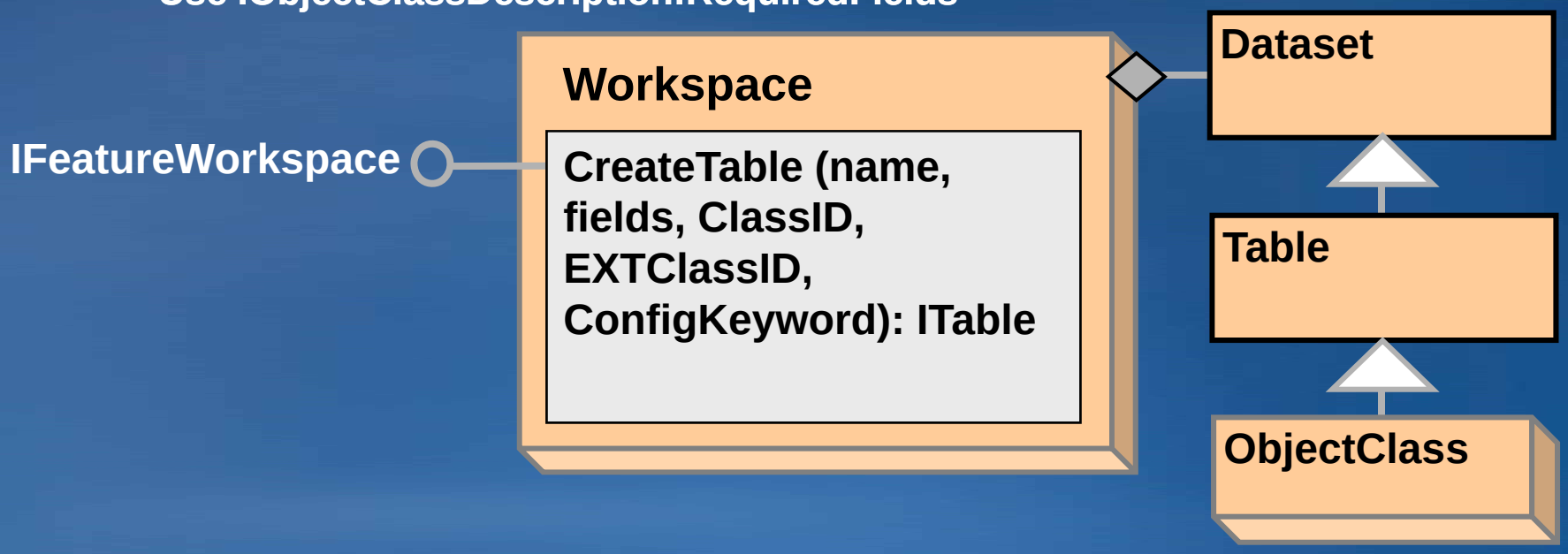
Attributes of Riverside_History.DBO.OWNERS

OBJECTID	RP	APPR_YEAR	ACCOUNT	RCD_TYPE	SEQNO	SALE_DATE	DEED_BOOK	DEED_PAGE	SALE_PRICE	GRANTOR	GRANTEE	VACA
361	R	2002	00562424	SALE	000	2/17/2000	14243	247	30000	Frieling, D Rynn	Fullwood, Troy	
362	R	2002	00565504	SALE	000	4/7/2000	14313	241	47500	Keele, Leslie D	Howell, Bobby & Ivy R	
363	R	2002	00565717	SALE	000	6/5/2000	14375	190	56752	Hudson, Richard J	Brant, Dorothy Laurestine	
364	R	2002	00566276	SALE	000	7/11/2000	14429	206	65000	Willis, Mary E	Copeland, Jerry Don	
365	R	2002	00567655	SALE	000	6/9/2000	14380	137	85000	Purcell, Charity F	Churkey, Dale Etux Johanna	
366	R	2002	00568066	SALE	000	5/23/2000	14356	330	145000	Hawkins, Gary D Etux Deborah L	Kjeldgaard, Larry Etux Linda M	

Record: 0 Show: All Selected Records (0 out of 846 Selected) Options

Create an Object Class

- Creates an ObjectClass, returns ITable interface
  - ObjectID field. Values are Never Reused.
  - Controlled by the Geodatabase
  - From a database perspective, can be viewed as a primary key
- Can also use it to create a Table
- Custom behavior ID's *(can use null for EXTClassID)*
- SDE configuration keyword *(can use "")*
- Can create Fields first
  - Use IObjectClassDescription:RequiredFields



Create an Object Class …

- Object Classes have geodatabase specific functionality that Tables do not, through several interfaces
  - IObjectClass
    - AliasName
    - ObjectClassID
    - RelationshipClasses
  - ISubtypes
    - AddSubtype
- Standard table operations, such as queries and row creation, are still performed on an object class using the ITable interface.

Features and Feature Classes

- Builds on the Relational Model
  - Extends the Object Class concept with a spatial attribute
- A feature is a spatial object.
- A feature is an instance of a feature class.
- Extended the relational model with
  - Geometry attribute types

A feature class is a table of rows, where each row has a geographic column

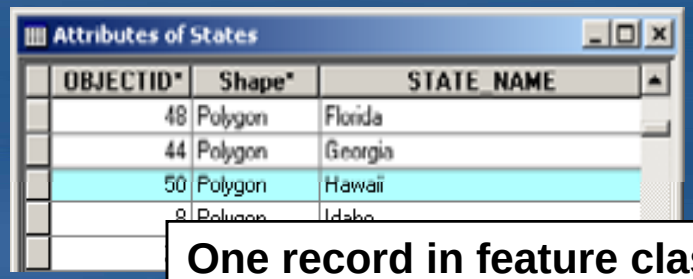
Attributes of Parcels

OBJECTID*	SHAPE*	PARCEL_ID*	ZONE_CODE*	SHAPE_Length	SHAPE_Area
4513	Polygon	67970	W	544.053559	9259.209935
4514	Polygon	67971	W	158.545394	774.602847
4515	Polygon	67973	R60M	400.003008	7499.965473
4516	Polygon	67974	B1	236.126101	2905.890606
4517	Polygon	67982	B1	550.458538	17499.011493

Geodatabase Supports Advanced Geometry

- Points, lines, polygons, text, and surfaces
- Single and multipart features

Feature with many parts



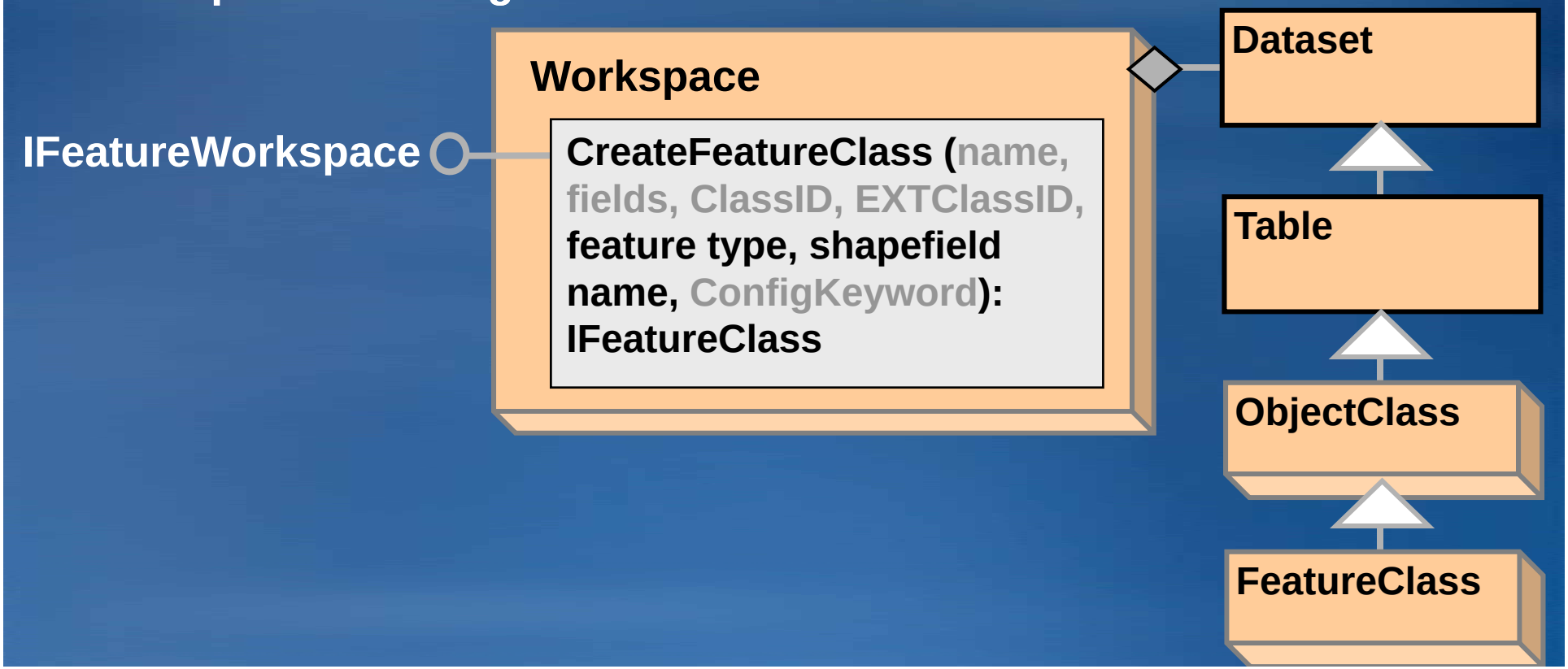
OBJECTID*	Shape*	STATE_NAME
48	Polygon	Florida
44	Polygon	Georgia
50	Polygon	Hawaii
8	Polygon	Idaho

One record in feature class table

- Flexible coordinates
  - XY, Z, M, True Curves

Create a Feature Class

- Same as CreateTable except:
  - Needs Shape type and Shape field name
    - IGeometryDefEdit used when defining new feature class
    - Use IObjectClassDescription:RequiredFields
  - Spatial Indexing considerations



Create a Feature Class …

- Feature classes that store simple features can be organized either inside or outside a feature dataset
  - Those outside a feature dataset are called standalone feature classes
  - Feature classes which store topological features, for example those participating in geometric networks, must be contained within a feature dataset to ensure a common spatial reference
- Each dataset in a geodatabase must have a unique name for the type of dataset
  - Feature classes must have a unique name

Table – Object Class – Feature Class

	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass. CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

Table – Object Class – Feature Class

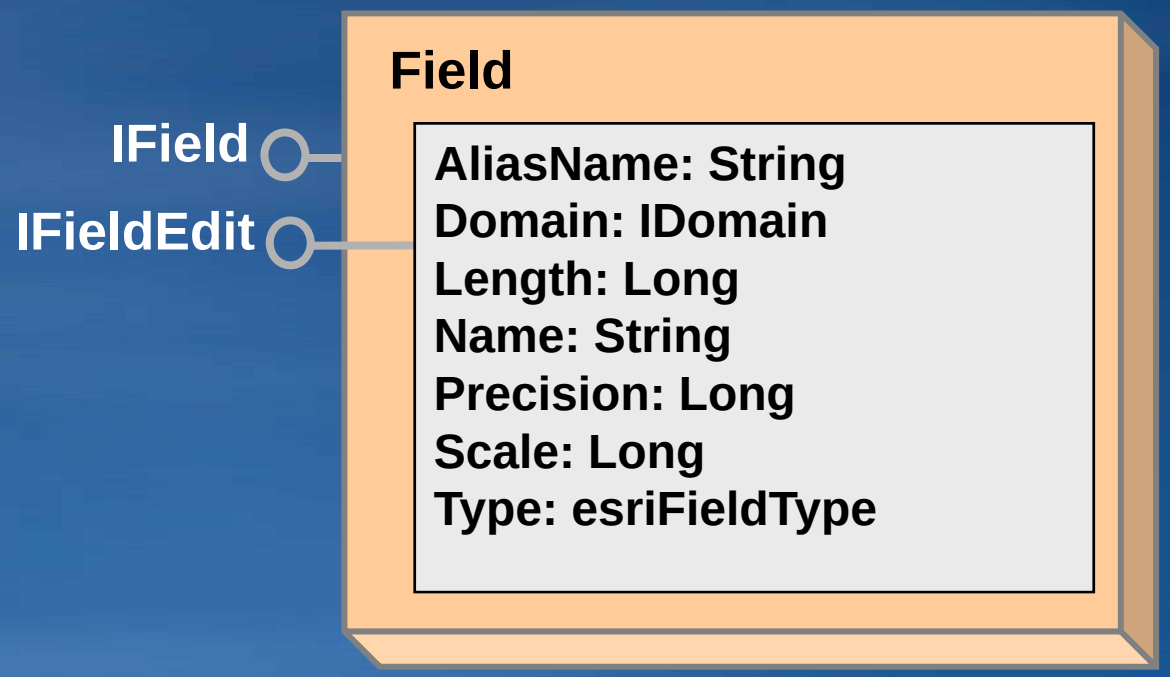
	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass. CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

Table – Object Class – Feature Class

	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass. CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

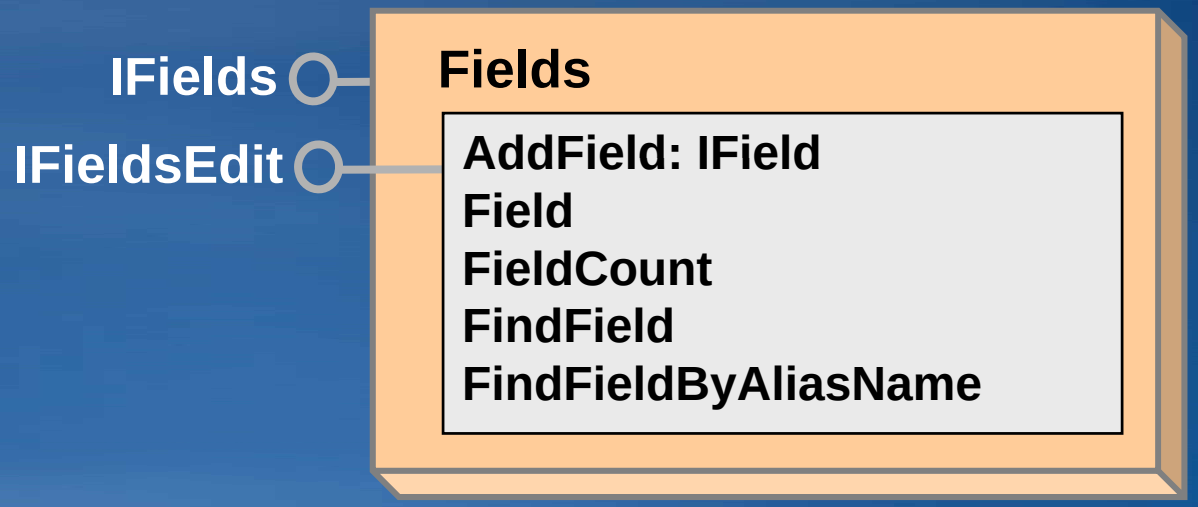
Create a Field

- Synonymous with a column
- Set properties with IFieldEdit
- Types include: Integer, Single, Double, String, Date, OID, Geometry, Blob, GUID, GlobalID, and Raster
- Geodatabase tables must have an ObjectID field
  - Using Class Descriptions will take care of this



Create a Field …

- Use a Field collection (Fields) when creating datasets
  - For existing tables use IClass::AddField method to add fields and IClass::DeleteField method to delete fields
- Set properties for the Field with the IFieldEdit interface
- Leverage Class Description whenever possible
  - ObjectClassDescription, FeatureClassDescription, etc

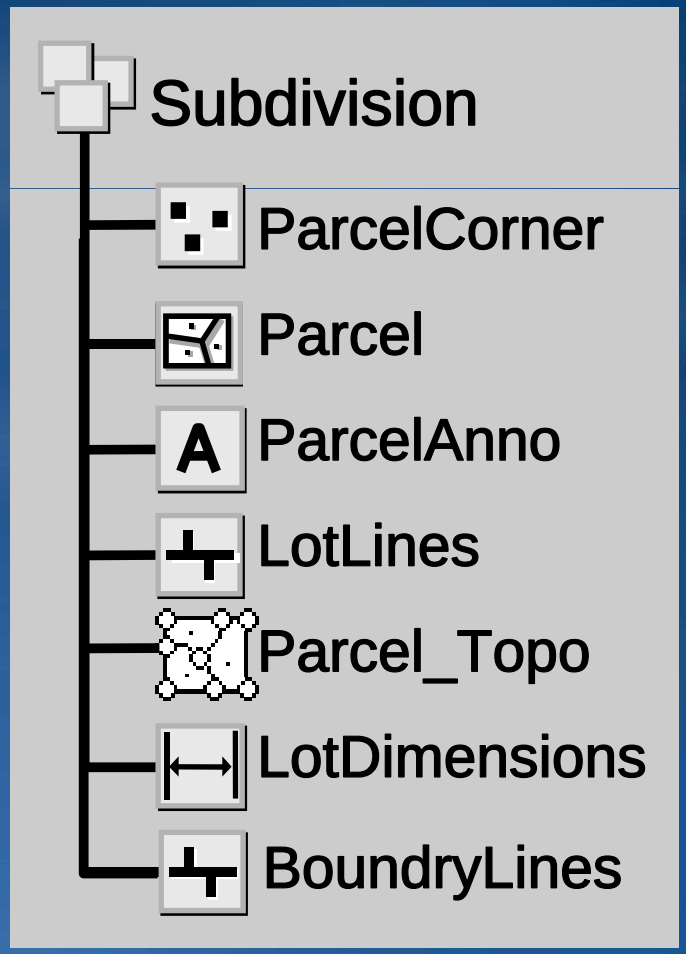


Create a Field …

- IFieldChecker has methods for validating fields collections and table names for a specific workspace
- Use it prior to creating Tables and Feature Classes
  - From scratch
  - Based on another Table or Feature Class
- Validate and ValidateField
  - Field name and data types
- ValidateTableName
  - Table name does not start with or contain invalid characters

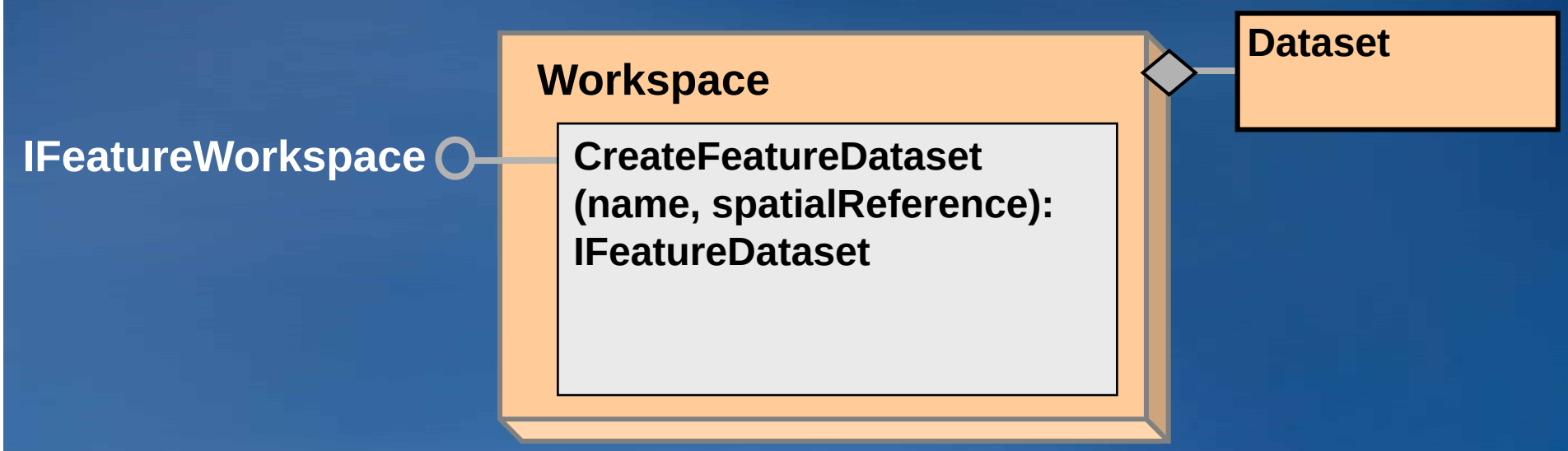
Feature Datasets

- A container object for datasets with the <u>same spatial reference</u>
  - Only exist within a Geodatabase
- Analogous to a coverage
  - Less restrictive
- Container for:
  - Geometric networks
  - Topologies
  - Network Datasets
  - Terrains
  - Cadastral Fabrics
  - Optionally relationship classes



Create a Feature Dataset

- Allows the creation of a Feature Dataset:
  - Methods supported by the returned feature dataset allow the creation of feature classes within the feature dataset
  - Can also be used to access datasets through the Dataset model
- Spatial Reference is required



Feature Datasets …

- Need to remember that feature classes may or may not belong to a feature dataset
  - The following code assumes a feature dataset exists and may fail:

```
IFeatureDataset featureDataset = featureClass.FeatureDataset;
IWorkspace workspace = featureDataset.Workspace;
```

  - This piece of code will work for standalone feature classes and those in feature datasets:

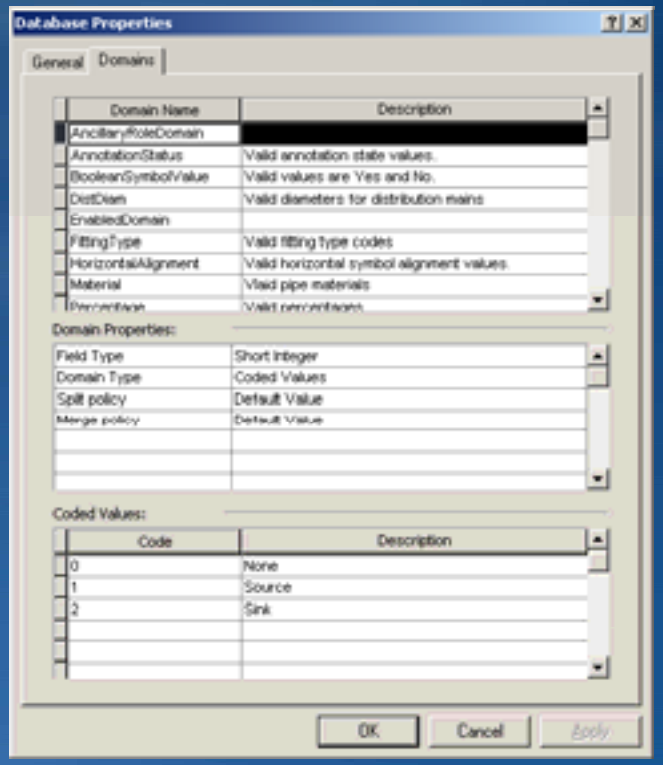
```
IDataset dataset = (IDataset)featureClass;
IWorkspace workspace = dataset.Workspace;
```

Demo

- Tour around a geodatabase – Part 2
- Create a file geodatabase (.gdb)
- Create a feature dataset and feature class
- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

Domains

- Describe the legal values of a field type.
  - Used to ensure attribute integrity
- Defined at the workspace level
- Domains constrain field values
- Associate a domain to a field(s)
  - During create use IFieldEdit:Domain
  - Or IClassSchemaEdit:AlterDomain

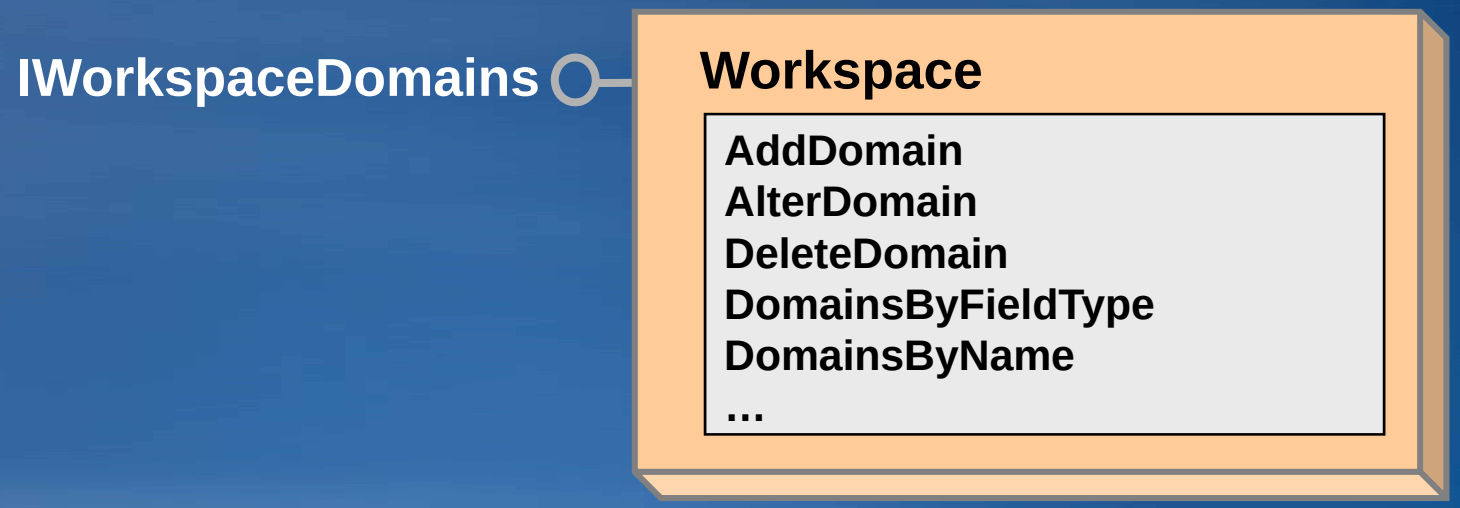


Domains …

- Types of domains:
  - Range
    - A tree can have a height between 0 and 300 feet.
    - A road can have between one and eight lanes.
  - Coded Value
    - A tree can be of type oak, redwood, or palm.
    - A road can be made of dirt, asphalt, or concrete.

Domains …

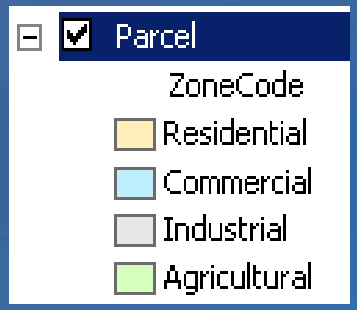
- Use the IWorkspaceDomains interface to manage the collection of domains found within a workspace.
  - Since Domains are shared amongst feature classes; the management of them is at the workspace level
- DeleteDomain will fail if the domain is associated with a field
- Domain names are unique across a workspace
  - Need to check for existence of Domain name prior to creation or trap for the error



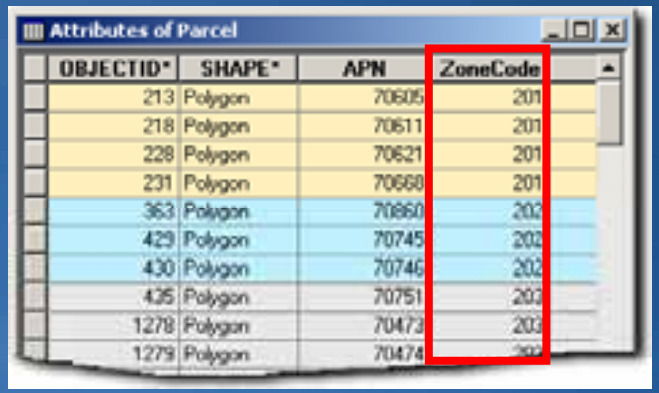
Subtypes

- Subtypes are specific to feature class
  - Partition the objects in a class into like groups
  - Can be used with Tables, Object Classes and Feature Classes
- Defined at the class level
  - Can only be Short or Long Integers
- Defined by the value of a subtype field.
  - Have the same attribute\behavior schema
  - Can have different default values and domains for each field
  - Can define topology rules between subtypes

Descriptions

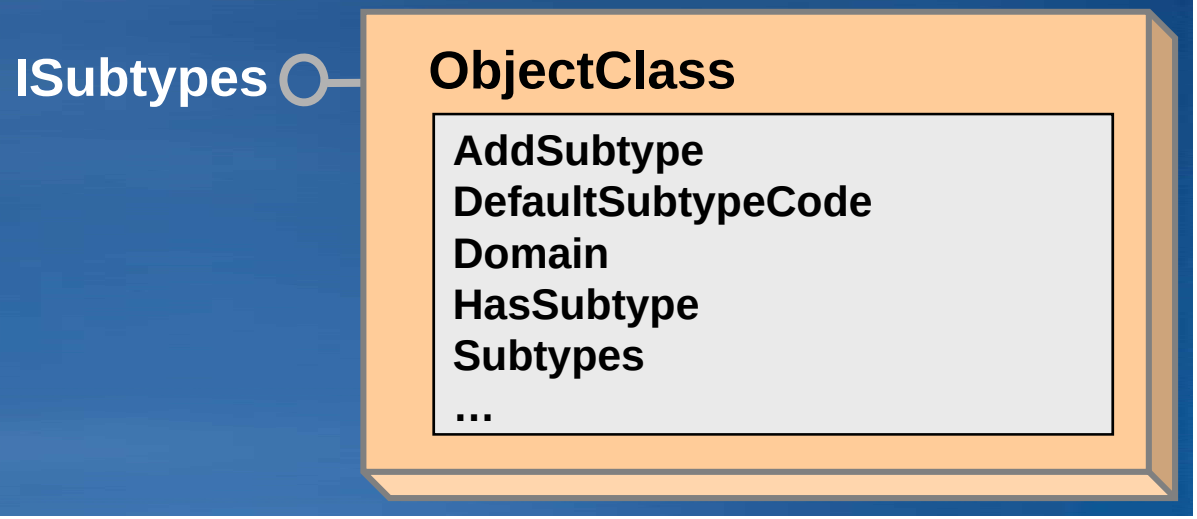


Codes



Subtypes …

- Once set, need to check attribute, connectivity and relationship rules at subtype level
- Each object class has a default subtype code
  - Important for feature creation and editing
- Subtypes are the first to be checked during Validation
- The ISubtypes interface is used for managing subtypes and the associated default values and attribute domains



Subtypes …

- Basic process for setting Subtypes
  - Define subtype field
    - ISubtypes:SubtypeFieldName
  - Check valid values of field
    - For populated classes
  - Assign Subtype code and name
    - ISubtypes:AddSubtype (code, name)
  - ISubtypes:DefaultSubtypeCode
    - Define default subtype code
  - Assign Default Values and Domains at the Subtype level
    - ISubtypes:DefaultValue
    - ISubtypes:Domain

Validation Rules

- Store attribute, connectivity, and relationship rules on objects as part of the geodatabase.
- Predefined, parameter driven
  - Subtypes
  - Attribute range rule or Attribute set rule
  - Connectivity rule
  - Relationship rule
- Rules are evaluated at a user specified time
  - Not performed when the feature is created, stored, edited, etc
  - Geodatabase has an optimistic view of the data
  - Allows you to load your data; then validate and correct it

Validation Rules

- Use the IValidate::Validate or IValidation::Validate methods to evaluate rules
  - There is an order to how rules are validated:
    - Validate the subtype
    - Validate the attribute rules
    - Validate the network connectivity rules (if network feature)
    - Perform custom validation (using optional class extension)
    - Validate the relationship rules
  - Validation stops once a check returns false
- Perform custom validation by writing code

Schema Locks

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

Schema Locks …

- When to use?
- You must promote a shared lock to exclusive when performing schema modification such as:
  - Modifications to attribute domains; coded or range
  - Adding or deleting a field to a feature or object class
  - Associating a class extension with a feature class
  - Creating a Topology, Geometric Network, Network Dataset, Terrain, Schematic Dataset, Representation or Cadastral Fabric on a set of feature classes
  - Any use of the IClassSchemaEdit interfaces
  - IFeatureClassLoad.LoadOnlyMode
  - Rebuilding spatial and attribute indexes

Schema Locks …

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

Working with Indexes

- Use the IIndexes interface to access the indexes for a specific table or feature class
  - Obtained from a table or feature class by using the IClass.Indexes
  - Operates in a similar manner to the IFields collection object

```
public void DisplayIndexCount(ITable table)
{
        IIndexes indexes = table.Indexes;
        Console.WriteLine("The table has an index count of {0}.", indexes.IndexCount);
}
```

- Use the IIndexEdit interface to create a new index
  - Operates in a similar way to IFieldEdit
  - Use the IClass.AddIndex method to add it to the class

Attribute Indexes

- Based on an ordered list of one or more fields in a table
  - List order determines which field is used first with queries
  - Limit of 10 fields in a geodatabase attribute index
  - File geodatabases does not support multi-field indexes
- All object or feature classes have one attribute index:
  - ObjectID field
  - Added by the Geodatabase
- Indexes created in the native environment of the database management system (DBMS) can also be accessed.
- For shapefiles, both spatial and attribute indexes can be manipulated
  - Limit of one field in an attribute index

Spatial Indexes

- Associated with the Shape field of a feature class
- Created automatically when a geodatabase feature class is created
- Spatial indexes have GridSize and GridCount properties
  - Initial grid size of 0 allows ArcGIS to determine the initial grid size
  - R-Tree spatial index (e.g. Oracle Spatial, Informix, PostgreSQL) return a value of -2
- Possible to delete and re-create the spatial index with the IFeatureClass.DeleteIndex and AddIndex methods
  - Not support for personal geodatabase feature classes

Data Access and Creation demo …

- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

Creating Rows and Features

- Basic process to create row or feature
  - CreateRow or CreateFeature
    - Can also use InsertCursor, more later
  - If subtypes present, set IRowSubtypes::SubtypeCode
  - If default values, call IRowSubtypes::InitDefaultValues
  - Set attribute values
  - Create geometry and set Shape
  - Call Store
    - Writes the values to the record in the table

Creating Rows and Features

```
public void CreateFeature(IFeatureClass featureClass, IPoint point)
{
  // Ensure the feature class contains points.
  if (featureClass.ShapeType != esriGeometryType.esriGeometryPoint)
  {
          return ;
  }
  // Build the feature.
  IFeature feature = featureClass.CreateFeature();
  feature.Shape = point;
  // Apply the appropriate subtype to the feature.
  ISubtypes subtypes = (ISubtypes)featureClass;
  IRowSubtypes rowSubtypes = (IRowSubtypes)feature;
  if (subtypes.HasSubtype)
  {
          // In this example, the value of 3 represents the Cross subtype.
          rowSubtypes.SubtypeCode = 3;
  }
  // Initialize any default values the feature has.
  rowSubtypes.InitDefaultValues();
  // Update the value on a string field that indicates who installed the feature.
  int contractorFieldIndex = featureClass.FindField("CONTRACTOR");
  feature.set_Value(contractorFieldIndex, “D. Heatley");
  // Commit the new feature to the geodatabase.
  feature.Store();
}
```

Simple vs. Complex Features

- Within the geodatabase, behavior is dependent upon whether a feature is simple or complex
- Simple features
  - Point, line, polygon, multipoint, multipatch features
  - Simple Relationships
- Complex features
  - Network features (simple edge, simple junction, complex edge)
  - Annotation features
  - Dimension features
  - Raster catalog
  - Custom features

Simple vs. Complex Features …

- Editing
  - You can perform non versioned edits on simple data only
    - points, lines, polygons, annotation, and relationships
  - You cannot perform non versioned edits on complex data such as feature classes in a topology, network dataset, or geometric network
  - Any dataset specific behavior; ie: for features created in geometric networks, topologies, etc; is handled at creation time
    - Not required to call Connect or create Dirty Areas

Simple vs. Complex Features …

- Cursors
  - Insert cursors can perform direct inserts outside of an edit session on simple data
    - Same rule applies to update cursors
    - Offers performance advantages; i.e.: events not fired
  - Using these APIs on complex objects (or on objects participating in composite relationships or relationships with notification) negates any performance advantages

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Geodatabase Editing - Edit Session

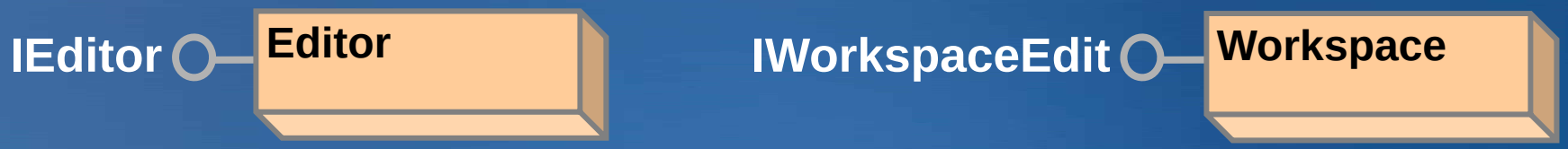
- Geodatabase explicitly stores change information when edited
- Only see the changes you've made within the edit session
  - Changes made by other applications are not seen
  - Until Save or Discard
- Edits should be made within an edit operation
  - StartEditOperation – StopEditOperation
  - Perform the edit as quickly as possible
  - Keep edit operation "tight and compact"
  - Collect the required information before starting the edit operation

Geodatabase Editing - Edit Session …

- Each edit operation represents a transaction
  - Stop commits the change
  - Abort rolls back, like undo
- Applications are responsible for calling:
  - AbortEditOperation when errors are detected
  - StopEditOperation to complete edit operations
    - Pushes the edit operation onto the undo stack
- UndoEditOperation, RedoEditOperation
  - Geodatabase moves the operation between the undo and redo stacks

Editing the Geodatabase

- When to use edit sessions?
  - Always…
  - Must use with topologies, geometric networks, terrains, etc
  - Use IObjectClassInfo2::CanBypassEditSession
- When to use IEditor or IWorkspaceEdit?
  - Use IEditor to edit within an application, like ArcMap
  - Ensures undo/redo consistency between edits made programmatically and through the UI
  - Must use IWorkspaceEdit in Engine environment
- Similar methods on each



Other Useful Method When Editing

- IDatasetEdit.IsBeingEdited
  - Determine if a particular dataset is participating in the edit session
- IWorkspaceEdit2.IsInEditOperation
  - Determine if the workspace is currently in an edit operation
  - Use when deciding whether to start an edit operation
- IWorkspaceEdit2.EditDataChanges
  - Determine which features have been changed with the scope of an edit session or edit operation.

Editing Demo

- Update Feature
- Edit Operations

Presentation Outline

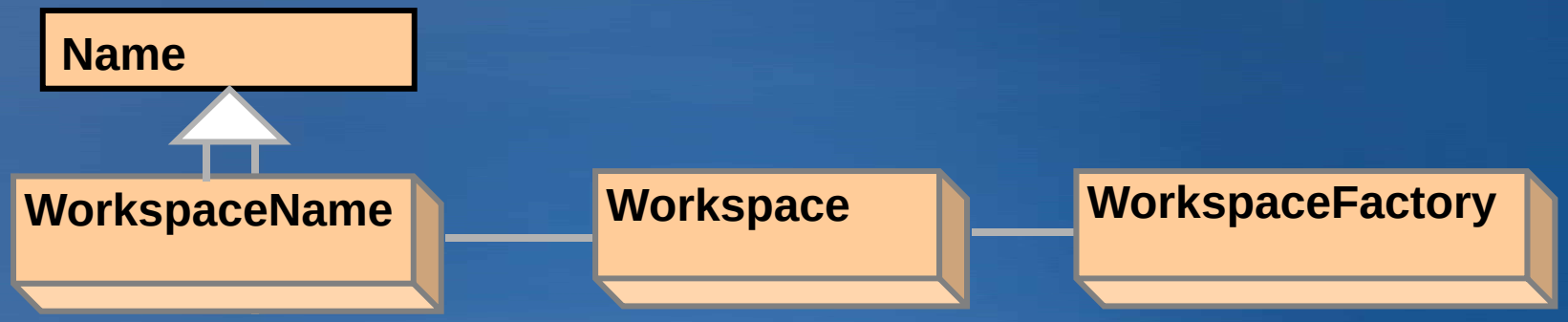
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Beyond Basics

- Name Objects
- Data Elements
- Event Model
- Relationship classes
- Annotation and Dimension feature classes
- Topology
- Geometric Networks
- Network Datasets
- Raster Datasets and Raster Catalogs
- Geodatabase XML

Name Objects

- Light-weight representation of real object
  - Name objects for each dataset
  - Acts as a moniker to the dataset
- Can be persisted
- Allows you to bind to the real object
  - You can get some properties from the name object
  - Can open dataset from Name object
  - Note: Opening feature classes in complex datasets (i.e: geometric networks) will open all feature classes in the dataset



DataElement Objects

- Used in geoprocessing functions, Web services Dataset Extensibility
  - Describe actual geodatabase datasets
  - DEFolder, DETable, and DEShapeFile, etc
- Simple structures whose properties describe the actual entity
  - Different from Name objects, you cannot directly open the dataset
- Support *IXMLSerialize* and *IPersistStream*
  - Can be serialized in XML or binary form

Geodatabase Event Model

- Types of events
  - Feature events
  - Class extension events
  - Object class 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

Geodatabase Event Model …

- No need to call Store within feature level event
  - Call to Store the object again can lead to unexpected behavior

```
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!
}
```

Geodatabase Event Model

- Some events may either be listened to or may be supported in object class extensions; e.g.,
  - IObjectClassEvents
  - ITopologyClassEvents
- Listenable event interfaces includes…
  - IWorkspaceEditEvents
  - IObjectClassSchemaEvents
  - IObjectClassEvents

Relationship Classes

- What is a Relationship Class?
  - An association between two object classes that is persisted within the Geodatabase
  - A class may participate in multiple relationship classes.

Parcel

OID	Zone	Area
28	Commercial	10000
794	Residential	5000
858	Residential	6050

Buildings

OID	Parcel_ID	Building Type
1	28	Office Building
2	794	Townhouse
3	794	Townhouse
4	050	Condo
5	858	Condo

Relationship Classes …

- What is a Relationship Class?
  - An association between two object classes that is persisted within the Geodatabase
  - A class may participate in multiple relationship classes.

ParcelToBuilding

Parcel

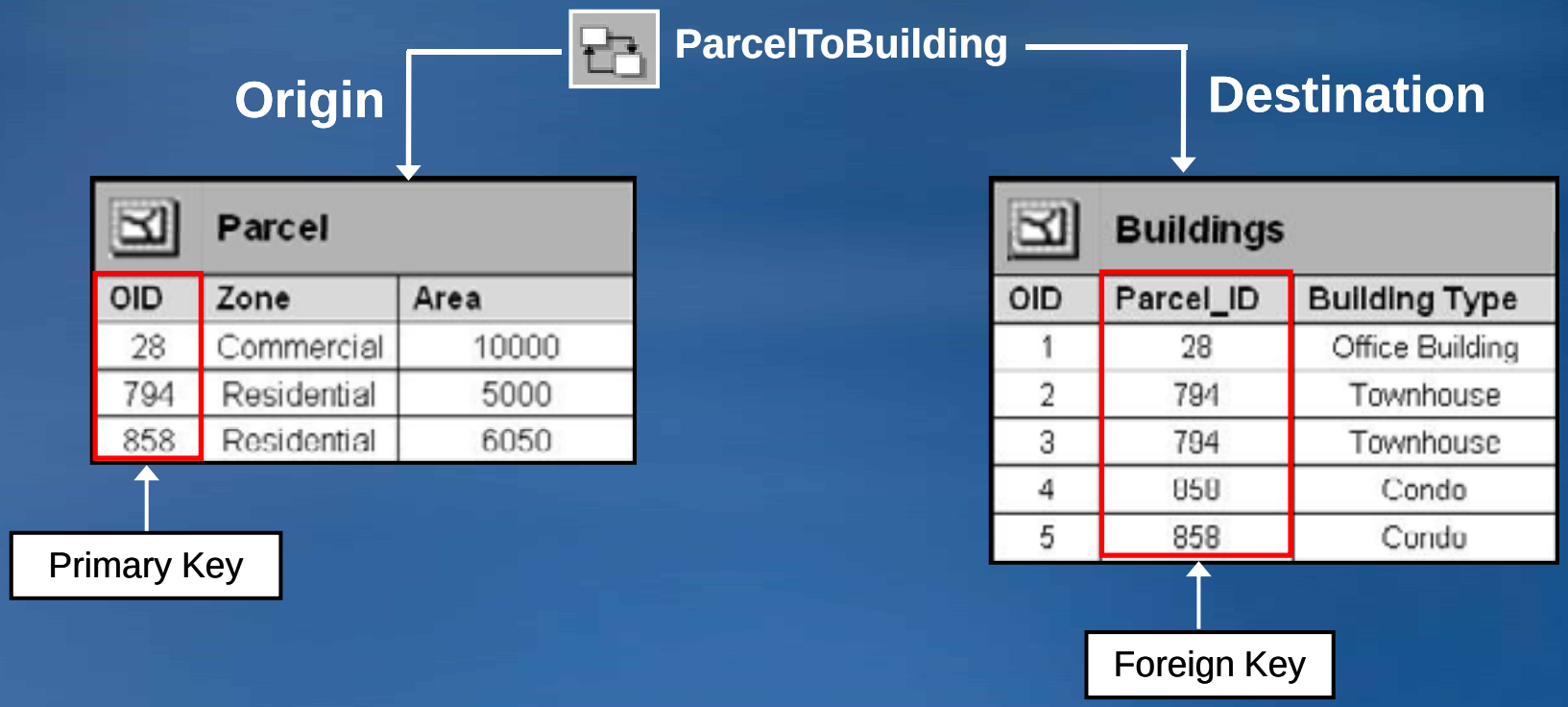
OID	Zone	Area
28	Commercial	10000
794	Residential	5000
858	Residential	6050

Buildings

OID	Parcel_ID	Building Type
1	28	Office Building
2	794	Townhouse
3	794	Townhouse
4	050	Condo
5	858	Condo

Relationship Classes …

- What is a Relationship Class?
  - An association between two object classes that is persisted within the Geodatabase
  - A class may participate in multiple relationship classes.



Types of Relationship Classes

- Simple
  - Related objects can exist independently of each other.
  - When an origin feature is deleted the Foreign key in the destination is set to Null.
- Composite
  - Related objects are dependent on the lifetime of the origin objects.
  - When an origin feature is deleted all of the related features in the destination are also deleted

Why use a Relationship Class?

- Stored in the Geodatabase
- Navigating
  - Identify related objects
- Editing
  - Enforces referential integrity
  - Facilitate editing with automatic updates
  - Related objects can message each other which can trigger specific behavior (cascade deletes, move to follow, custom)
  - Relationship rules define and control how objects relate
  - Relationship rules can be enforced through Validation

How to create a Relationship Class?

- In the Root level of the Geodatabase
  - IFeatureWorkspace.CreateRelationshipClass
- In a Feature Dataset
  - IRelationshipClassContainer.CreateRelationshipClass
- You set parameters to define the relationship
  - Origin and destination tables (Object Classes)
  - Primary and Foreign keys (Field names)
  - Cardinality (1:1, 1:M or M:N)
  - Type of Relationship (Simple or Composite)
  - Attributes, Messaging, labels, etc
- IRelClassSchemaEdit
  - Can only change labels, and relationship type

Identifying Related Features

- Once a relationship has been created you work with it using the methods on the IRelationshipClass2 interface.
- Identify related objects using:
  - GetObjectsRelatedToObject
  - GetObjectsRelatedToObjectSet
- Identify the relationship that associates objects using:
  - GetRelationship
  - GetRelationshipsForObject
  - GetRelationshipsForObjectSet
- Identify related object pairs using:
  - GetObjectsMatchingObjectArray
  - GetObjectsMatchingObjectSet
  - GetObjectsMatchingObjectSetEx

Relationship Class Messaging

- Used to notify related objects of changes
  - So further behavior can occur
- Does come at a cost
  - edits and inserts to datasets that trigger notification is noticeably slower than the same operation on datasets that do not trigger any notification
- For inserts, ensure that all notified classes are opened prior to inserts
  - Failing to open the notified class may cause performance to degrade by an order of magnitude per class

Relationship Class Demo

PolestoTransformers

Origin → **Destination**

Utility Poles

OID	Material	Height
1	Wood	50
2	Wood	50
3	Steel	60
4	Wood	50

Primary Key

Transformers

OID	Pole_ID	Installation Date
1	1	3/2/2006
2	1	3/25/2006
3	1	5/29/2006
4	2	2/15/2007
5	2	9/8/2007

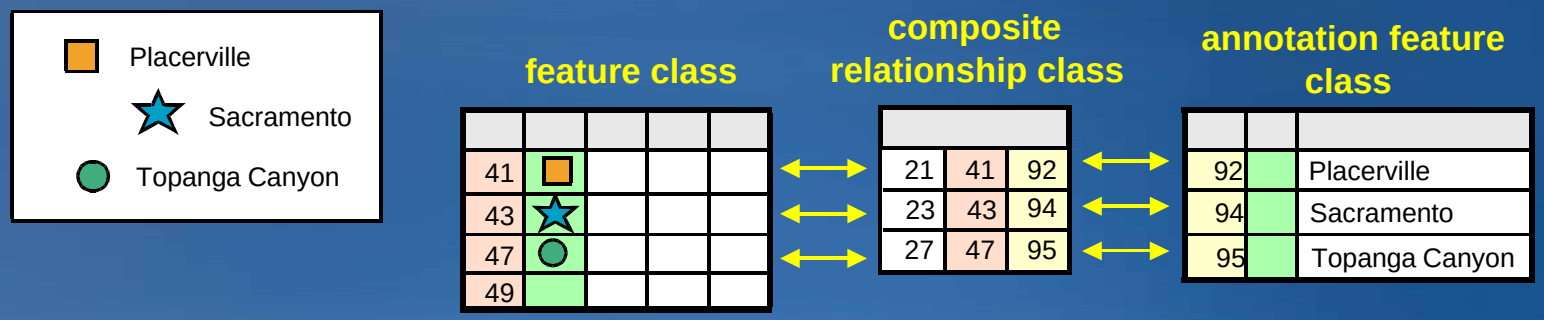
Foreign Key

Relationship Rules:

1. Wooden utility poles can support up to 3 transformers
2. Steel utility poles can support up to 5 transformers

Annotation

- Annotation feature classes may be …
  - Feature linked annotation or standard annotation
- Feature linked annotation is managed by a composite relationship class
- Can store text as well as other graphics
  - Lines, arrows, boxes, etc.
- Labeling can be performed using Maplex or the ESRI standard label engine



Annotation …

- Support for multiple classes inside an Annotation feature class
- Annotation feature class vs. Annotation class
  - Not synonymous
  - Annotation feature class is the actual dataset in the geodatabase
  - Annotation class refers to the different classes of annotation within the feature class
    - Determines how a subset of annotation is displayed
    - Can be viewed through the feature class' properties in ArcCatalog on the Annotation Classes tab

Creating Annotation feature classes

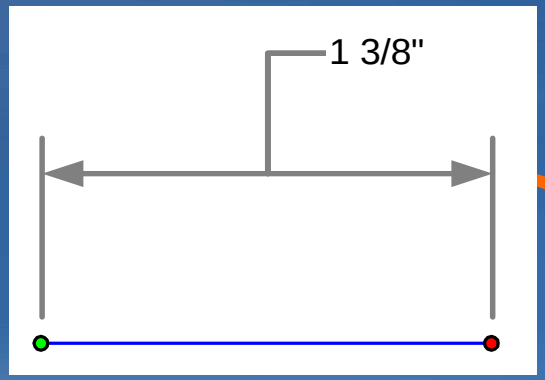
- Little in common with creating a regular feature class
- Workflow more often used for standard, can also be used for feature-linked annotation
- IAnnotationLayerFactory interface is used to generate a new annotation feature class
- CreateAnnotationLayer method of that interface requires a number of parameters to create a new class such as:
  - Feature dataset
    - Null parameter will create the class at the workspace level
  - Associated feature class
    - Null parameter will create a standard annotation feature classes
  - Overposter properties
    - Labeling placement properties for the feature class' label engine

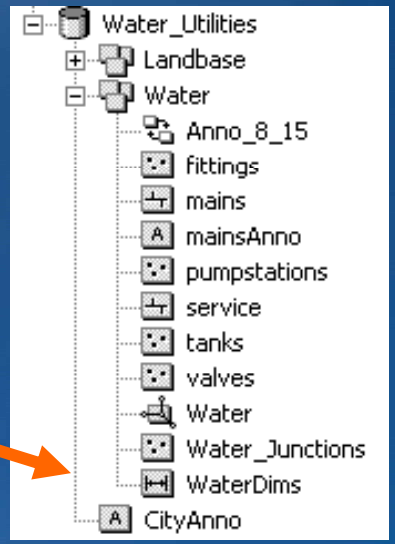
Converting Labels to Annotation

- If not using the before mentioned workflow; can use the ConvertLabelsToAnnotation class
- Coarse-grained object that encapsulates the logic needed to perform this conversion
  - Once the Feature-linked annotation class is created; new feature-linked annotation is handled with the creation of features from the associated feature class

Dimension Features

- Type of annotation that displays specific distances on a map
- Graphic features stored in a dimension feature class
- “Smart” feature
  - Special drawing
  - Special editing
- Contain styles which describe how the Dimension features will draw themselves





Creating Dimension Feature Classes

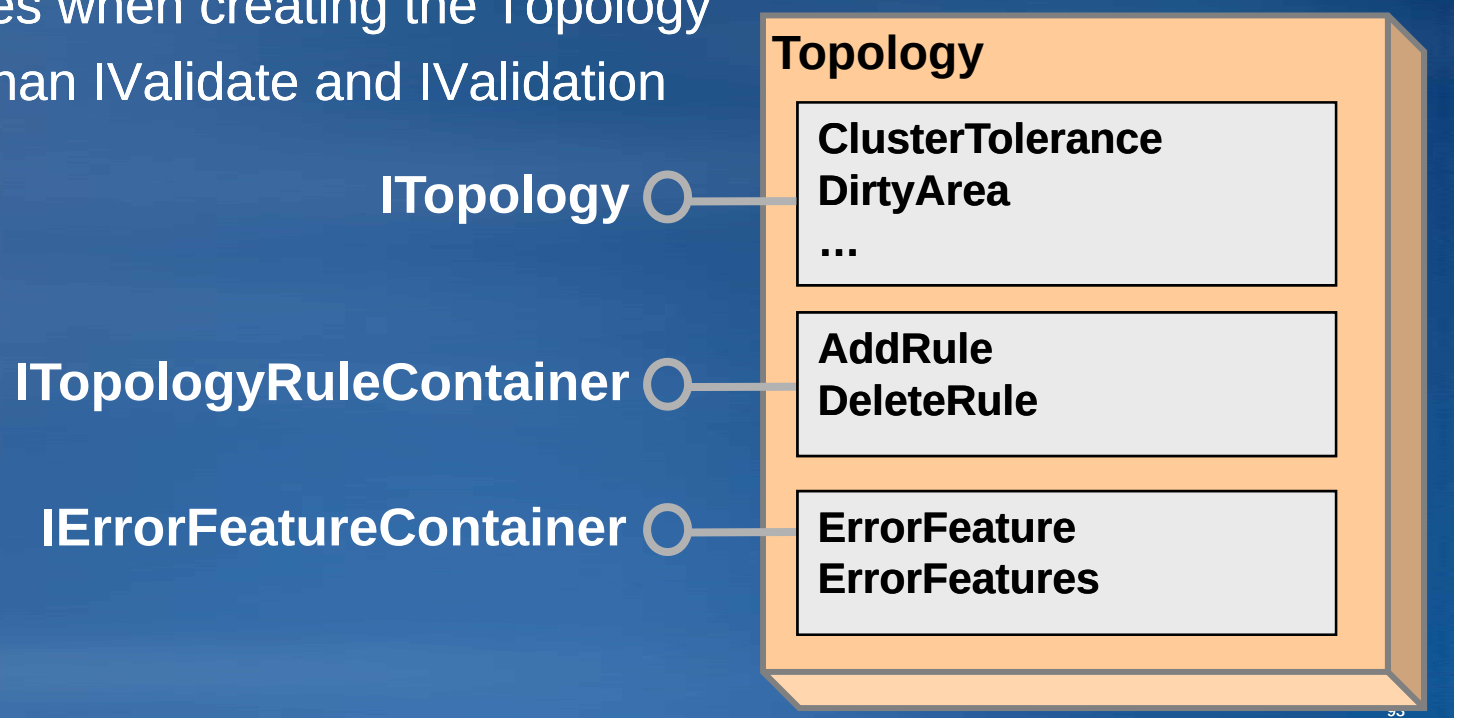
- Similar to creating a regular feature class, with three main differences:
  - Slightly different parameters for the CreateFeatureClass method
    - Fields, CLSID, EXTCLSID and FeatureType
    - Use DimensionClassDescription class as a shortcut
  - Setting extension properties through IDimensionClassExtension
    - Define the ReferenceScale and ReferenceScaleUnits
  - Dimension styles must be added to the Dimension feature class prior to use
    - Styles are maintained by the dimension class extension
    - Describe the look and feel of a dimension feature when rendered

Geodatabase Topology

- A topology manages a set of simple feature classes that share geometry
- Topology is used to
  - Integrate feature geometry
  - Validate features
  - Control editing tools
  - Define relationships between features
  - Ensure the quality of data
- Live within a Feature Dataset

Topological Integrity

- A topology defines integrity rules and constraints for the feature classes associated with the topology
  - Participating feature classes / subtypes
  - Cluster tolerance (XY and Z), ranks and rules
- Rules are evaluated during Validation
  - Define rules when creating the Topology
  - Different than IValidate and IValidation



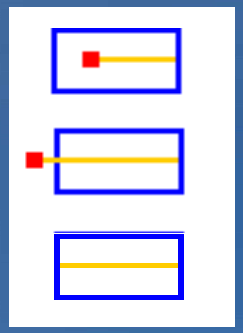
Topologies …

- Each topology has one inherent rule
  - esriTRTFeatureLargerThanClusterTolerance
  - Identifies features that are less than the defined cluster tolerance for the topology
- Other topology rules can be added and accessed through ITopologyRuleContainer
- Errors can be promoted to or demoted from exceptions through ITopologyRuleContainer
- IErrorFeatureContainer returns errors associated with topology through a number of methods:
  - Specific error feature
  - Errors associated with an instance of a topology rule
  - Errors of a particular geometry type
  - Errors of a particular rule type

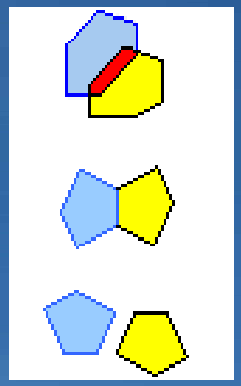
Topology Error Examples

- Rules enforced to maintain topological integrity
  - 25+ topology rules in ArcGIS
- Violations of these rules are expressed as error features managed in the database as a part of the topology
  - Error and Exceptions
  - Examine and Fix errors in ArcMap

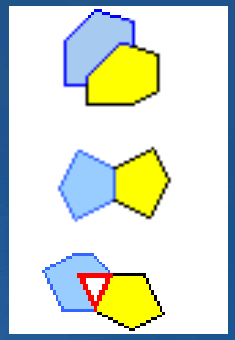
Dangling edge



Overlap



Gap



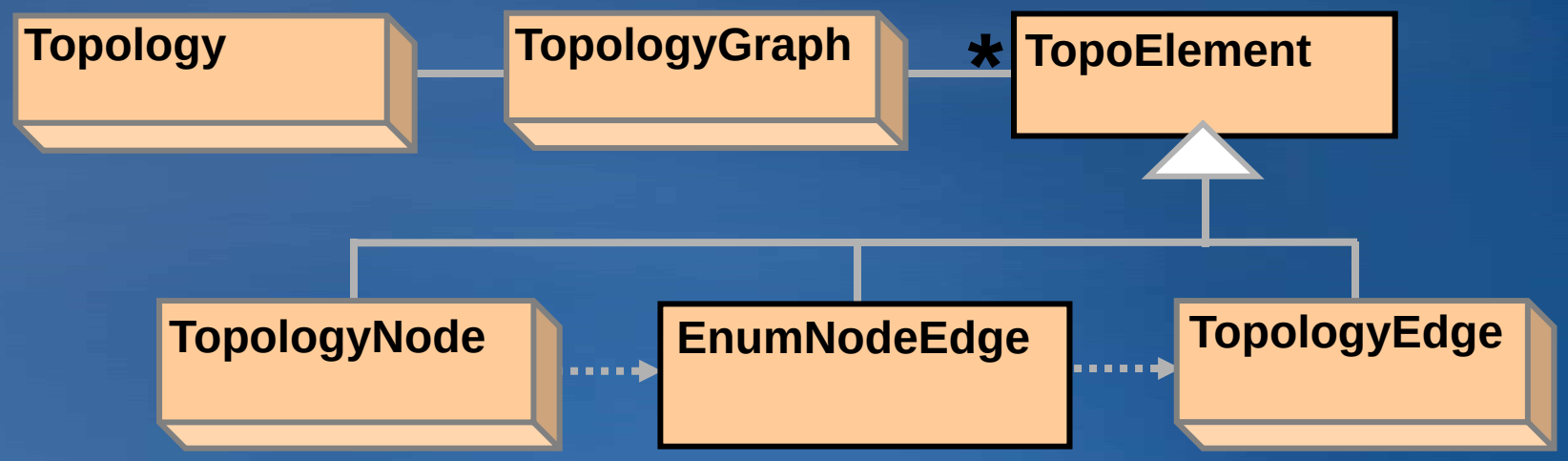
Editing with a Topology

- Editing creates a dirty area
  - Area has been edited and may contain errors
  - Can be symbolized
- Errors are found during validation
  - Errors have properties
    - What rule was violated
    - Which feature(s) created the error
- Your options:
  - Ignore the error
  - Mark as exception
  - Fix the error

Parcels overlap

Topology Graph and Elements

- Each topology has a graph of elements
  - Planar representation
- Use ITopologyGraph to
  - Edit participating topology features
  - Without breaking adjacency or topological relationships
  - Graph access to topo relationships between features



Topology Demo

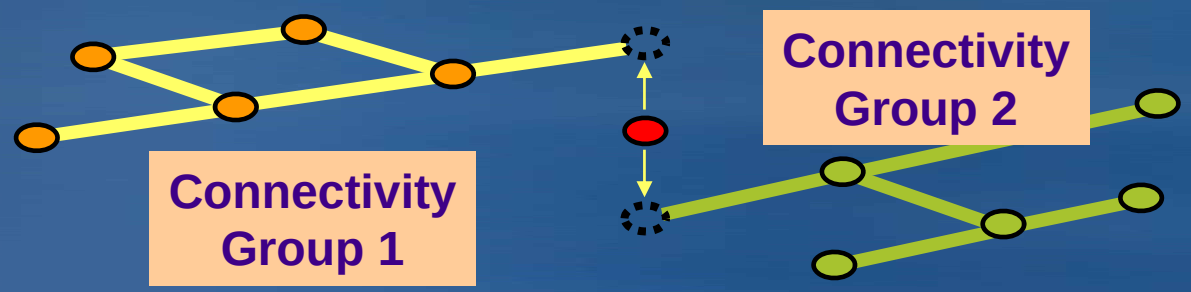
- Editing a shared edge with the Topology Graph

Network Datasets

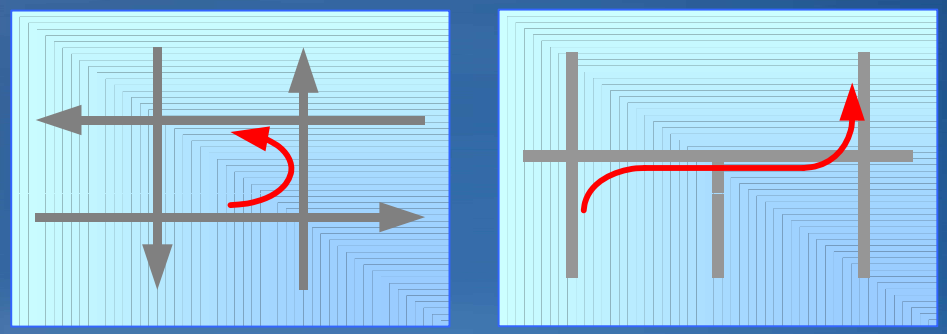
- Designed for Transportation industry
  - Does not replace the Geometric Network
- Works with the Topology work flow
  - dirty areas, validation
- Transportation specific functionality
  - Multimodal
  - Turns
  - Attributes
    - On-the-fly calculation of costs
    - Significant improvements for analysis

Network Datasets …

- Multimodal
  - Points span multiple connectivity groups
  - used to create connectivity between lines in different groups



- Turns
  - Turns do not alter connectivity, but traversability (e.g. U-Turn restriction)

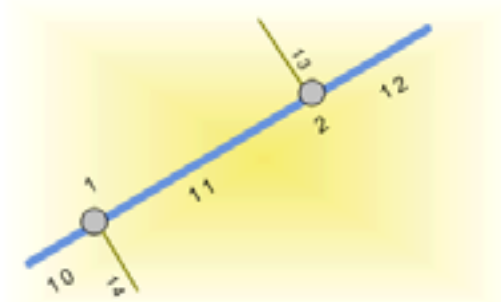


Network Datasets …

- Dataset Extension
  - Leverages Data Element and IDatasetContainer2 for creation and updates
- Different than geometric networks
  - Simple features
  - No custom behavior
  - Can be built on data sources other than the Geodatabase
    - Shapefiles
- Need to perform connectivity analysis
  - INetworkForwardStar
  - INetworkForwardStartAdjacencies
- Support Custom Evaluators (i.e. Cost, Restrictions) and Custom Solvers

Geometric Networks

- Used to model network systems
  - Primarly designed for Utilities\Natural Resources industries
- Connectivity relationships between feature classes.
  - Can associate connectivity rules with the network.
  - Connectivity is based on geometric coincidence, always live.
  - Live within a Feature Dataset
- Each feature class has a role in the network
  - A network may have multiple feature classes in the same role.



Water mains (Lines)

OID	Shape	Diameter	Material
10		8	Concrete
11		10	PVC
12		8	Concrete

Water services (Lines)

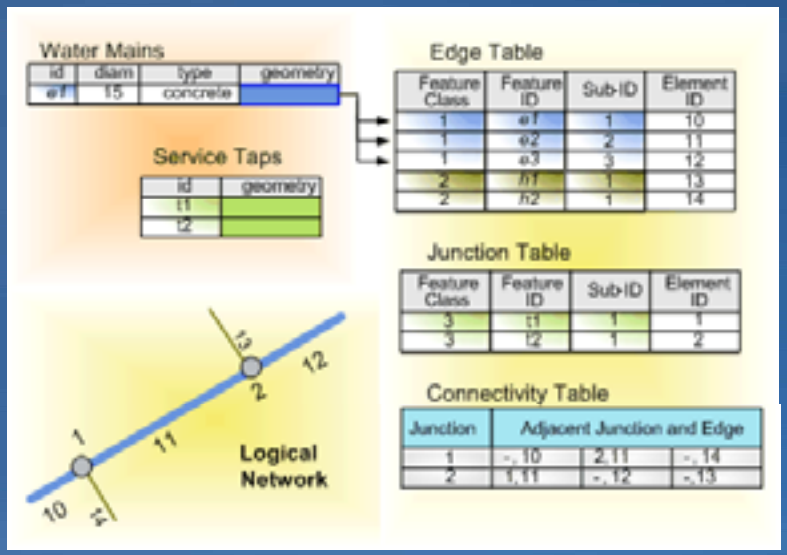
OID	Shape	Service ID	Material
13		1001	Cast iron
14		1002	Copper

Water junction fittings (Points)

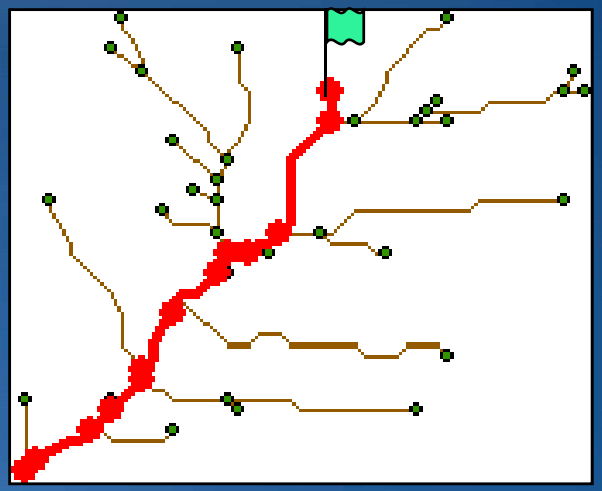
OID	Shape	Equip ID	Valve Type
1		816-32	T203
2		816-45	Y53

Geometric Networks …

- A geometric network is associated with a logical network.
  - Each network feature is associated with one or more elements in the logical network.
- Trace solvers on the logical network provide
  - Connectivity tracing, cycle detection, flow directions
  - Upstream/downstream tracing, Isolation tracing



Downstream Trace



Geometric Networks …

- Use INetworkLoader for creation of geometric networks
  - Specify the input parameters for the geometric network
  - Once all parameters are specified, use the LoadNetwork method to create the geometric network according to the specified parameters
- Parameters of note include:
  - Network name
  - Enabled and AncillaryRole field
  - Snapping and Snap tolerance
    - Uses the Tolerance for the Feature Dataset
  - Adding feature classes
    - Check if they are supported; INetworkLoader2::CanUseFeatureClass
  - Adding weights and weight associations
    - Fields must pre-exist
  - After building the network
    - Check for existence of build errors

Geometric Networks …

- Geometric Network features are classified as complex features
  - Do not support non versioned edits
  - Must be editing with an Edit Session and Edit Operation
- For creating new network features; same basic set of steps apply
- Geometric Network specific behavior is handled by the Geometric Network at creation time
  - Not required to call Connect
  - Not required create any logical network connectivity; ie: CreateNetworkElements method
  - Enabled and AncillaryRole values are set by the feature
- Not required to call Disconnect and Connect with spatial updates to features; geometric network will ensure integrity
  - Unless, you want to edit the feature geometry without impacting connected features

Geometric Networks …

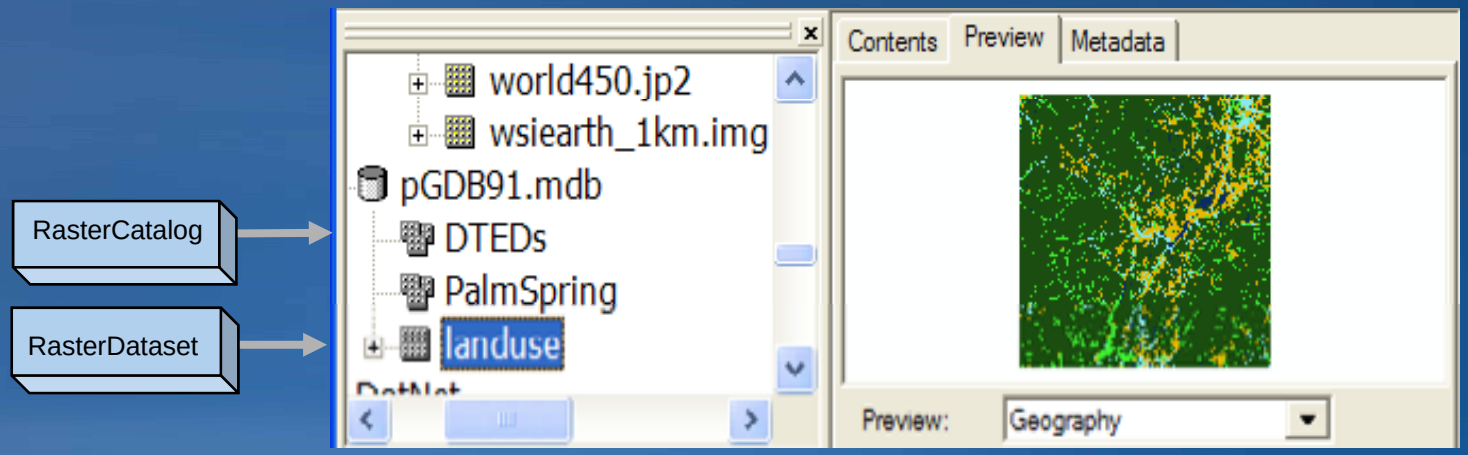
- Use logical network API for navigation and tracing whenever possible
  - IForwardStar
- Navigational APIs available at the geometric network feature level
  - Very slow
  - Use only for small tactical navigation
- Analysis algorithms (e.g., solvers) should always consume the logical network APIs
  - Orders of magnitude faster
  - INetwork
  - INetTopology

Raster Data

- Many file formats
  - GRID, TIFF, IMG, JPEG, JP2000, BMP, GIF, PNG, BIL/BIP/BSQ, PCI, ECW, USGS DEM, BSB, HDF4…
  - Can import these into the Geodatabase
- Geodatabase raster
  - ArcSDE geodatabase
  - Personal geodatabase
  - File geodatabase
- Services
  - WCS service
  - ArcGIS Server Image Service

The ArcGIS Raster Data Model

- Raster datasets
  - Separate rasters (raster dataset)
    - May mosaic during loading
- Raster catalogs
  - A collection of raster datasets
  - Two Types
    - Managed
    - Unmanaged

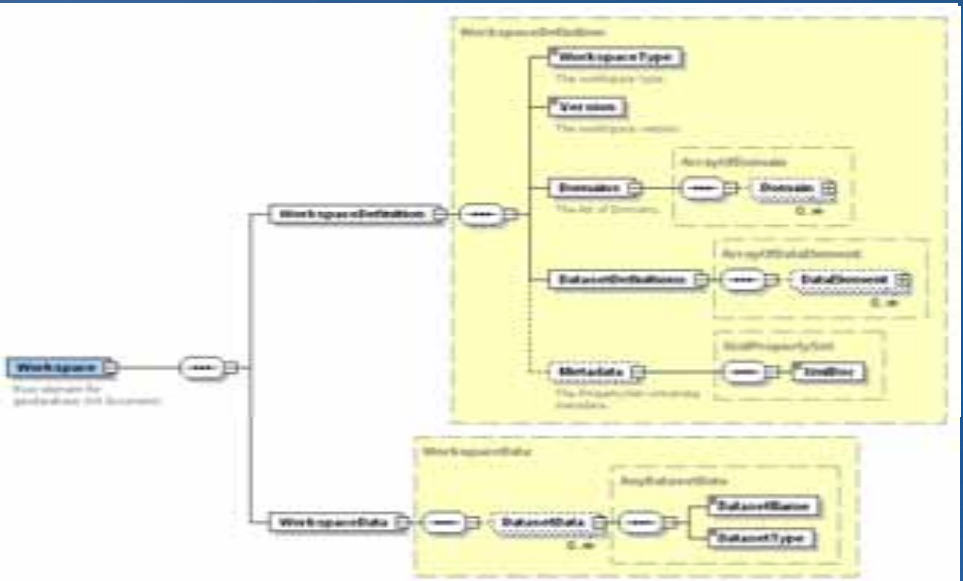


Geodatabase XML

- Why XML?
  - Platform / language independence
  - Open (can parse easily)
  - Documented Format (Xml Schema)
    - Google “Xml Geodatabase Schema”
- Binary vs. Normalized
- Validation
- What you can export?
  - Geodatabase (Workspace)
  - Feature Class (RecordSet)
  - Replica Data Changes (UpdateGram)

Workspace Document

- Geodatabase in XML
  - Schema (Full geodatabase schema description)
  - Feature class, Geometric Networks, FeatureDatasets, Metadata etc…
  - Workspace document can be schema-only
- Data
  - One recordset per feature class



XML Demo

- Reading Geodatabase XML workspace document demo

Presentation Outline

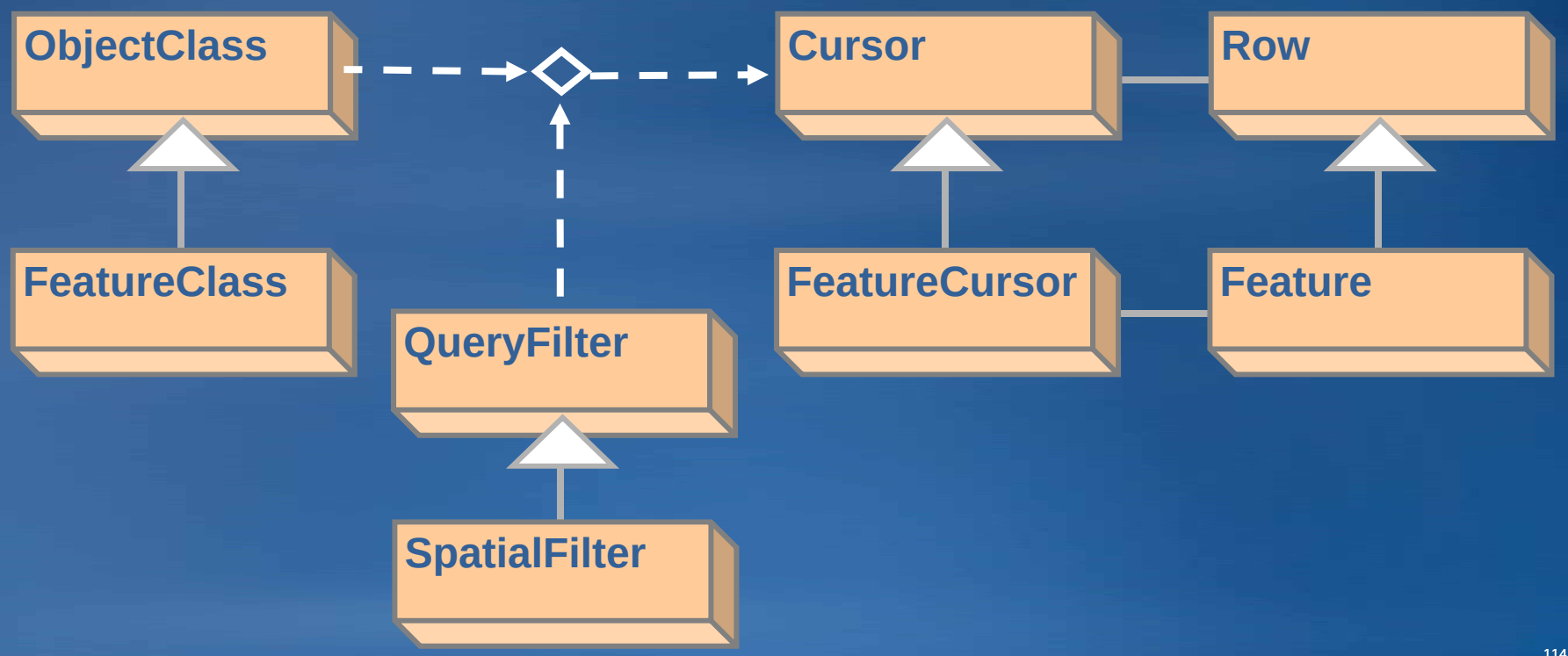
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Cursors

- A geodatabase object used for the iteration of records returned from a query
- 3 Class Cursors
  - Search (general query cursor)
  - Update (positioned update cursor)
  - Insert (bulk inserts)
- 1 QueryDef Cursor
  - 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 a QueryDef cursor are not bound to a class

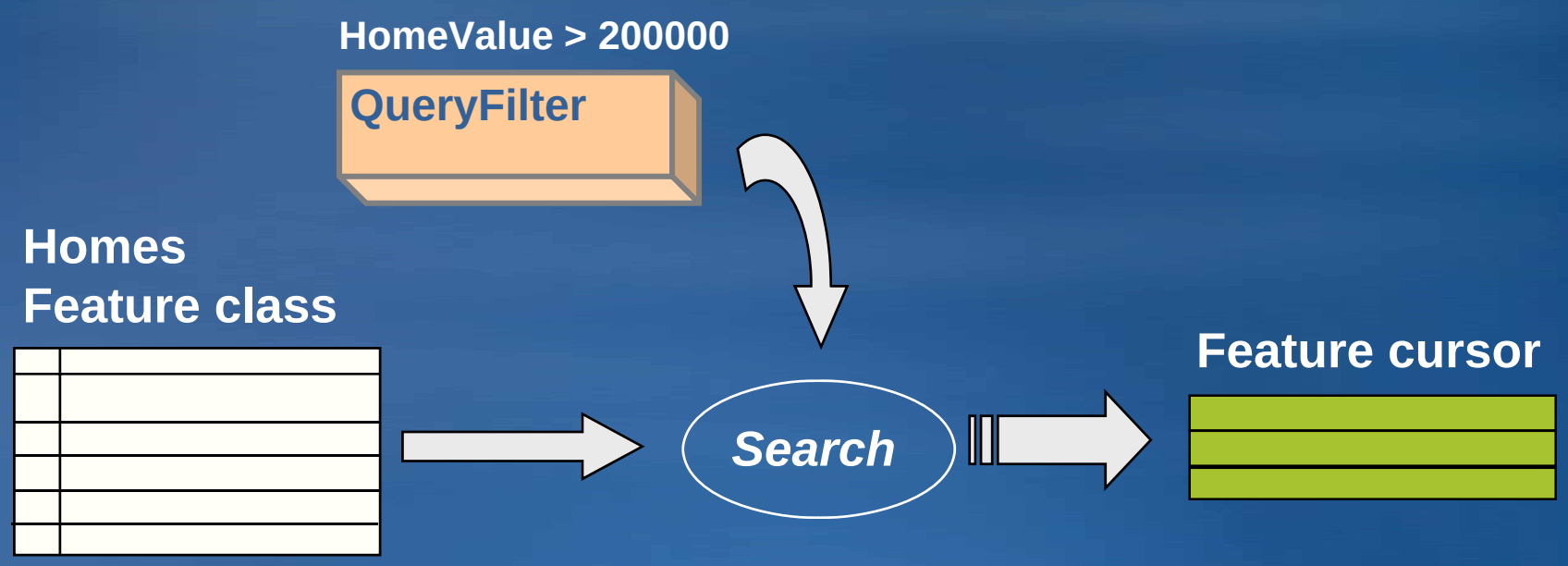
Class Cursors

- A table and a query return a cursor
- Used to:
  - Iterate over a set of rows in a table
  - Insert new rows into a table
- A cursor gives you access to one row at a time



Creating a Cursor

- QueryFilter contains an SQL-like statement
- The cursor contains a subset
  - No filter\nothing, all rows returned



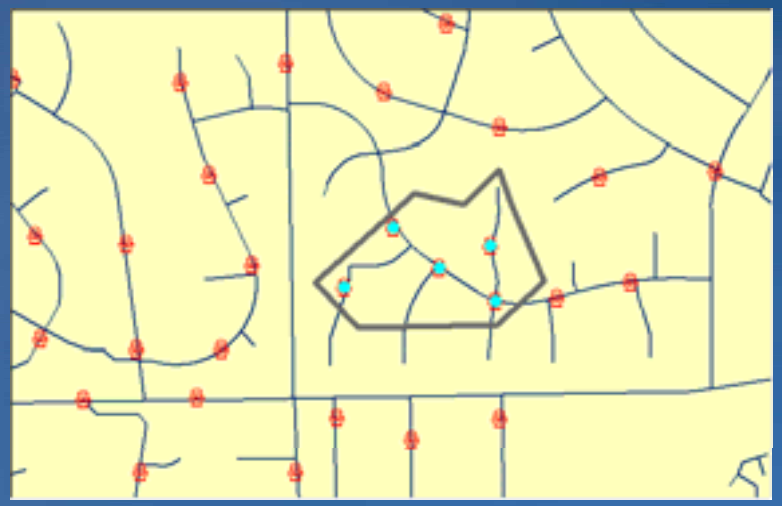
IQueryFilter

```
IQueryFilter queryFilter = new QueryFilterClass();
queryFilter.SubFields = "OBJECTID,FULLNAME,ParcelID";
queryFilter.WhereClause = “FULLNAME like ‘D%’";

IQueryFilterDefinition queryFilterDef =
(IQueryFilterDefinition)queryFilter;
queryFilterDef.PostFixClause = "ORDER BY FULLNAME";

IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
```

- **IQueryFilterDefinition::PostFixClause**
  - Supports functions such as ORDER BY

Creating a Cursor …

- SpatialFilter need a geometry and relationship
- Below the geometry is a polygon
- Below the spatial relationship is *contains*



Contains

Crosses

Intersects

Overlaps

Touches

Within

ISpatialFilter

- Used to query spatial aspects of a feature class
  - Inherits from IQueryFilter

```
ISpatialFilter spatialFilter = new spatialFilterClass();

spatialFilter.SubFields = “OBJECTID,FULLNAME,ParcelID,SHAPE”;
spatialFilter.Geometry = envelope;
spatialFilter.SpatialRel = within;
spatialFilter.WhereClause = “FULLNAME like ‘D%’”;

IFeatureCursor featureCursor = featureClass.Search(spatialFilter, true);
```

Types of Class Cursors

- Search cursors
  - Returns rows specified by a Query or Spatial Filter
- Update cursors
  - Update and delete rows specified by the filter
  - Specify the ObjectID field
- Insert cursors
  - Used for inserting rows into a table
- Accessed by
  - Corresponding methods on table or feature class

Types of Class Cursors …

- Forward only, do not support
  - Backing up and retrieving rows already retrieved
  - Making multiple passes
  - Resetting
- Solution:
  - Re-execute the query
- Release Class Cursors with
  - Marshal.ReleaseComObject
  - Cleaner.release()

Types of Class Cursors …

- Insert cursors are used to bulk insert rows
  - Faster for loading simple data than IFeature.Store
    - Bypasses events
    - IObjectClassInfo2 and IWorkspaceEditControl to override
  - Not Faster for non-simple data
    - Behavior, composite relationships, and notification
    - Need CreateRow and Store methods, so no performance gain
  - Use of Buffering is key
    - Pre-define attribute values
    - Buffers inserts on client, sends to database on Flush
- Flush – Call or not
  - Interval flushing: Check for room or handle errors
  - Careful: Insert cursors flush on destruction
    - No chance to detect errors

Types of Class Cursors …

- Scope cursors to edit operations
- Cursor is bound to a specific state of the geodatabase
- When state of the geodatabase changes cursor is no longer valid and should not be used
  - Performing edits on a cursor that is incorrectly scoped can cause data corruption and other unexpected behavior.

Recycling Method

- Recycling
  - A recycling cursor is a cursor that does not create a new client side row object for each row retrieved from the database
  - Allocate a single row object
    - Re-hydrate on each fetch
  - Performance advantages
  - Primarily used for reading data
- Non Recycling
  - A different row object on each fetch
  - Always has full set of fields, even if IQueryFilter::Subfields used

```
pCursor = theMeds.Update(pFilter,false)
```

Cursors - Efficient Use of FindField

- FindField is the API used to get a value index
- The Fields collection of a cursor created by a class is identical to the Fields collection of the class regardless of the SubFields specified in a QueryFilter
  - Index of the field is consistent
  - Not true for cursors created by IQueryDef.Evaluate
- Call FindField on the coarsest grain object with the matching Fields collection (i.e. class or cursor)
  - Avoid calls to FindField in a loop
  - use of Fields collection of a row and calling FindField is rare

Cursors

Example: Efficient Use of FindField

```
public void EfficientExample(IFeatureWorkspace featureWorkspace)
{
  ITable testTable = featureWorkspace.OpenTable("Parcels");
  int parcelIdIndex = testTable.FindField("PARCEL_ID");
  int parcelKeyIndex = testTable.FindField("PARCEL_KEY");

  ICursor cursor1 = testTable.Search(null, true);
  IRow row1 = null;
  while ((row1 = cursor1.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row1.get_Value(parcelIdIndex));
    Console.WriteLine("PARCEL_KEY = {0}", row1.get_Value(parcelKeyIndex));
  }

  ICursor cursor2 = testTable.Search(null, true);
  IRow row2 = null;
  while ((row2 = cursor2.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row2.get_Value(parcelIdIndex));
    Console.WriteLine("PARCEL_KEY = {0}", row2.get_Value(parcelKeyIndex));
  }
}
```

Cursors

Example: Inefficient Use of FindField

```
public void InefficientExample(IFeatureWorkspace featureWorkspace)
{
  ITable testTable = featureWorkspace.OpenTable("Parcels");

  ICursor cursor1 = testTable.Search(null, true);
  IRow row1 = null;
  while ((row1 = cursor1.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row1.get_Value(testTable.FindField("PARCEL_ID")));
    Console.WriteLine("PARCEL_KEY = {0}", row1.get_Value(testTable.FindField("PARCEL_KEY")));
  }

  ICursor cursor2 = testTable.Search(null, true);
  IRow row2 = null;
  while ((row2 = cursor2.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row2.get_Value(testTable.FindField("PARCEL_ID")));
    Console.WriteLine("PARCEL_KEY = {0}", row2.get_Value(testTable.FindField("PARCEL_KEY")));
  }
}
```

QueryDef Cursors

- Query based on one or more tables
  - Analogous to an SQL Query
  - Get a cursor back
  - Not bound to one class
- Tables must be in database
  - Do not have to be registered with the geodatabase
- Can result in a feature layer
  - Need to use IQueryName2 if no ObjectIDs in input tables
- Are also used to establish joins

QueryDef Cursors (IQueryDef)

- Simple QueryDef between 2 tables

```
//Create the query definition
IQueryDef queryDef = featureWorkspace.CreateQueryDef();

//Provide a list of tables to join
queryDef.Tables = "PoleFeature, TransformerFeature";

//Retrieve the fields from all tables
queryDef.SubFields = "bob.PoleFeature.TAG, bob.PoleFeature.SHAPE,
bob.TransformerFeature.Tag_Val";

//Set up the join based on the owner_name attribute
queryDef.WhereClause = "PoleFeature.TAG = TransformerFeature.Tag_Val";

ICursor cursor = queryDef.Evaluate();
IRow row = cursor.NextRow();
```

GetFeature vs GetFeatures

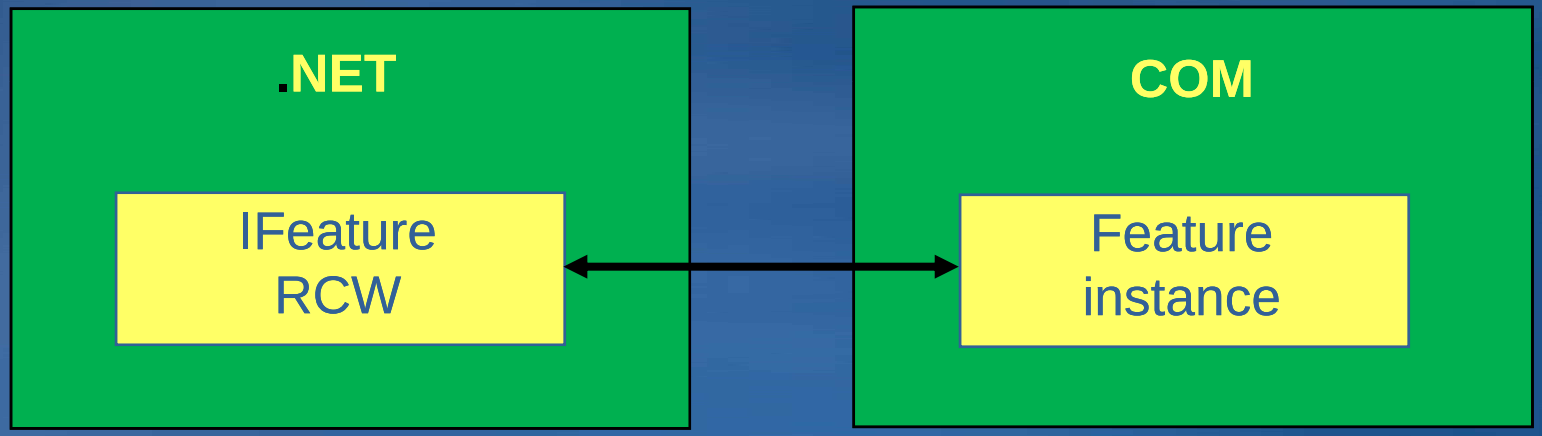
- Two similar methods for getting features with ObjectIDs
  - GetFeature returns a single feature based on an ObjectID
  - GetFeatures returns a cursor with features specified in an integer array parameter
  - Methods are available on IGeodatabaseBridge for use in .Net and Java
- Use GetFeatures any time more than one feature is being retrieved using a known Object ID
  - Avoid looping over GetFeature
- GetFeatures outperforms GetFeature example on as few as two features
  - Difference will grow as more features are requested
  - With 1000 features GetFeature will often take up to 20 times as long

Cursor demo

- Cursors examples
  - Search
  - Update
  - Insert

COM objects in .NET

- COM objects are accessible through a thin .NET object - runtime-callable wrapper (RCW)
- True for all COM objects, but particularly relevant for Geodatabase API developers



Why is this important?

- Some geodatabase objects maintain locks on files or DBMS resources
- To release these resources, the COM object's reference count must = 0
- RCW clean-up relies on the .NET garbage collector
  - Non-deterministic
- To deterministically release resources, COM object must be explicitly released from RCW

Marshal.ReleaseComObject

- Decrements a COM object's reference count by 1
- Detaches the RCW from the COM object
  - Must be re-fetched to use again
- Some workflows require try/catch/finally blocks

```
ICursor cursor = null;
try
{
  cursor = featureClass.Search(null, true);
  // Do something with the cursor...
}
catch (Exception)
{
  // Throw new exception, return value, etc...
}
finally
{
  if (cursor != null)
    Marshal.ReleaseComObject(cursor)
}
```

ComReleaser class

- In ESRI.ArcGIS.ADF assembly
- Implements IDisposable
  - Using statement guarantees managed objects are disposed
- Decrements COM object's refcount to 0
- May not work well for situations with user interaction
  - Depends how objects are scoped

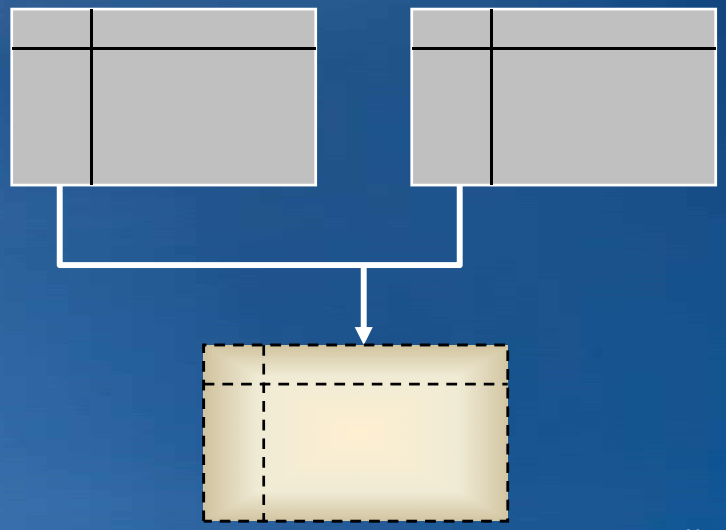
```
using (ComReleaser releaser = new ComReleaser())
{
  ICursor cursor = featureClass.Search(null, true);
  releaser.ManageLifetime(cursor);

  // Do something with the cursor...
}
```

Which objects should be released?

- Technically, all COM objects
  - Unrealistic
- Cursors
  - Always
- Workspaces, versions, datasets, rows
  - Some workflows require these as well

Views and Table Joins

- Joining data involves appending the fields from one or more tables to another table
  - Method used depends on the data sources as well as the cardinality of the data.
- Persisted
  - ArcSDE Views (MultiVersion Views)
- On-The-Fly
  - Queries
    - QueryDef
  - Tables
    - QueryTables
    - RelQueryTables



Persisted Views

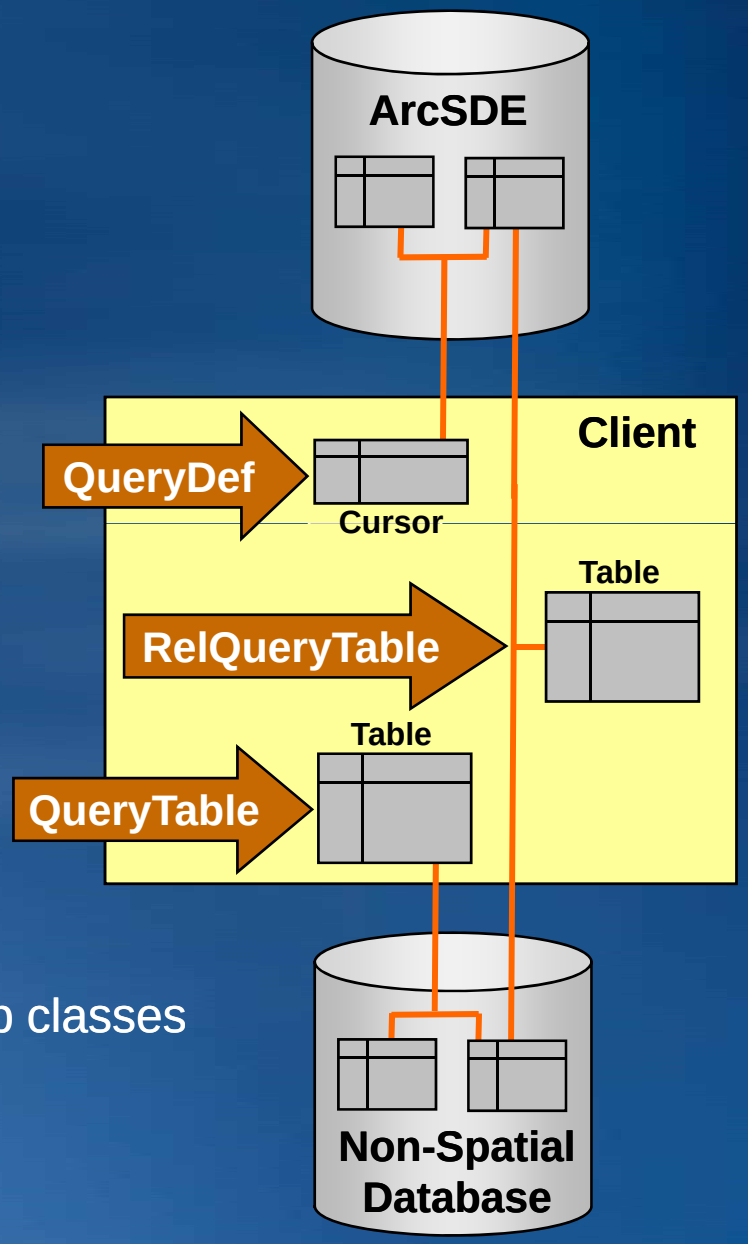
- Open views as tables (read only)

```
//open view through API as (read only) table
ITable myJoinedTable = fWorkspace.OpenTable(“databaseView");
```

- Must satisfy geodatabase rules for a valid table or feature class
  - Only one spatial column
  - Supported data types in returned fields

On-The-Fly Joins

- QueryTables (**`ITableQueryName`**)
  - Tables must be within same datasource
  - Matches all candidates
  - Uses QueryDef object
  - Can be used with non-spatial tables
- RelQueryTables (**`IRelQueryTable`**)
  - Tables can be in different datasources
  - Matches only first candidate on 1:M join
  - Uses in-memory or geodatabase relationship classes



Example: QueryTables (ITableQueryName)

- Steps to create a join via ITableQueryName
  1. Create a new TableQueryName object
  2. Set the QueryDef and PrimaryKey property
  3. Cast to IDatasetName setting WorkspaceName and Name
  4. Open the name object as a table

```
// Make the new TableQueryName
IQueryName2 qn2 = (IQueryName2)new TableQueryName();
qn2.QueryDef = qdef;
qn2.PrimaryKey = "ObjectID";
qn2.CopyLocally = false;

// Set the workspace and name of the new QueryTable
IDatasetName pDSName = (IDatasetName)qn2;
pDSName.WorkspaceName = WSName;
pDSName.Name = TableName;

// Open and return the table
IName name = (IName)qn2;
ITable table = (ITable)name.Open();
```

Example: RelQueryTables (`IRelQueryTable`)

- Steps to create a join via `IRelQueryTable`
  1. Create a new memory `RelationshipClass`
     - Passing in the `ObjectClasses` that need to be joined
     - An existing relationship class can be used
  2. Call open on `RelQueryTableFactory` casting to `ITable`
     - Passing in the relationship class

```
// build a memoryrelationshipclass
IMemoryRelationshipClassFactory mRCfactory = new
MemoryRelationshipClassFactoryClass();

// open the memoryrelationshipclass
IRelationshipClass memRC = mRCfactory.Open("memrc", targetObjectClass,
fromField, joinObjectClass, toField, "forward", "backward",
esriRelCardinality.esriRelCardinalityOneToOne);

// Open the relquerytable as a table
IRelQueryTableFactory rqtfactory = new RelQueryTableFactoryClass();

ITable rqTable = (ITable)rqtfactory.Open(memRC, true, null, null, "",
false, true);
```

Join Demo

- Need to join a feature class and a DBF file
  - To answer a question with information from both tables
  - Who are the owners for certain parcels?

ParcelsOwners

Parcels

OID	Property_I	Area
1	1001	120000
2	1002	305000
3	1003	58000

Owners.DBF

OID	PROPERTY_I	OWNER_NAME
1	1001	DOE JOHN
10	1002	DOE JANE
23	1003	CITY

Presentation Outline

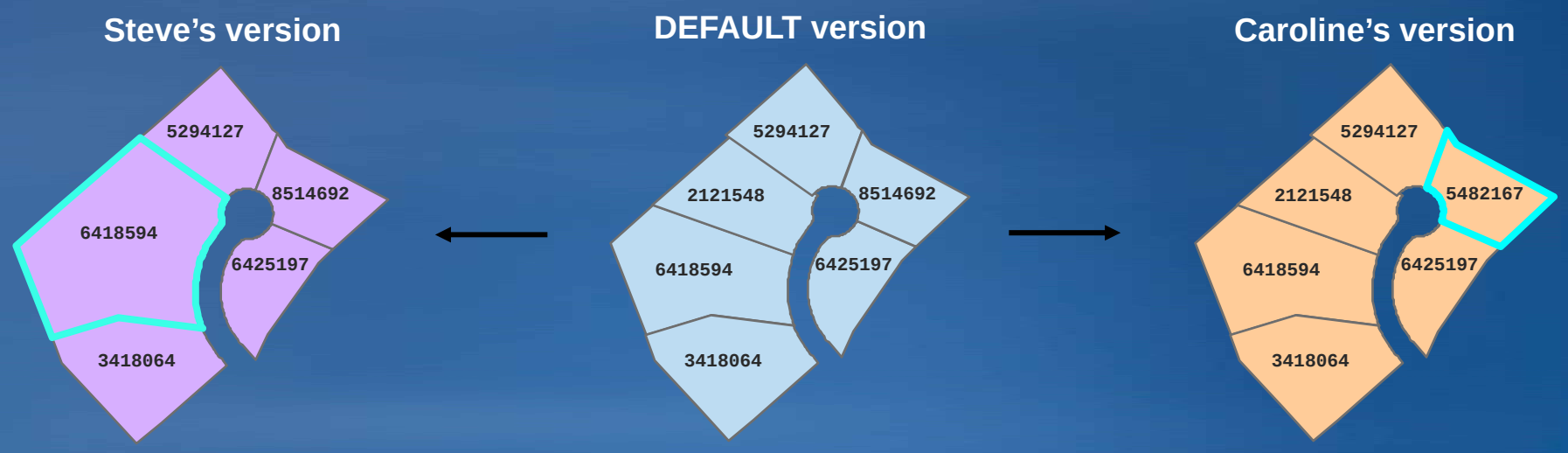
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Geodatabase Transaction User Requirements

- Edit sessions
  - Support concurrent editing with long transactions (hours/days).
  - Undo/redo editing experience.
  - No locking or data extraction required.
- Persistent design alternatives
  - Multiple, parallel states in the database
  - Difference detection and reconciliation
  - Manage referential integrity and role-based behavior on parallel versions

What is Versioning?

- Our solution to allow multi-user editing of geographic data
  - A version is just a state in the database.
- A method for presenting and tracking changes
  - Changes accessed through a version
  - Changes preserved in delta tables
- Includes mechanisms for reconciling versions
  - Provides tools to resolve conflicts



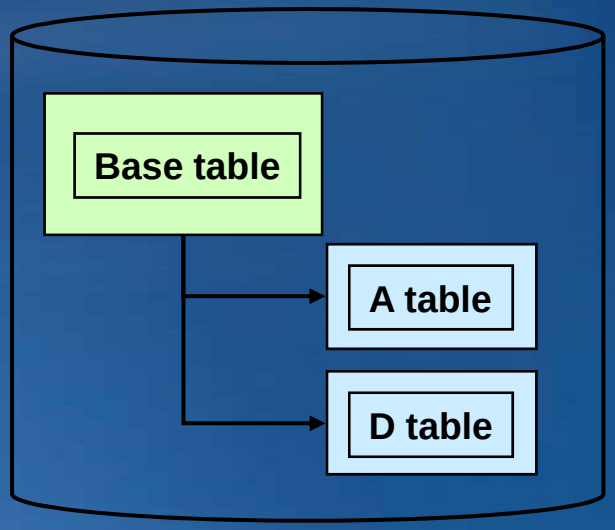
Versioned editing

- Advantages include:
  - Isolate editor's work over an extended period
  - Reconcile and resolve conflicts between edits
  - Implement workflow for business procedures
  - Perform geodatabase archiving and replication
  - Features not locked during editing

Versioned editing

- Advantages include:
  - Isolate editor's work over an extended period
  - Reconcile and resolve conflicts between edits
  - Implement workflow for business procedures
  - Perform geodatabase archiving and replication
  - Features not locked during editing
- Architecture:
  - Change to each feature class is preserved in A and D tables
  - Change accessed through a version

Versioned editing:
Changes stored in A and D tables

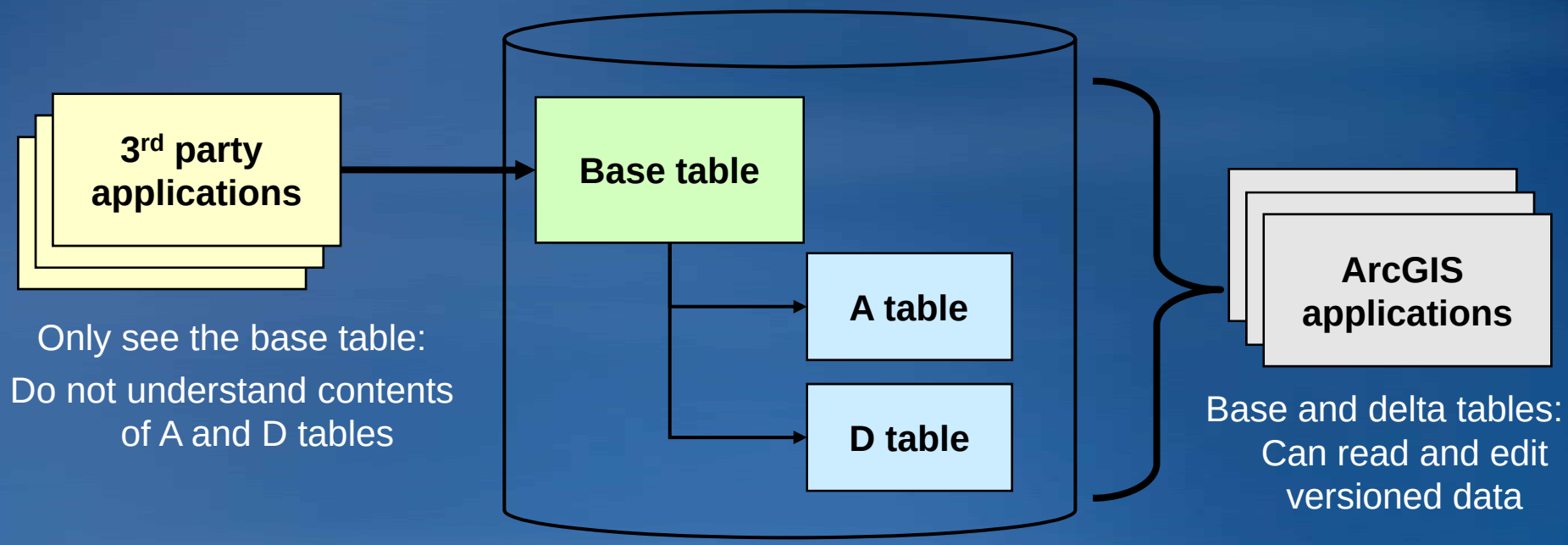


Versioned editing

- Once an editors work is complete
- Changes can be integrated into other versions
  - Through a mechanism called Reconcile and Post
  - Compares changes in your edited version with the version into which you want to merge the edits
- Identifies any features edited within both versions as a Conflict
- IVersionEdit::Reconcile4 (VersionName, acquireLock, abortIfConflicts, ChildWins, ColumnLevel)

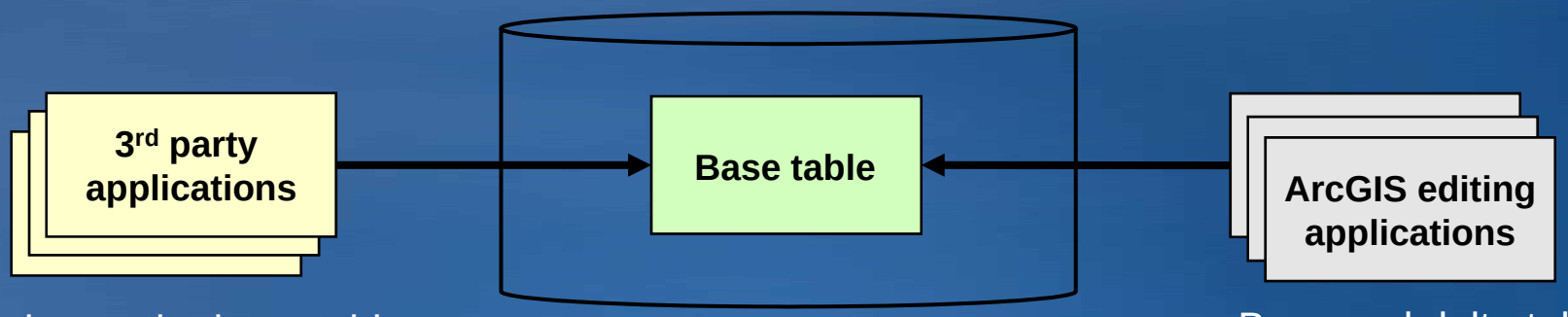
Versioned editing

- Does not easily support:
  - Non-ESRI client access to versioned data
    - Edits are preserved in delta tables
    - Cannot see edits in the base table
  - DBMS behavior such as triggers and constraints



Non-versioned editing

- Easy to implement
- No A and D tables
  - Edits immediately saved to base tables
- DBMS behavior is easy to implement
  - Uses the underlying database transaction model
- Easy IT integration
  - Non-ESRI applications see edits in base tables



Only see the base table:
Do not understand contents of A and D tables

Base and delta tables:
Can read and edit versioned data

Comparison of Editing Models

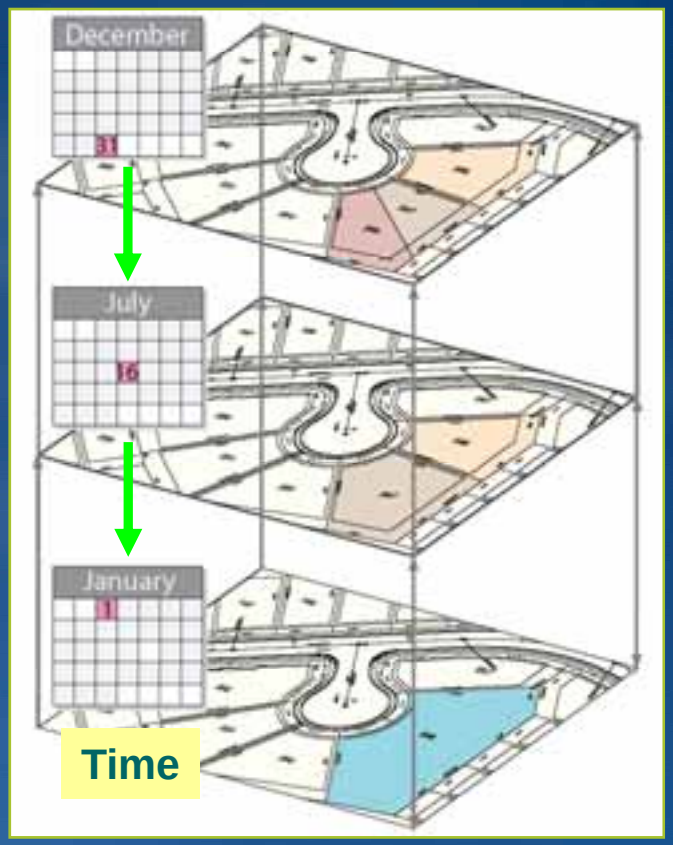
- Non-versioned editing provides
  - Direct editing using database transactions
  - Immediate visibility of edited information upon saving
  - Supported only on simple features and tables
    - points, lines, polygons, annotation, relationships
- Versioned editing provides
  - Long transactions, design versions, proposals
  - Geodatabase archiving
  - Geodatabase replication
  - Supported on all Geodatabase feature types and datasets

Comparison of Editing Models …

- IWorkspaceEdit vs IMultiUserWorkspaceEdit
  - IWorkspaceEdit
    - Local geodatabase data (Personal, File) and ArcSDE versioned data
  - IMultiUserWorkspaceEdit
    - Leveraged by non-versioned editing
    - ArcSDE data only
    - Can be used on both versioned and non-versioned data

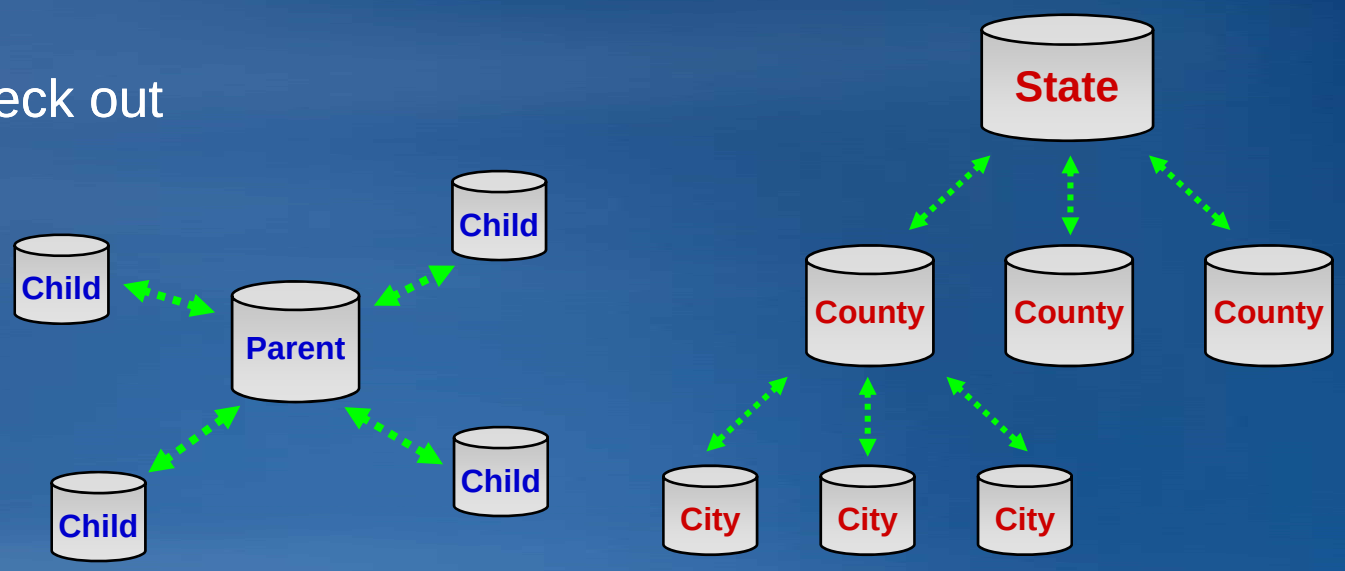
Geodatabase Archiving

- Ability to maintain & query historical database states
- All changes made to the DEFAULT version are archived
  - Stores transaction time, not valid time
- Edits stored in separate archive table
- Archives can be queried based on date information

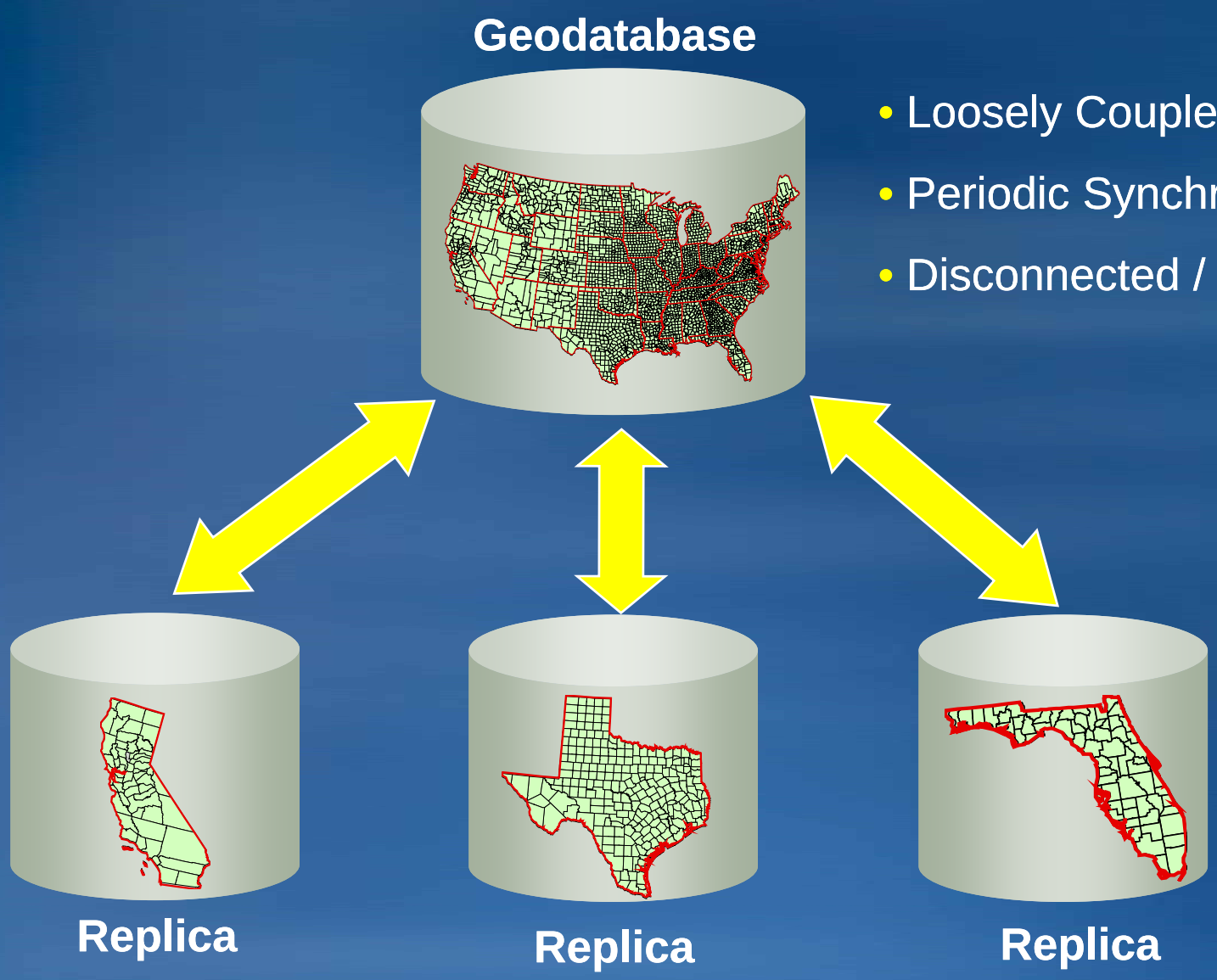


Geodatabase Replication

- Allows you to distribute copies of data across 2 or more Geodatabases
- You can edit the databases independently and synchronize them as needed.
- Three types:
  - Check in\Check out
  - One-Way
  - Two-Way



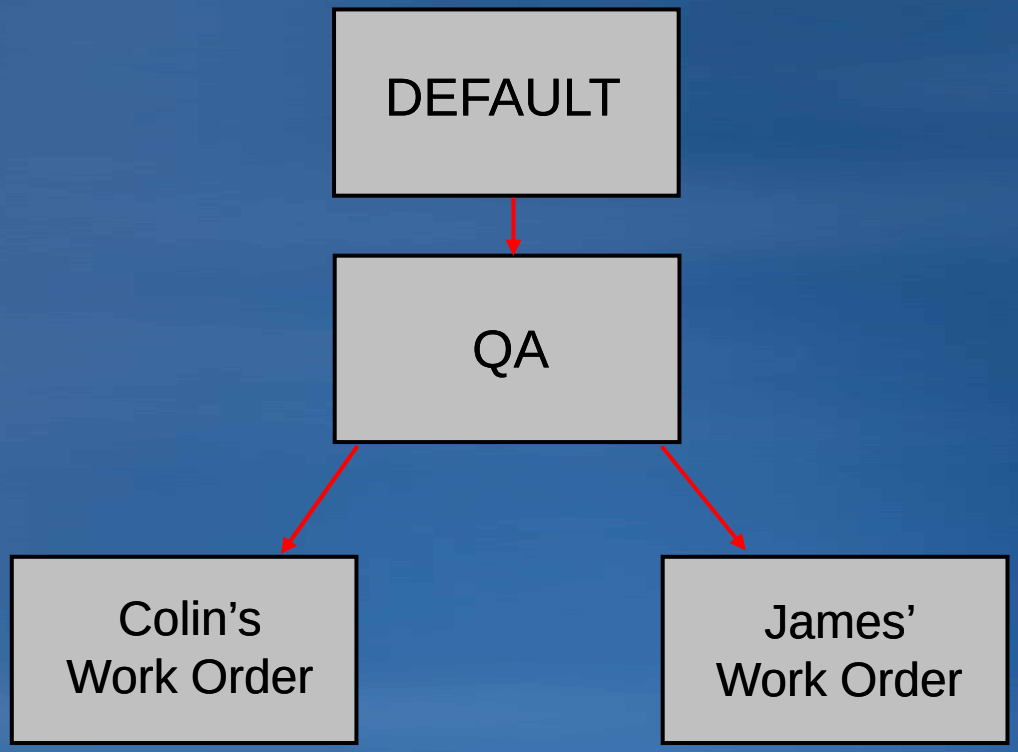
Multi Generation Replication



- Loosely Coupled
- Periodic Synchronization
- Disconnected / Connected

Versioning Demo

- The QA version has two children, EditorA and EditorB
- EditorB has already edited a feature and posted the changes to the QA version
- EditorA's will attempt to reconcile and post



Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Conversion and Loading

- Loading into the geodatabase
  - IFeatureDataConverter
    - Used for converting simple features only
    - A lot of set-up required
    - Provides many options and a lot of control
  - Geoprocessing tools
    - Course grained
    - Less options available
    - Facilitates loading

Conversion and Loading

- Between Geodatabases
  - IGeoDBDataTransfer
    - Supports simple and non-simple features
    - Works at Dataset level, not the workspace level
  - IGdbXmlImport and IGdbXmlExport
    - Supports simple and non-simple features
    - Workspace\Dataset level
  - Can also use methods outlined in previous slide
    - Not as optimal as methods outlined above

Conversion and Loading

- IFeatureClassLoad::LoadOnlyMode
  - ArcSDE and File Geodatabases
- For a feature class or table in load-only mode:
  - Disable updating of spatial and attribute indexes (File Geodatabase only)
- Taking the feature class or table out of load-only mode rebuilds indexes
- Keep your use of LoadOnlyMode tight!
  - While in load-only mode, other applications cannot work with the data
  - Acquire an exclusive schema lock on the feature class prior to putting it into Load Only Mode

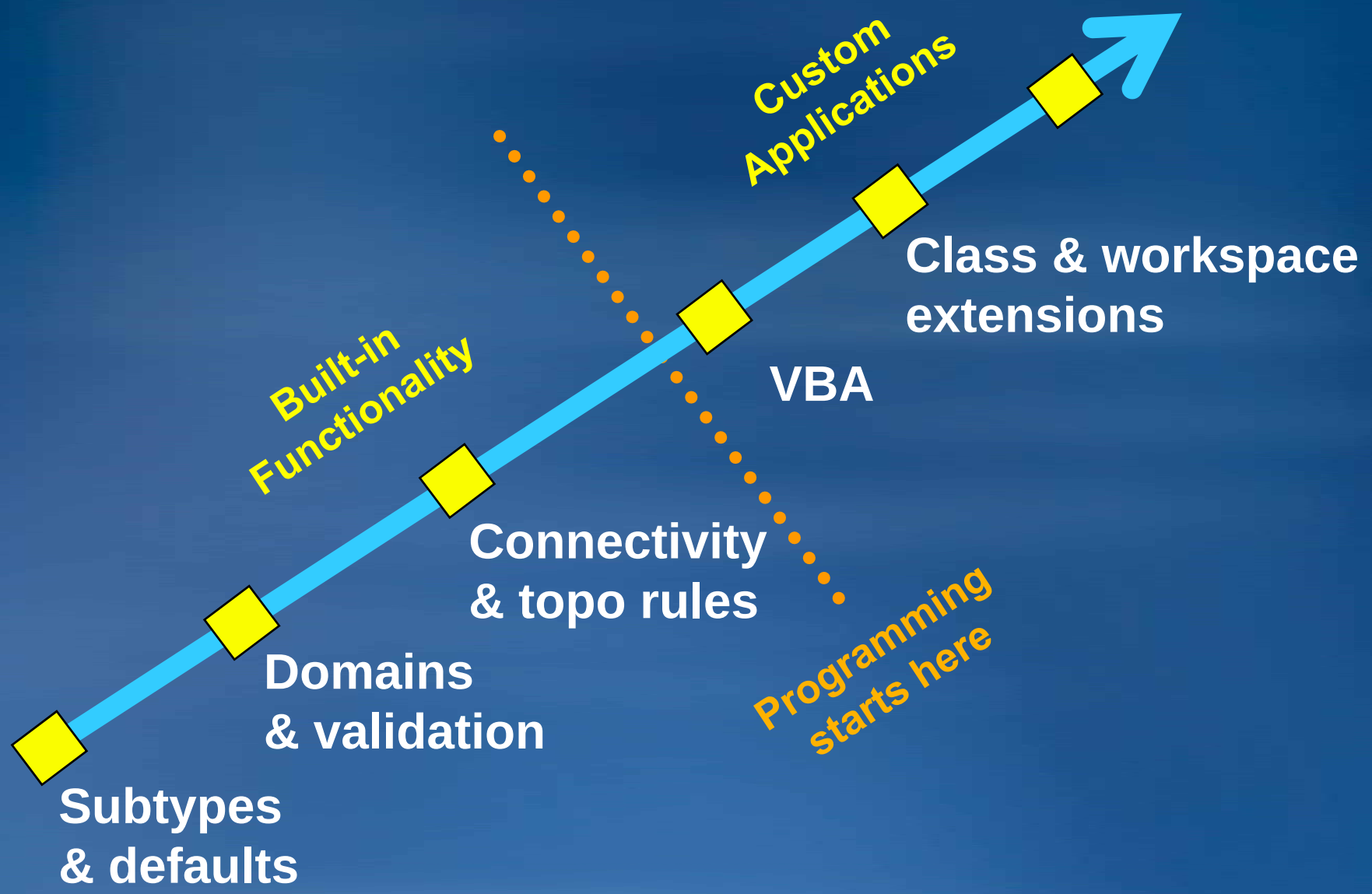
Data Loading demo

- Data Loading

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Levels of Customization



Level of Customization

- Application level
  - Pros
    - Business logic is stored within application
    - Can access data without customization
  - Cons
    - Only available when application is running
    - Users do not always interact with application customization

Level of Customization

- Database level
  - Class / Workspace extensions
  - Pros
    - Business Logic is stored with data
    - Always available, regardless of application
  - Cons
    - One class extension per feature class
    - All users require dll to even view data
    - Database is unusable if code fails
    - Impacts on performance
- Use for important business rules that can be simply implemented without serious performance considerations.

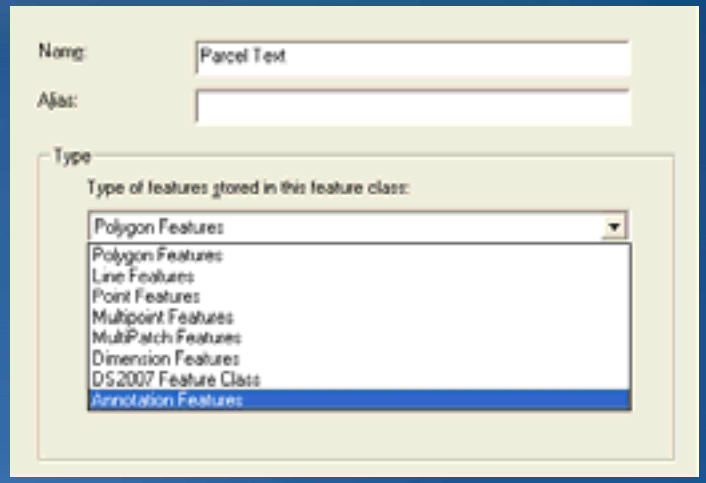
Level of Customization

- Custom features
  - Pros
    - Provide near total control over functionality.
  - Cons
    - Performance can suffer, since code is executed redundantly for every feature.
    - Handling of row and relationship events is less stable than class extensions.
    - Technically challenging to implement.
    - Only supported in C++
- Problem can often be solved by class extension or application customization

Examples of Customization

- Schema generation
- Custom drawing
- Custom property inspection and validation
- Custom split policies
- Related object creation notification
- PlugIn Data Sources
- Workspace logs

Your description here
(In ArcCatalog)



Customization demo

- “Time Stamper” demo

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

Common Geodatabase API Programming Mistakes

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

- Recycling and Cursors
- Overuse of FindField
- Scoping cursors to edit operations
- Performing DDL inside of edit sessions
- Calling Store inside of Store-triggered events
- GetFeature and GetFeatures
- Careless reuse of variables
- Inserts and relationship class notification
- Modifying schema objects

Other recommended sessions…

- Geometric Networks for Developers (Demo Theatre)
  - Wednesday, 12:00pm, Oasis 1
- Effective Geodatabase Programming
  - Tuesday, 1:00pm – 2:15pm, Pasadena/Ventura/Sierra
- Developing with ArcGIS Raster APIs
  - Wednesday, 1:00pm – 2:15pm, Smoketree A - E
- Distributed Geodatabase Development
  - Wednesday, 2:45pm – 4:00pm, Smoketree A - E
- Implementing Enterprise Applications with the Geodatabase
  - Wednesday, 4:30pm – 5:45pm, Smoketree A - E
- Working Effectively with the Geodatabase Using SQL
  - Thursday, 8:30am – 9:45am, Primrose C/D

Additional Resources

Questions, answers and information…

- ***Tech Talk***
  - ***Outside this room right now!***
- ***Meet the Team***
  - ***Wednesday at 10:30 am***
- ***ESRI Resource Centers***
  - **PPTs, code and video**

resources.esri.com

- ***Social Networking***

www.twitter.com/ESRIDevSummit

tinyurl.com/ESRIDevSummitFB

Want to Learn More?

ESRI Training and Education Resources

- **Instructor-Led Training**
  - **Data Management in the Multiuser Geodatabase**

http://www.esri.com/training