



**Welcome to:**

# **Developers Guide to the Geodatabase (Best Practices)**

**Craig Gillgrass**

**Colin Zwicker**



**Please!**

turn OFF cell phones  
and refrain from using  
flash photography

# Session

- **Session focus on:**
  - **How to work with the Geodatabase API**
    - **As opposed to customizing the geodatabase**
  - **Tip\Tricks for using the API**
- **Lots of content, little time**
  - **Please hold all questions to the end**
  - **We'll be at the Tech Talk for questions**
  - **Also Island area today and tomorrow**

# Assumptions

- **Geodatabase knowledge**
  - Basic understanding of geodatabase concepts and terms
    - We'll have a brief overview
  - Programmer/developer or user side
- **Basic programming skills**
  - C# demos
  - Code will be available with slides
  - Ability to read OMDs
  - ArcObjects supports: VB 6, .Net, Java, etc

# Presentation Outline

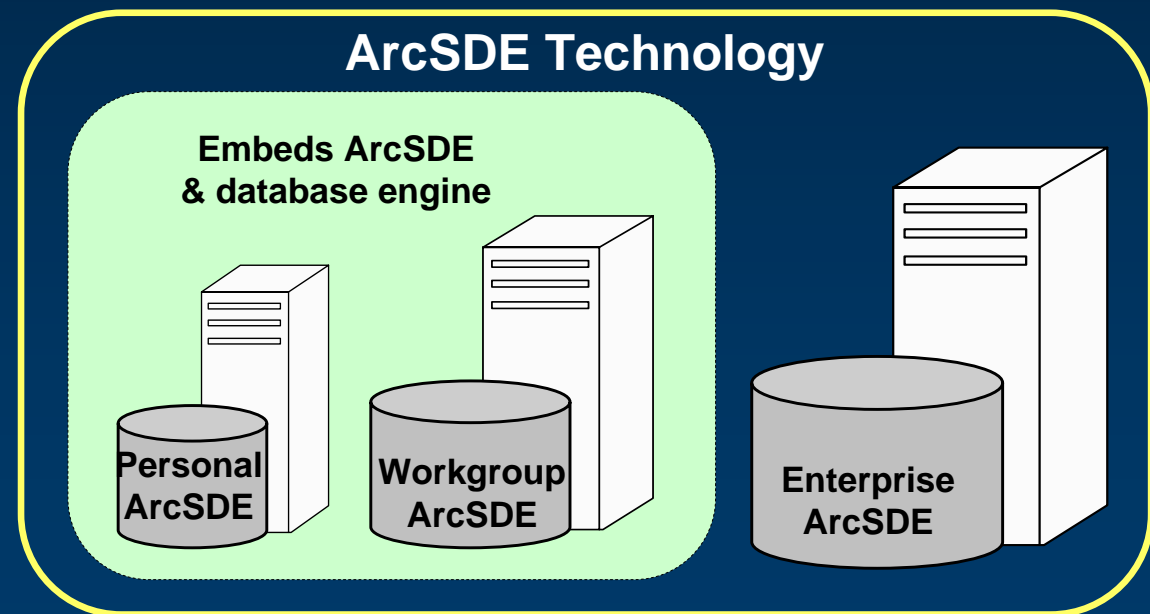
- **Introduction to Geodatabase Programming**
- **Licensing**
- **Accessing and Creating Data**
- **Beyond Basics And Editing**
- **Cursors**
- **Conversions and Loading**
- **Extending the Geodatabase**
- **Questions at End**

# Geodatabase

- **Geodatabase is...**
  - Core ArcGIS data model
  - Set of components in ArcGIS for accessing data
  - A physical store of geographic data
- **Built on Relational DBMS**
- **Geodatabase is a workspace**
  - Contains datasets; feature classes, tables, etc
  - Extend with topologies, geometric networks, terrains, etc
- **Geodatabase API**
  - Building blocks for programming with GIS data
    - Not just the native geodatabase model
  - Leverage the API against other datasets
    - Shapefiles, CAD, Coverages, etc

# Geodatabase ...

- **Geodatabase types at ArcGIS 9.2**
- **Geodatabases can be stored different ways**
  - Personal geodatabase (Access mdb file)
  - File geodatabase (new at 9.2)
    - Directory of binary files
  - Personal and workgroup geodatabase (new at 9.2)
    - SQL Server Express
  - Enterprise geodatabase
    - 4 supported DBMSs



Increasing size and/or functionality

# Presentation Outline

- Introduction to Geodatabase Programming
- **Licensing**
- Accessing and Creating Data
- Beyond Basics And Editing
- Cursors
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

# Licensing

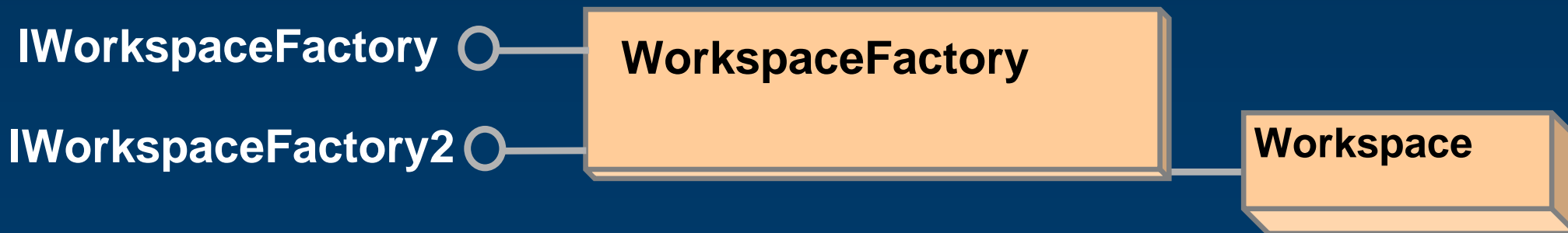
- **Required to initialize licensing to correct level when working with geodatabase in Engine environment**
  - If not, get an error on call to open the workspace
- **Configure license at application start time**
  - `esriLicenseProductCodeEngineGeoDB`
- **Enterprise geodatabase editing application requires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

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- **Accessing and Creating Data**
- Beyond Basics And Editing
- Cursors
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

# Workspace Factory and Workspaces

- **Workspace Factory is a dispenser of workspaces**
  - Personal, File, ArcSDE
- **Create a factory from a factory coclass**
- ***A workspace is a container of datasets.***
- **Examples :**
  - Geodatabase, coverage workspace, folder of shapefiles
- **Create a workspace from a factory**
  - Path to data and window handle (app ID)
    - For SDE, can use .sde connection file
  - Propertyset or string containing connection properties



# Creating Datasets

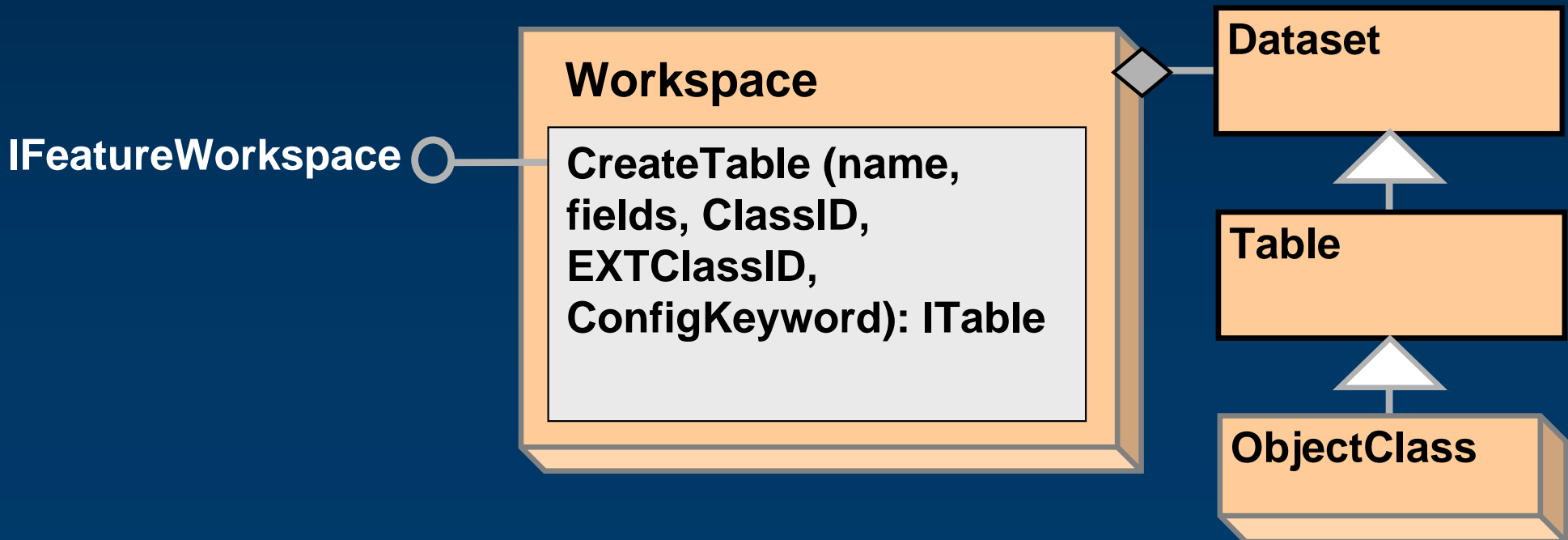
- **Dataset model**
  - Open methods on `IFeatureWorkspace`, `IFeatureClassContainer`, etc
- **Dataset Extensibility model**
  - Open methods on `IDatasetContainer2`
- **Use a name – "Streets"**
  - Use `ISQLSyntax::QualifyTableName` for SDE
  - Owner not required to qualify name
- **Use a ClassID**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

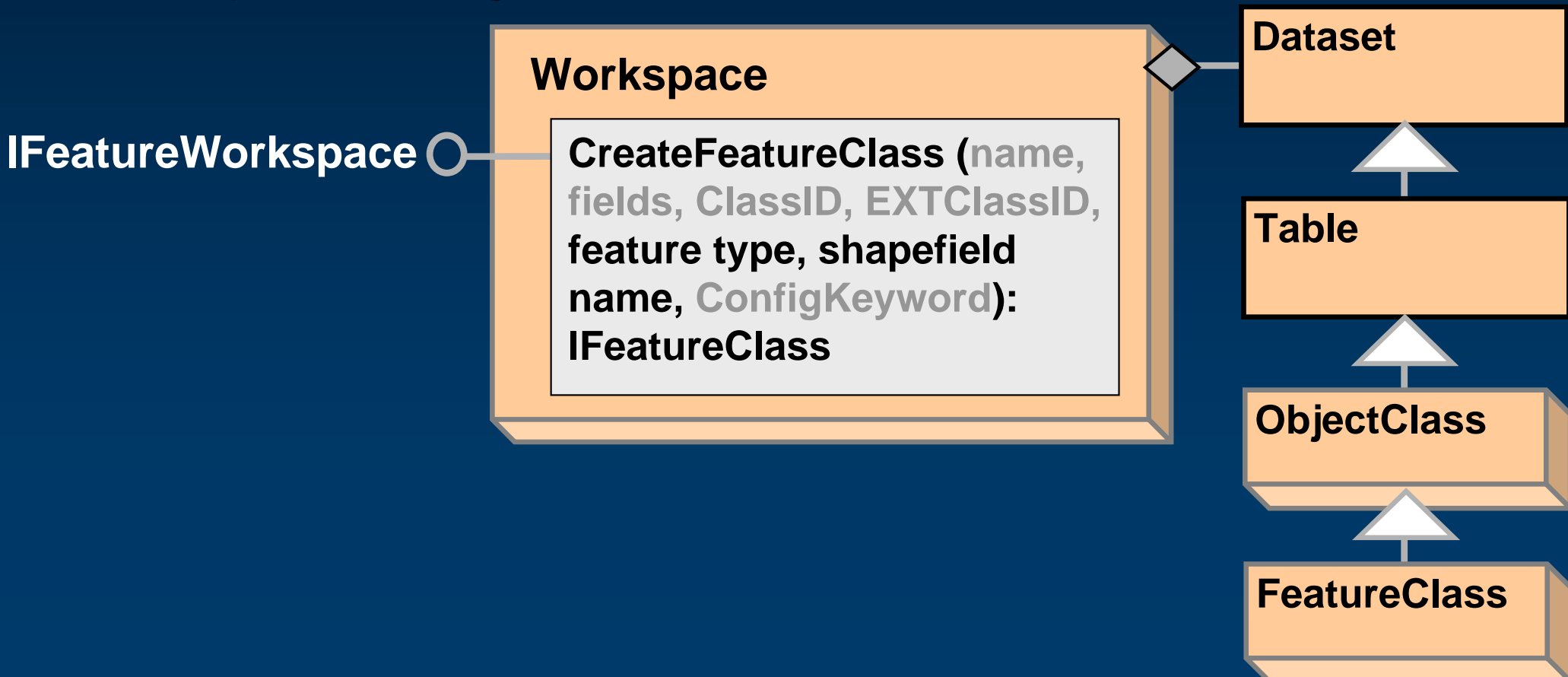
# Create a Table

- Creates an ObjectClass, returns ITable interface
  - ObjectID field. Values are Never Reused.
- Custom behavior ID's (*can use null*)
- SDE configuration keyword (*can use " "*)
- Can create Fields first
  - Use IObjectClassDescription:RequiredFields



# Create a Feature Class

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



# 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**

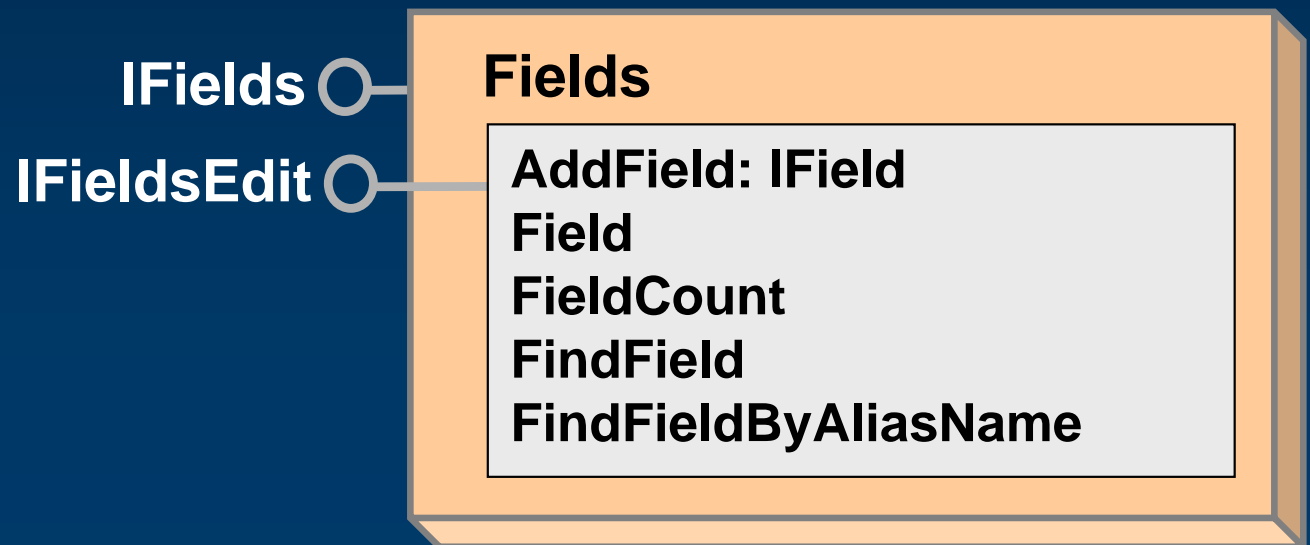
**IField** ○  
**IFieldEdit** ○

## Field

**AliasName: String**  
**Domain: IDomain**  
**Length: Long**  
**Name: String**  
**Precision: Long**  
**Scale: Long**  
**Type: esriFieldType**

# Create a Field ...

- Use a Field collection (Fields) when creating datasets
- Used for setting fields collection when creating feature classes
  - 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

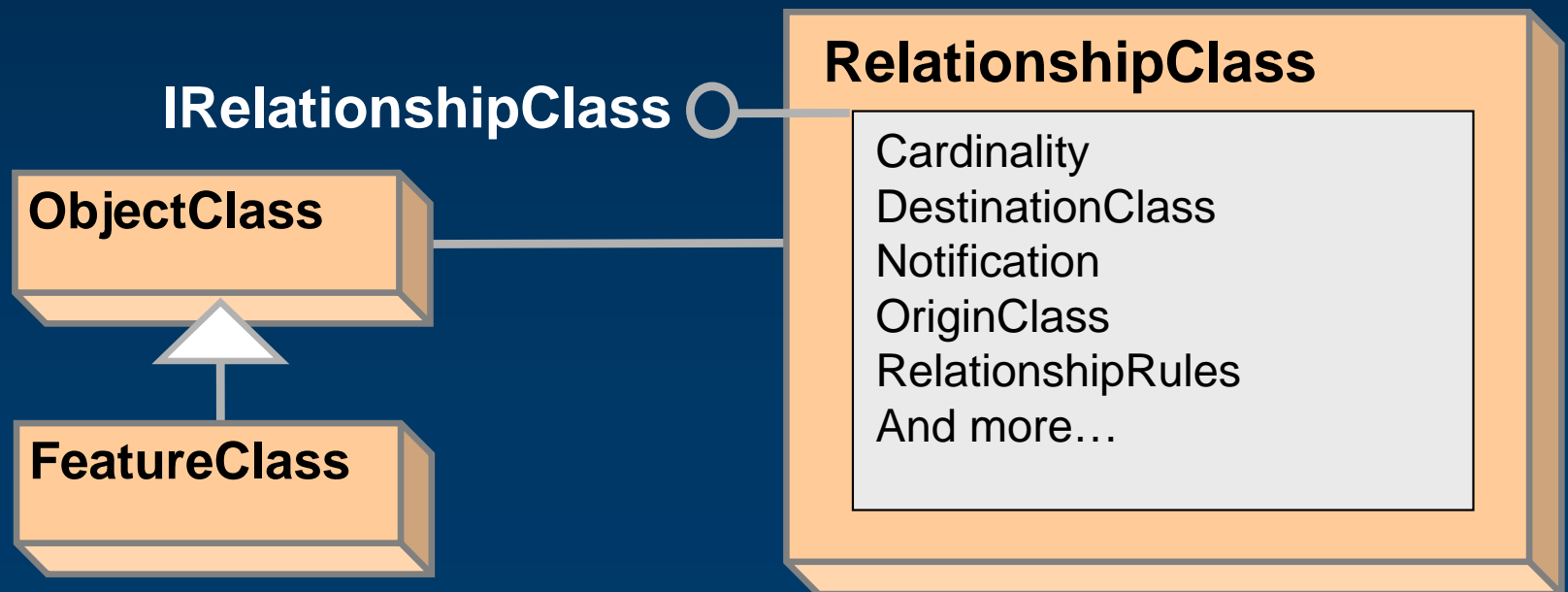


# Domains and Subtypes

- **Domain created at workspace level**
  - Domains constrain field values
  - Associate a domain to a field(s)
    - During create use IFieldEdit:Domain
    - Or IClassSchemaEdit:AlterDomain
- **Subtypes are specific to feature class**
  - Partition the objects in a class into like groups
  - Defined by the value of a subtype field
    - Constrain rules\behavior to the subtype level
    - Once set, need to check attribute, connectivity and relationship rules at subtype level
  - Have a default subtype code
    - Important for feature creation and editing

# RelationshipClass Properties

- Create with IFeatureWorkspace or IRelationshipClassContainer
- You set properties to define the relationship
  - Origin and destination tables
  - Primary – Foreign key relations (fields)
  - RelationshipRules, notification, cardinality
- Find related objects
  - Get related object sets using GetObject methods



# 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 not 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

# Data Access and Creation demo

- **Create a file geodatabase (.gdb)**
- **Create a feature dataset and feature class**
- **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

# 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**

# Data Access and Creation demo

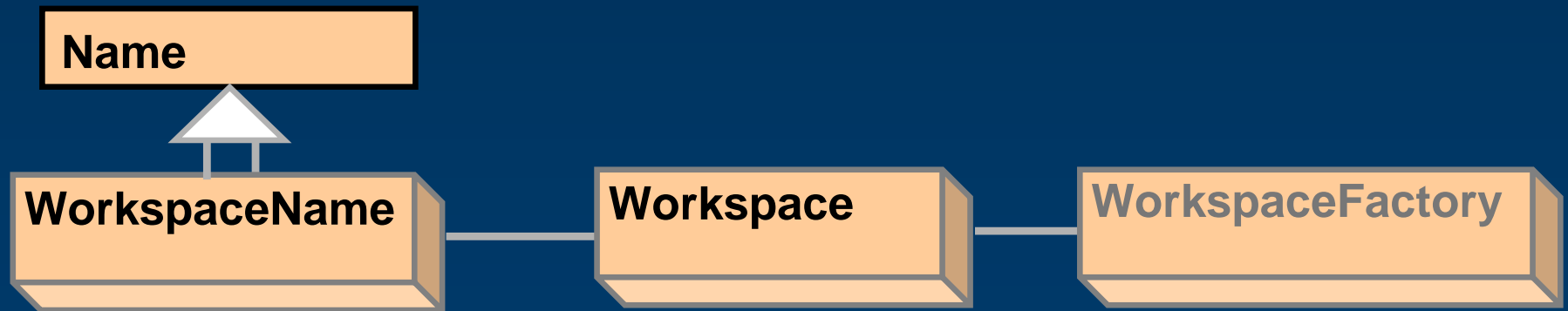
- **Creating features**
- **Set values for the new feature**
- **Store values**

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- Accessing and Creating Data
- **Beyond Basics And Editing**
- Cursors
- Conversions and Loading
- Extending the Geodatabase
- Questions and other information

# 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

# RelQueryTable (Joins) vs QueryDefs

- **Both used to return results from 2 tables**
  - But for different reasons
- **RelQueryTable**
  - Joins two datasets
    - Can be across different data sources
    - Treat like a table or feature class
  - Based on a RelationshipClass or MemoryRelationshipClass
  - Use IDisplayTable in ArcMap (esriCarto.olb) to visualize

# RelQueryTable (Joins) vs QueryDefs ...

- **QueryDefs**

- **Query based on one or more tables**

- Analogous to an SQL Query

- Get a cursor back

- **Tables must be in database**

- Do not have to be registered with the geodatabase

- **Can result in a feature layer**

- Need to use `TableQueryName::IQueryName2` if no ObjectIDs in input tables

# Geometric Networks

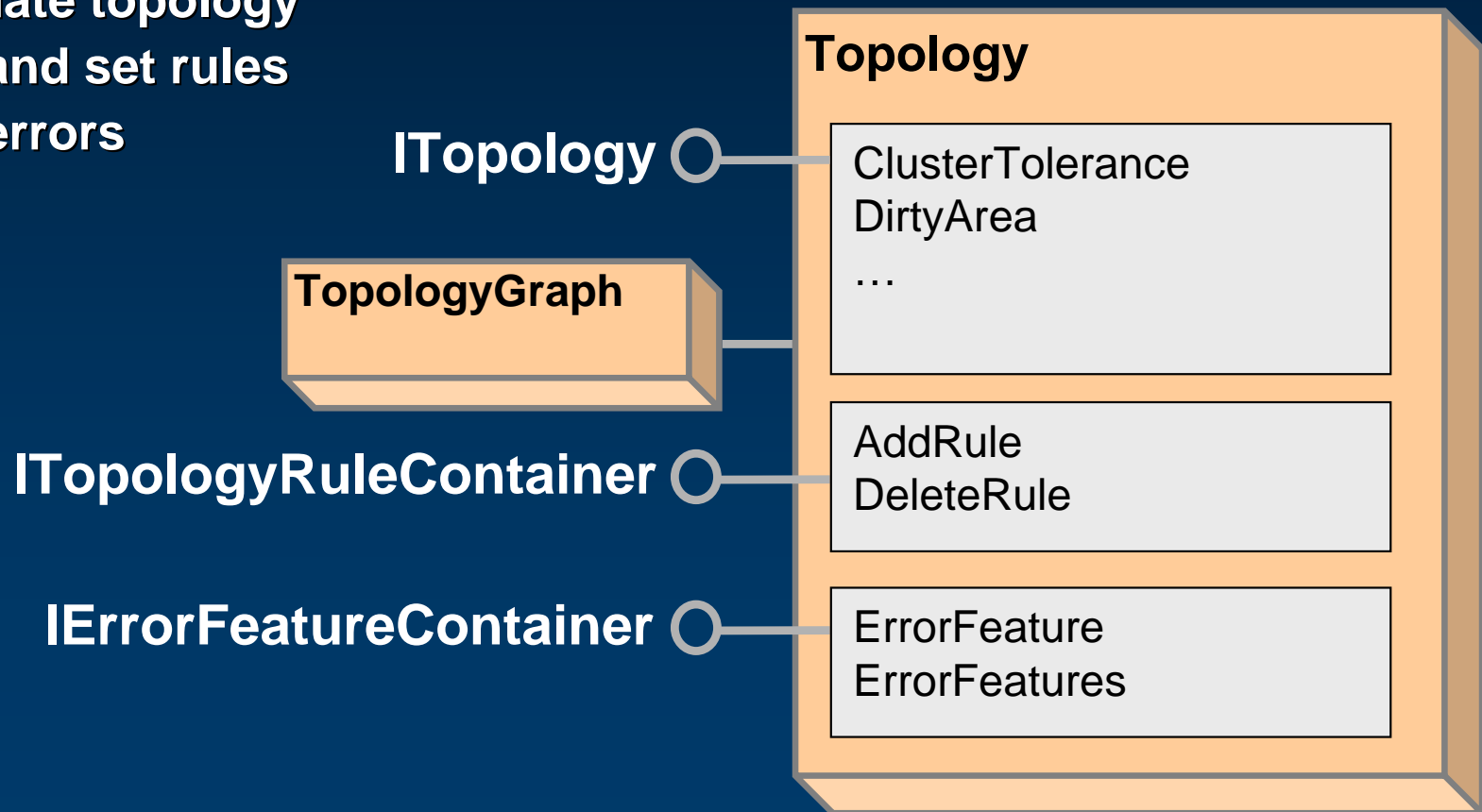
- **Designed for Utilities\Natural Resources industries**
- **Use INetworkLoader for creation of geometric networks**
- **Use logical network API for navigation and tracing whenever possible**
  - **IForwardStar**
- **Navigational APIs available at the geometric network feature level**
  - **Very slow**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

# Network Datasets

- **Designed for Transportation industry**
- **Dataset Extension**
  - Leverages Data Element and IDatasetContainer2 for creation and updates
- **Different than geometric networks**
  - Simple features
  - No custom behavior
- **Need to perform connectivity analysis**
  - INetworkForwardStar
  - INetworkForwardStartAdjacencies
- **Support Custom Evaluators (i.e. Cost, Restrictions) and Custom Solvers**

# Topologies

- A set of rules and behaviors that model how points, lines, and polygons share coincident geometry
- A topology can contain many feature classes and many topology error features
  - Validate topology
  - Get and set rules
  - Get errors

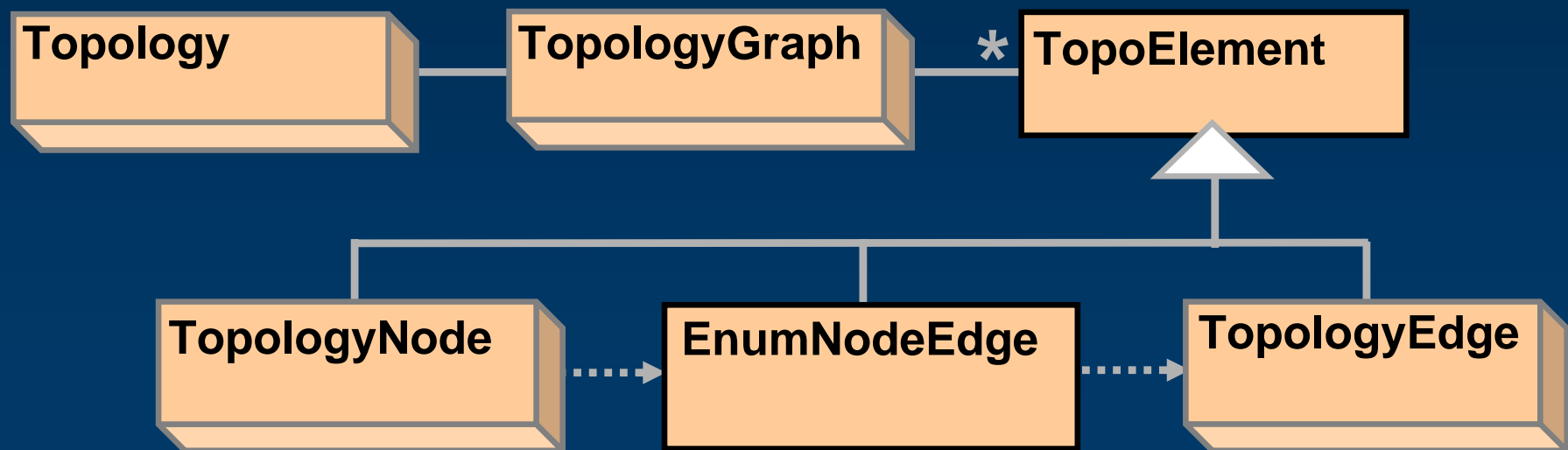


# Topologies ...

- 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 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



# 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**



# Editing the Geodatabase ...

- **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

# Editing the Geodatabase ...

- **Useful methods 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

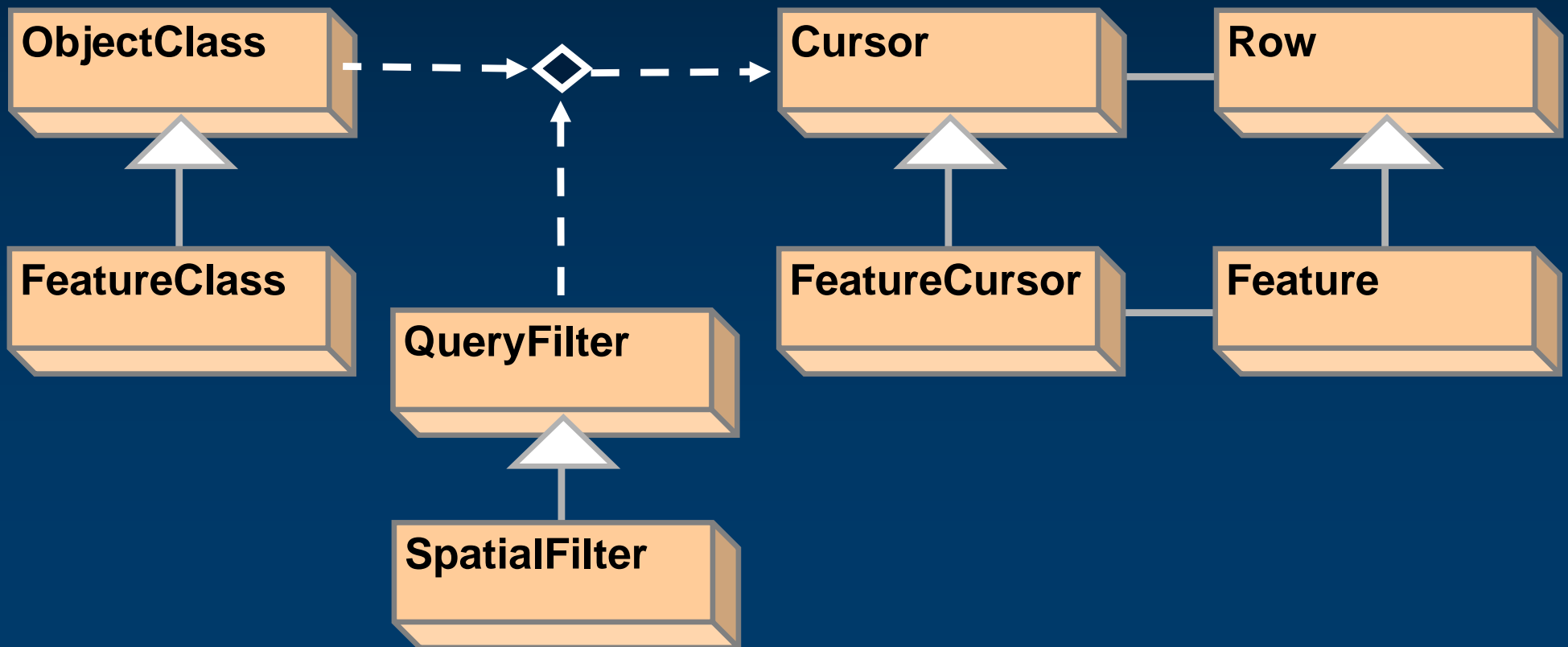
- **List domain values for feature based on subtype**

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- Accessing and Creating Data
- Beyond Basics And Editing
- **Cursors**
- **Conversions and Loading**
- **Extending the Geodatabase**
- **Questions and other information**

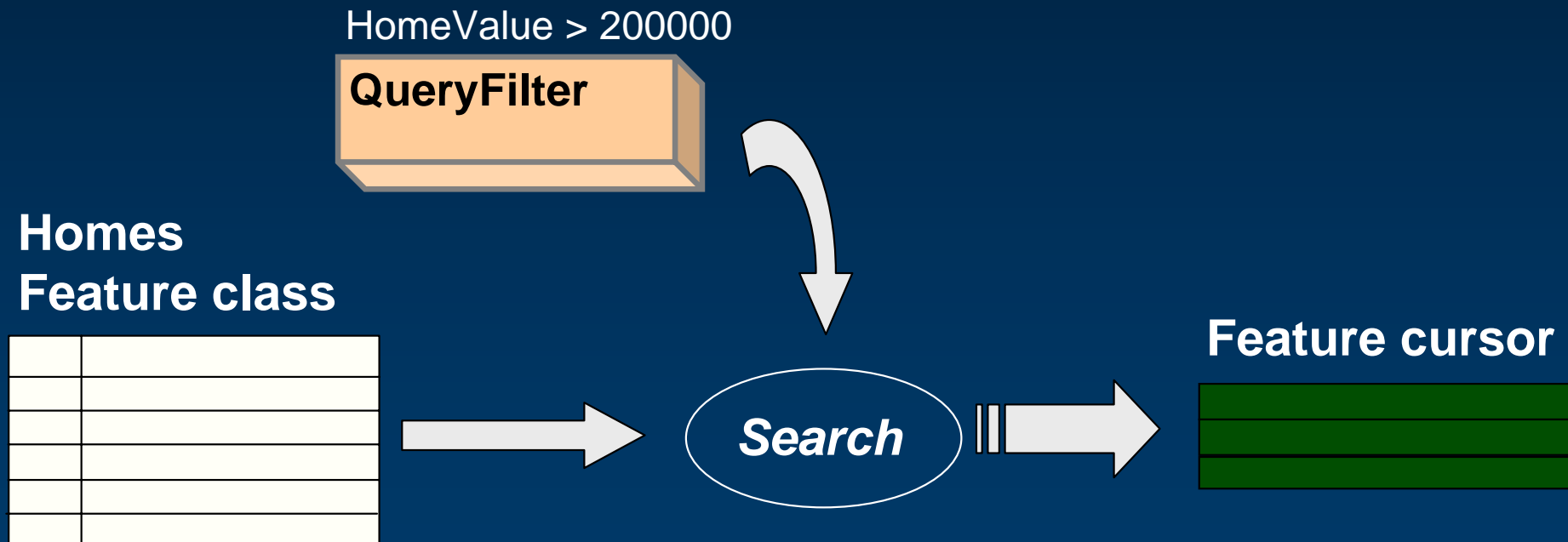
# Cursors

- 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



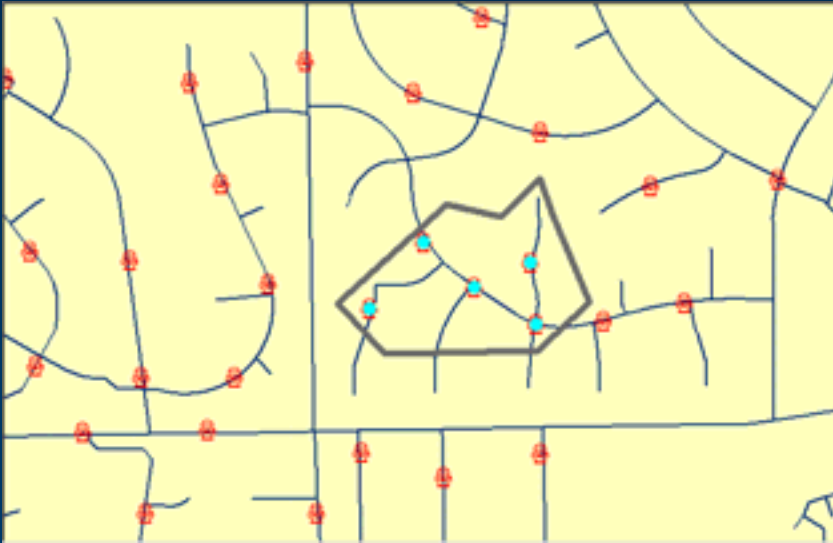
# Creating a Cursor

- QueryFilter contains an SQL-like statement
- The cursor contains a subset
  - No filter\nnothing, all rows returned



# 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**

# Types of 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 Cursors ...

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

# Types of 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

# 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**
- **The need to use the Fields collection of a row and call FindField is rare**

# Cursors

## Example: Efficient Use of FindField

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

    ITable testTable = workspace.openTable("parcels");
    int pidx = testTable.findField("PARCEL_ID");
    int kidx = testTable.findField("PARCELKEY");

    ICursor cursor1 = testTable.ITable_search(null, true);
    IRow row1 = cursor1.nextRow();
    while (row1 != null) {
        System.out.println("PARCEL_ID = " + (String)row1.getValue(pidx));
        System.out.println("PARCEL_KEY = " + (Integer)row1.getValue(kidx));
        row1 = cursor1.nextRow();
    }

    ICursor cursor2 = testTable.ITable_search(null, true);
    IRow row2 = cursor2.nextRow();
    while (row2 != null) {
        System.out.println("PARCEL_ID = " + (String)row2.getValue(pidx));
        System.out.println("PARCEL_KEY = " + (Integer)row2.getValue(kidx));
        row2 = cursor2.nextRow();
    }
}
```

# Cursors

## Example: Inefficient Use of FindField

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

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

    ICursor cursor1 = testTable.ITable_search(null, true);
    IRow row1 = cursor1.nextRow();
    while (row1 != null) {
        System.out.println("PARCEL_ID = " + (String)row1.getValue(cursor1.findField("PARCEL_ID")));
        System.out.println("PARCEL_KEY = " + (Integer)row1.getValue(cursor1.findField("PARCELKEY")));
        row1 = cursor1.nextRow();
    }

    ICursor cursor2 = testTable.ITable_search(null, true);
    IRow row2 = cursor2.nextRow();
    while (row2 != null) {
        System.out.println("PARCEL_ID = " + (String)row2.getValue(cursor2.findField("PARCEL_ID")));
        System.out.println("PARCEL_KEY = " + (Integer)row2.getValue(cursor2.findField("PARCELKEY")));
        row2 = cursor2.nextRow();
    }
}
```

# Cursor demo

- **Cursors examples**
  - Search
  - Update
  - Insert

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- Accessing and Creating Data
- Beyond Basics And Editing
- Cursors
- **Conversions and Loading**
- Extending the Geodatabase
- Questions and other information

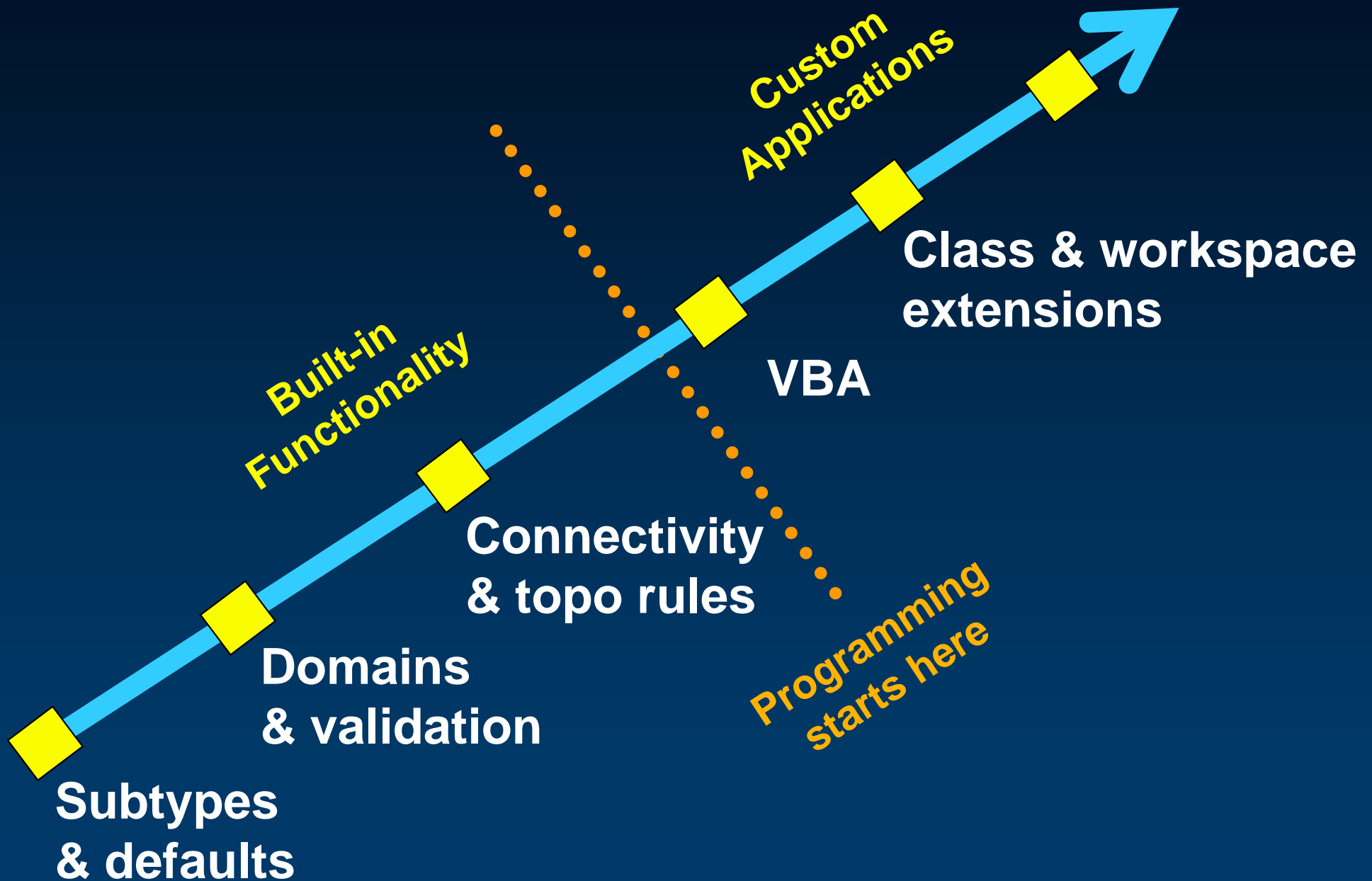
# Conversion and Loading

- **Loading into the geodatabase**
  - **IFeatureDataConverter**
    - Used for converting simple features only
    - A lot of set-up required
  - **Geoprocessing tools**
    - Course grained, facilitates 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 geoprocessing tools**

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- Accessing and Creating Data
- Beyond Basics And Editing
- Cursors
- 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**

- **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**

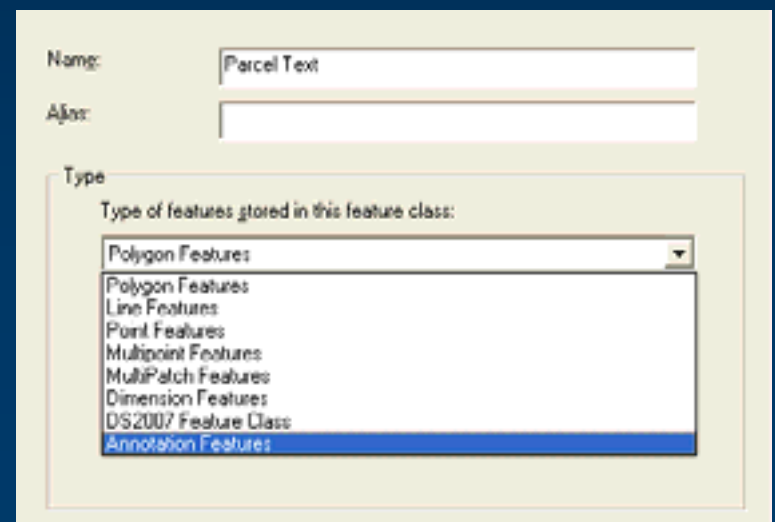
- Performance can suffer, since code is executed redundantly for every feature.
    - Handling of row and relationship events is less stable than class extensions.
    - Technically challenging to implement.

- **Problem can often be solved by class extension or application customization**

# Class extension uses

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

Your description here →  
(In ArcCatalog)



Name: Parcel Text

Alias:

Type

Type of features stored in this feature class:

- Polygon Features
- Line Features
- Point Features
- Multipoint Features
- Multipatch Features
- Dimension Features
- DS2007 Feature Class
- Annotation Features

# Presentation Outline

- Introduction to Geodatabase Programming
- Licensing
- Accessing and Creating Data
- Beyond Basics And Editing
- Cursors
- Conversions and Loading
- Extending the Geodatabase
- **Questions and other information**

# Other Sessions to Attend

- All sessions are in the Catalina/Madera rooms
- Implementing Enterprise Applications with the Geodatabase
  - Tuesday, 20th, 4:30-5:45 pm
- Using the SQL API
  - Wednesday, 21st , 10:30-11:45 am
- Turbocharged Geodatabase Programming
  - Wednesday, 21st , 1-2:15 pm
- Distributed Geodatabase for Developers
  - Wednesday, 21st, 4:30-5:45 pm
- Raster Data Management in ArcGIS 9.2
  - Thursday, 22nd , 8:30-9:45 am

# Further Questions

- **TECH-TALK**

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

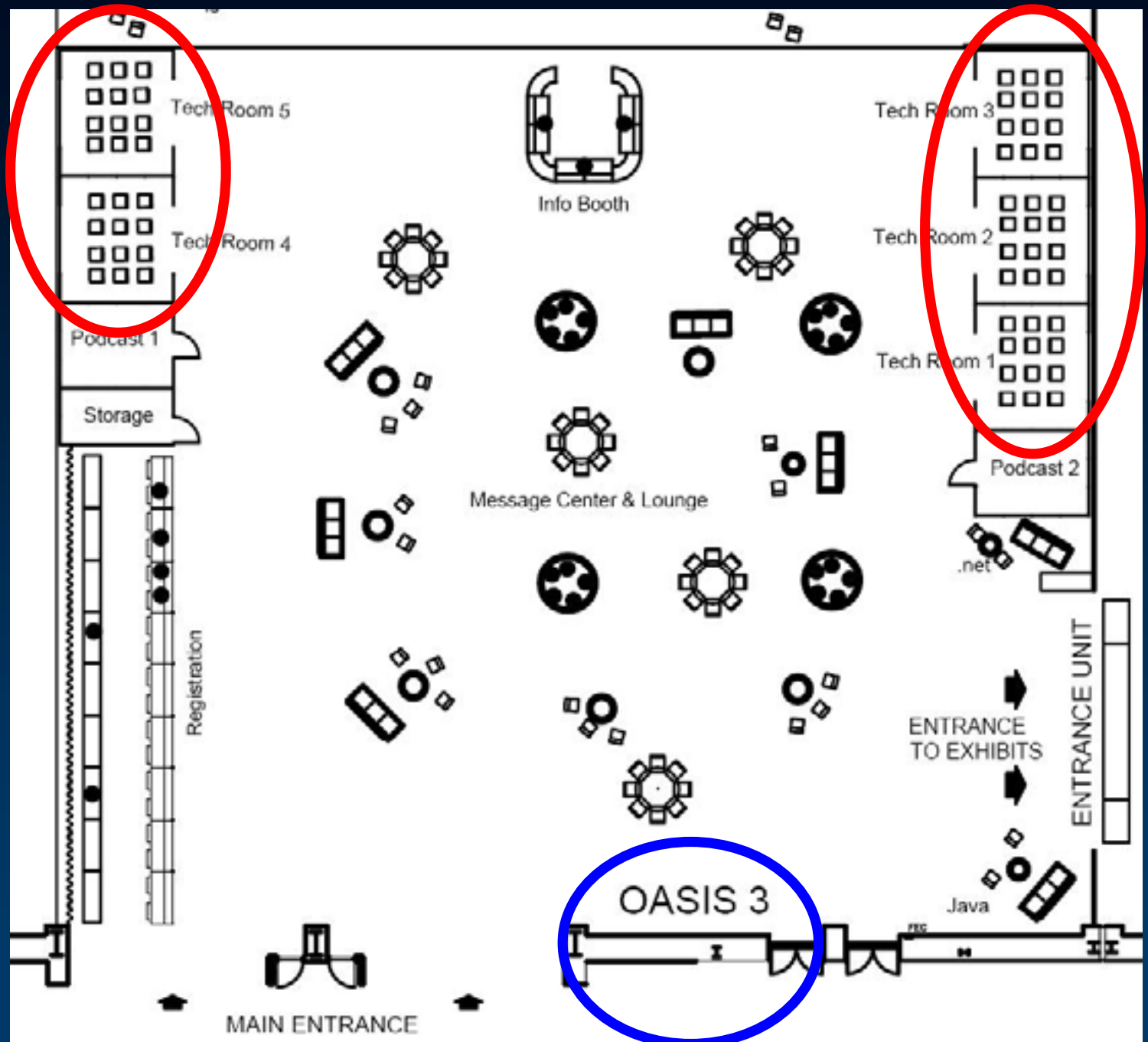
- **Meet the Teams**

- **When:** Wednesday, 11:00am, Message Center and Lounge, Oasis 3

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

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

# Tech Talk Map



# Conclusion

- **Thank you for coming!**
- **Please fill out the Session Surveys at the back of the room**
- **Return to:**
  - **Info Desk**
  - **Registration Desk**
  - **Any available drop box**
  - **Or, leave with us**