Displaying and selecting features

Chapter 17 – **Controlling feature display**

- pp. 315-338
- **Exercises 17A & 17B**

Chapter 18 – **Working with selected features**

- pp. 339-357
- **Exercises 18A & 18B**

Chapter 17 – Controlling feature display

- Making definition queries
- Selecting features and setting the selection color

Chapter 17 – Controlling feature display

- We are now ready to **start working directly with some subsets of spatial datasets**
- In doing so, we are going to **identify part of a set of features**, and use that subset to perform a task **WITHOUT** **changing the underlying data files**
  - The ArcGIS data model is **designed to work with spatial data without changing the underlying files** → A lot of the ideas in the next three chapters are about **working with subsets of features in a temporary fashion** (e.g. so we can do something with 5 out of 50 polygons **WITHOUT** **changing the polygon shapefile where they are stored**)
- **Definition queries** and **feature selections** are two ways to do this

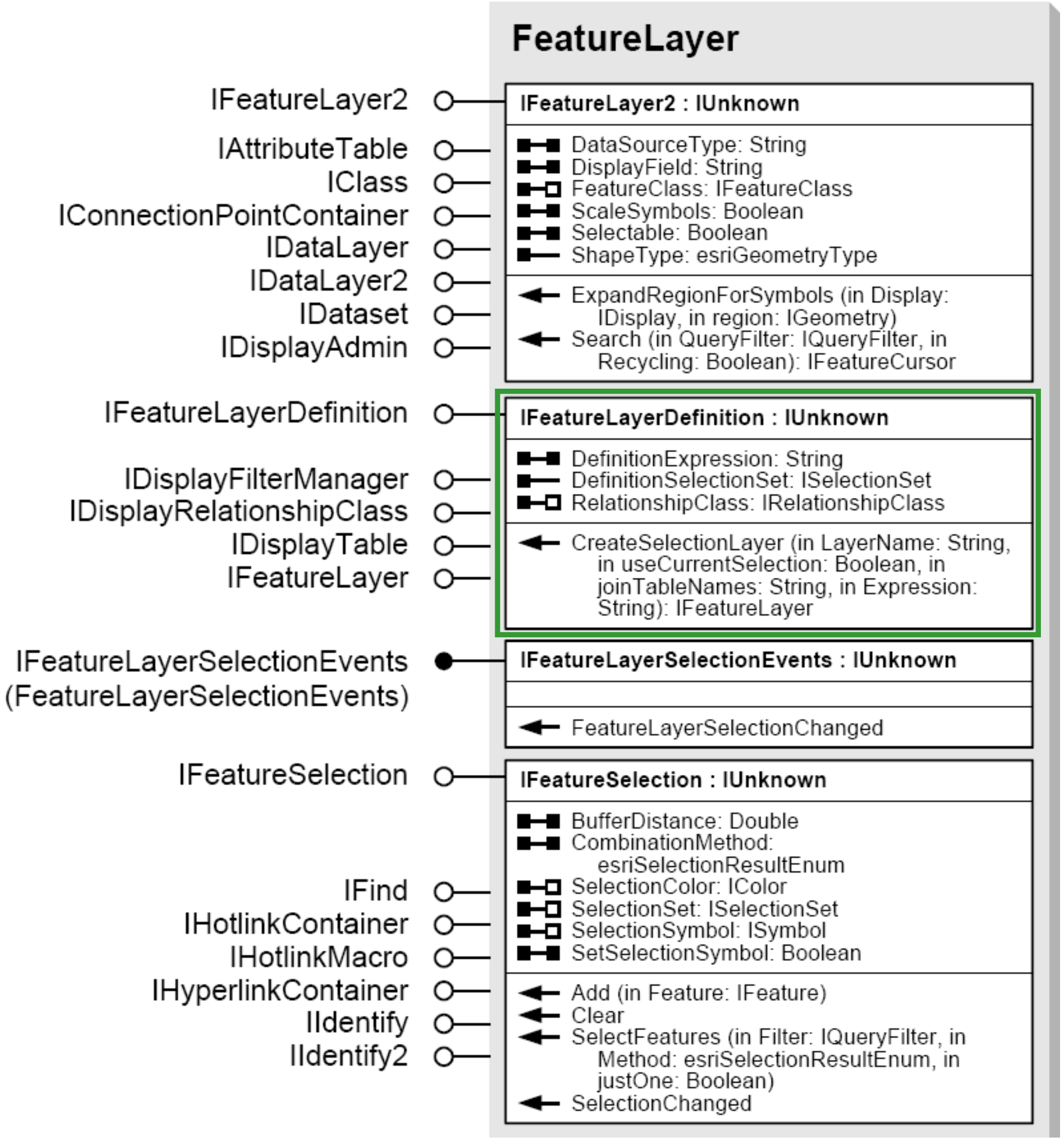
Chapter 17 – Controlling feature display

- Both definition queries and feature selections are **based on the idea of a query**, which you are undoubtedly familiar with from your previous GIS coursework:
  - Given a **set of features**, can we **identify a subset** of them that **meets a particular set of criteria**
  - E.g.: “Which states in the United States have a population of over twelve million?” which as a query, would read:

    ```
    “State_population > 12000000”
    ```
- A query contains a **field name**, an **operator**, and a **value**
- A **definition query limits the features displayed** to include those that meet the criteria
- A **feature selection** highlights the appropriate features

Making definition queries

- Given a **feature layer** in a map, we can **set its definition query** by setting the **DefinitionExpression property** on the **IFeatureLayerDefinition** interface
- However, first we need to have **access to the right layer object** to do this … and this can be tricky with an MxDocument which **contains multiple maps**



FeatureLayer

IFeatureLayer2 ○—
IAttributeTable ○—
IClass ○—
IConnectionPointContainer ○—
IDataLayer ○—
IDataLayer2 ○—
IDataset ○—
IDisplayAdmin ○—
IFeatureLayerDefinition ○—
IDisplayFilterManager ○—
IDisplayRelationshipClass ○—
IDisplayTable ○—
IFeatureLayer ○—
IFeatureLayerSelectionEvents (FeatureLayerSelectionEvents) ●—
IFeatureSelection ○—
IFind ○—
IHotlinkContainer ○—
IHotlinkMacro ○—
IHyperlinkContainer ○—
IIdentify ○—
IIdentify2 ○—

IFeatureLayer2 : IUnknown

DataSourceType: String
DisplayField: String
FeatureClass: IFeatureClass
ScaleSymbols: Boolean
Selectable: Boolean
ShapeType: esriGeometryType

ExpandRegionForSymbols (in Display: IDisplay, in region: IGeometry)
Search (in QueryFilter: IQueryFilter, in Recycling: Boolean): IFeatureCursor

IFeatureLayerDefinition : IUnknown

DefinitionExpression: String
DefinitionSelectionSet: ISelectionSet
RelationshipClass: IRelationshipClass

CreateSelectionLayer (in LayerName: String, in useCurrentSelection: Boolean, in joinTableNames: String, in Expression: String): IFeatureLayer

IFeatureLayerSelectionEvents : IUnknown

FeatureLayerSelectionChanged

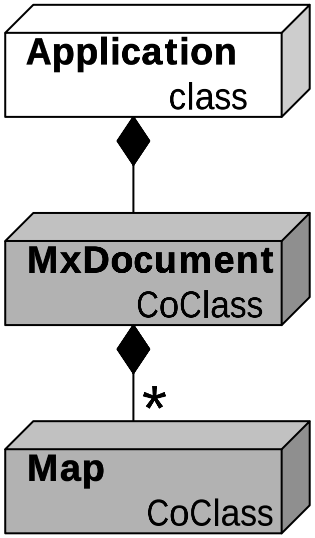
IFeatureSelection : IUnknown

BufferDistance: Double
CombinationMethod: esriSelectionResultEnum
SelectionColor: IColor
SelectionSet: ISelectionSet
SelectionSymbol: ISymbol
SetSelectionSymbol: Boolean

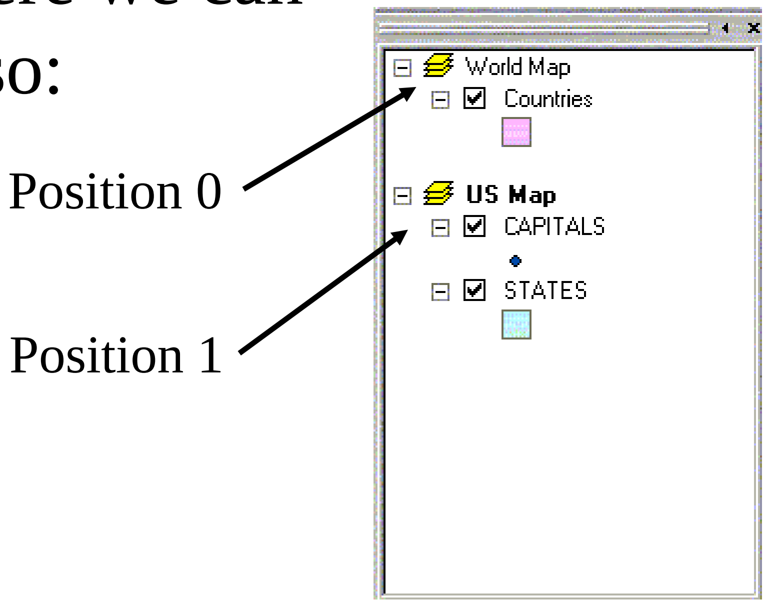
Add (in Feature: IFeature)
Clear
SelectFeatures (in Filter: IQueryFilter, in Method: esriSelectionResultEnum, in justOne: Boolean)
SelectionChanged

Making definition queries

- It is possible for an MxDocument to **contain multiple maps**, which we can get at through the **IMaps interface**, which provides access to the **collection of Map objects**
- This works like the **Enums** we worked with last time; it’s a **list of objects** where we can **specify one by its position**, like so:



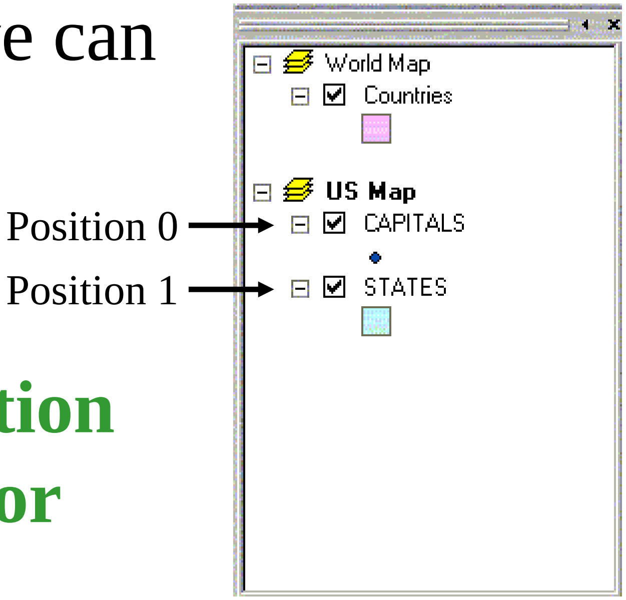
```
Dim pMaps As IMaps
Set pMaps = pMxDoc.Maps
Dim pWorldMap as IMap
Set pWorldMap = pMaps.Item(0)
Dim pUSAMap as IMap
Set pUSAMap = pMaps.Item(1)
```



Making definition queries

- Once we have our appropriate Map, we can easily **set up the layer we need**:

```
Dim pStatesLayer as ILayer
Set pStatesLayer = pUSAMap.Layer(1)
```



Position 0

Position 1

- We can use the **IFeatureLayerDefinition** interface to **set the definition query for the layer**:

```
Dim pStatesLayerDef as IFeatureLayerDefinition
Set pStatesLayerDef = pStatesLayer
pStatesLayerDef.DefinitionExpression _
    “State_Population > 12000000”
```

- The query statement is **double quoted** because it is a string (special rules exist to handle **strings within strings**)

Making definition queries

- In Exercise 17A, you will use a definition query that **specifies one state in a layer of the United States**, and the user will select which state using a combo box, containing a pull down list of all the states' name attributes
- The **resulting DefinitionExpression** will look like this:

```
pStateLayerDef.DefinitionExpression _
    “State_Name = ‘Arizona’”
```

- However, we will need to **use some string operators to form the query**, since we will not know before the fact the name of the state in question (as the user will select it in a combo box)

Making definition queries

- We can obtain the name of the state the user selected in the combo box using the **combo box's EditText property**, and we can store that in a string variable:

```
Dim StrState As String
strState = cboStateNames.EditText
```

- The tricky part is **putting together the full query string**, which can do by **concatenating** several strings together
  - **Concatenation** simply means **attaching multiple strings together**, and it is done in VBA using the **&** symbol
- We know we want the **query string to start with**:

```
"State_Name = '"
```

  - A **single quote inside a string becomes a double quote**

Making definition queries

- We also want the **query string to end with a quote**:

```
“‘”
```

- We want to **sandwich the state name we stored in strState in between those two parts**, which we can do by **concatenating** the three pieces like so:

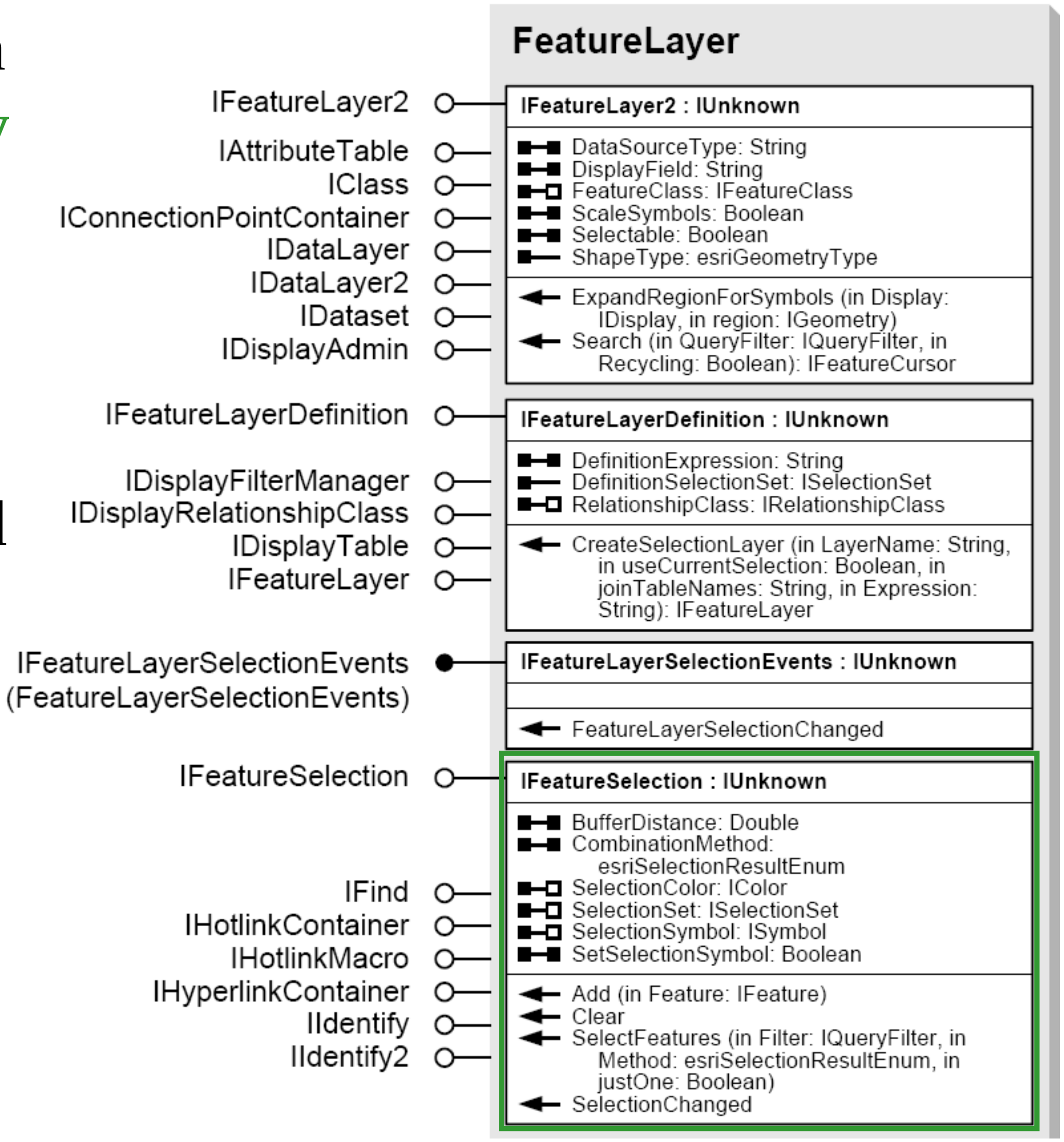
```
“State_Name = ‘” & strState & “‘”
```

- Altogether, that **makes a single string** that we want to use for the definition expression, which we can declare and store, and then use:

```
Dim strQuery As String
strQuery = “State_Name = ‘” & strState & “‘”
pStateLayerDef.DefinitionExpression = strQuery
```

Selecting features and setting the selection color

- **Selecting features** works in a **similar fashion**: A **query is used to specify what to select**, although it uses different objects, interfaces and properties
- The **SelectFeatures** method on the **IFeatureSelection** interface is one way to make a feature selection
- This method requires a **query filter**, a **selection method**, and the **justOne argument**



FeatureLayer

IFeatureLayer2

IAttributeTable
IClass
IConnectionPointContainer
IDataLayer
IDataLayer2
IDataset
IDisplayAdmin

IFeatureLayerDefinition

IDisplayFilterManager
IDisplayRelationshipClass
IDisplayTable
IFeatureLayer

IFeatureLayerSelectionEvents
(FeatureLayerSelectionEvents)

IFeatureSelection

IFind
IHotlinkContainer
IHotlinkMacro
IHyperlinkContainer
IIdentify
IIdentify2

IFeatureLayer2 : IUnknown

DataSourceType: String
DisplayField: String
FeatureClass: IFeatureClass
ScaleSymbols: Boolean
Selectable: Boolean
ShapeType: esriGeometryType

ExpandRegionForSymbols (in Display: IDisplay, in region: IGeometry)
Search (in QueryFilter: IQueryFilter, in Recycling: Boolean): IFeatureCursor

IFeatureLayerDefinition : IUnknown

DefinitionExpression: String
DefinitionSelectionSet: ISelectionSet
RelationshipClass: IRelationshipClass

CreateSelectionLayer (in LayerName: String, in useCurrentSelection: Boolean, in joinTableNames: String, in Expression: String): IFeatureLayer

IFeatureLayerSelectionEvents : IUnknown

FeatureLayerSelectionChanged

IFeatureSelection : IUnknown

BufferDistance: Double
CombinationMethod: esriSelectionResultEnum
SelectionColor: IColor
SelectionSet: ISelectionSet
SelectionSymbol: ISymbol
SetSelectionSymbol: Boolean

Add (in Feature: IFeature)
Clear
SelectFeatures (in Filter: IQueryFilter, in Method: esriSelectionResultEnum, in justOne: Boolean)
SelectionChanged

Selecting features and setting the selection color

- A **QueryFilter** is an object that can be **used to build and store query statements**
  - The query string is stored in the **WhereClause** property:

```
Dim pFilter As IQueryFilter
Set pFilter = NewQueryFilter
pFilter.WhereClause = “State_Name = ‘Arizona’”
```

- There are five types of **selection methods** that can be used for the **second argument** of the **SelectFeatures method**:
  - esriSelectionResultNew – Create **totally new** selection
  - esriSelectionResultAdd – **Add features** to current selection
  - esriSelectionResultSubtract – **Remove features** from current selection
  - esriSelectionResultAnd – **Select features** from current selection
  - esriSelectionResultXOR – **Reverse status** of features satisfying query

Selecting features and setting the selection color

- The **justOne argument** of the **SelectFeatures method** is a Boolean argument that **specifies whether to find**:
  - **The first feature** that satisfies the query (when true) OR
  - **All features** that satisfy the query (when false)
- **Putting all three arguments together**, the code that would use the SelectFeatures method with a QueryFilter called pFilter, performing a query where the results are used in an entirely new selection, and would only look for the first feature that satisfies the query would be:

```
pFSLayer.SelectFeatures _
    pFilter, esriSelectionResultNew, True
```

Chapter 18 – Working with selected features

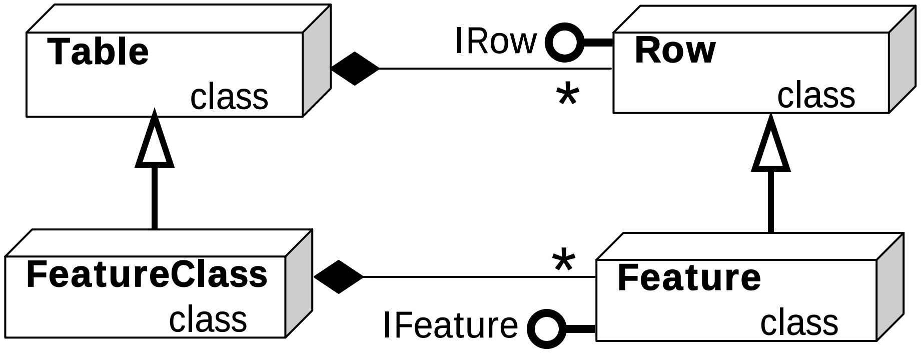
- Using selection sets
- Using cursors

Chapter 18 – Working with selected features

- Now that we know **how to select a set of features**, we will next learn **how to do something** with them
- **Selection sets** collect selected features as a **group**
  - A selection set is a **container** for a set of features
  - Like all **collection objects** we can **add and remove items**
  - **Unlike other collections** we have worked with, you **<u>CANNOT</u>** **access particular objects** in the selection set
  - One **important property** a selection set does have is a **Count property** to **report the total number of features** it contains
- To work with **selected features one at a time**, you make a **cursor**
  - This usage of the word cursor is **different** from indicating the **position of text** being edited in Word

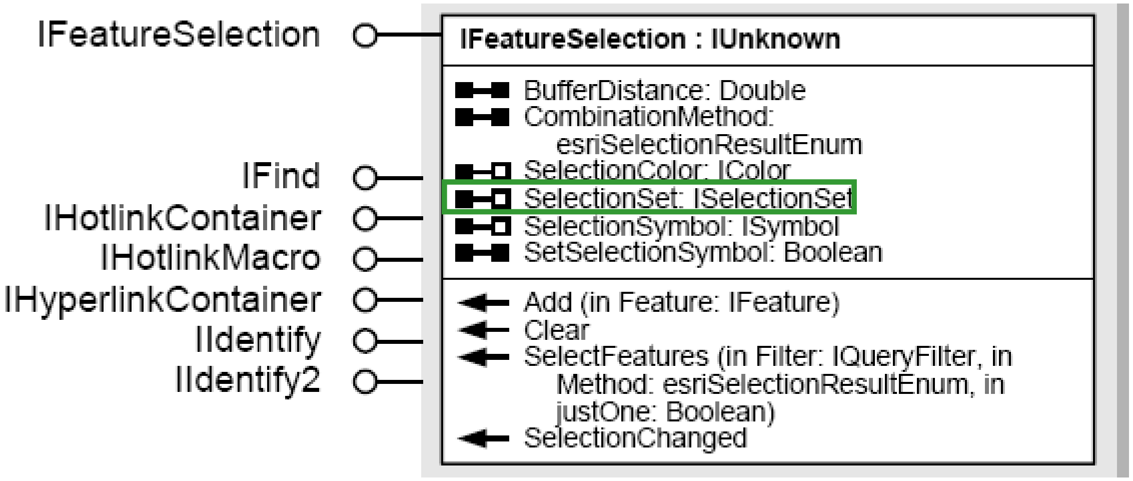
Chapter 18 – Working with selected features

- A **cursor** is like an Enum, with **a pointer** and **method to move from one object to the next** (e.g. in a selection set)
- It can be used to **obtain and modify a feature's spatial and attribute information**
  - When it comes to **editing features** to store (for example) the results of some analysis you just performed using VBA code that you wrote, **a cursor is used** to write results to feature datasets
- Selection sets and cursors are **made up of records**
  - Records refers to both **rows in a table** and **features in a feature class** (each of the latter is **composed** of several of the former)



Using selection sets

- Every feature layer has a **SelectionSet property**
  - Even **if nothing is selected**; it is still there, just **empty**


IFeatureSelection
IFind
IHotlinkContainer
IHotlinkMacro
IHyperlinkContainer
IIdentify
IIdentify2
IFeatureSelection : IUnknown
BufferDistance: Double
CombinationMethod: esriSelectionResultEnum
SelectionColor: IColor
SelectionSet: ISelectionSet
SelectionSymbol: ISymbol
SetSelectionSymbol: Boolean
Add (in Feature: IFeature)
Clear
SelectFeatures (in Filter: IQueryFilter, in Method: esriSelectionResultEnum, in justOne: Boolean)
SelectionChanged

- Whether **user-defined** (using parts of the GUI like the **Select Features tool** or the **Selection menu**) or **set by code** (using a **QueryFilter** as we saw earlier in this class) we can get the selection set by **getting the SelectionSet property** on the **FeatureLayer's IFeatureSelection interface**

Using selection sets

- We can get the layer of interest **using the usual approach** (see below), letting VBA switch automatically to the **IFeatureSelection interface**, and then **declare a variable to ISelectionSet** and set it equal to the **feature layer's SelectionSet property**:

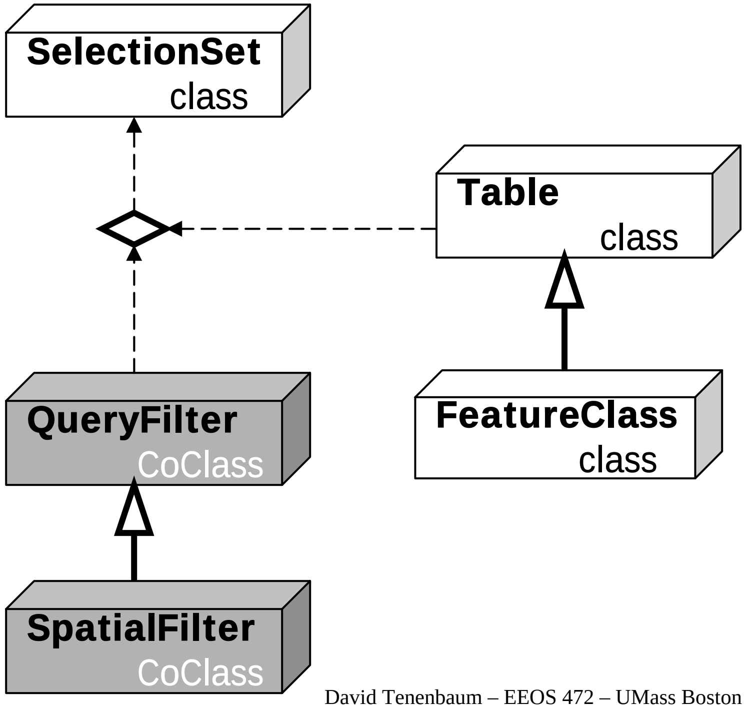
```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap
Dim pFLayer as IFeatureSelection
Set pFLayer = pMap.Layer(0)
Dim pWestSelectionSet As ISelectionSet
Set pWestSelectionSet = pFLayer.SelectionSet
```

Using selection sets

- A feature layer **can have multiple selection sets**, but can **only display one of them at a time**
  - The one displayed is **switched by setting the SelectionSet property**, and then **refreshing the map's active view**

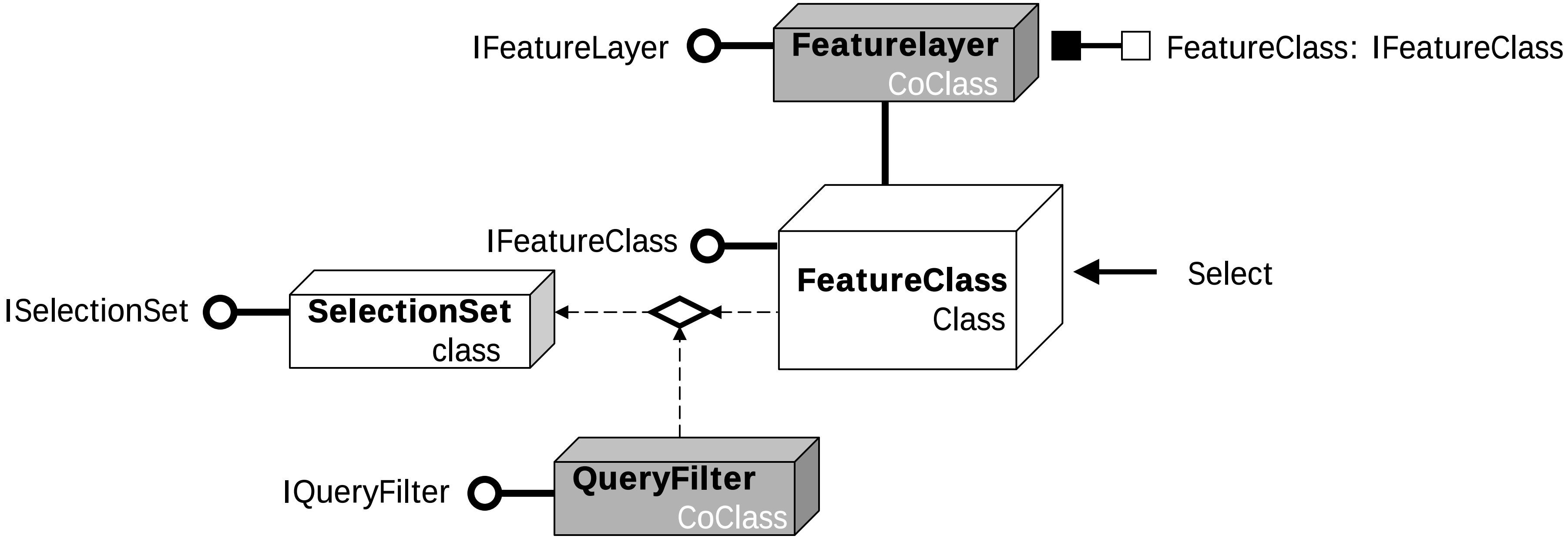
```
Set pFLayer.SelectionSet = pWestSelectionSet
pMxDoc.ActiveView.Refresh
```

- **A Table** and **a QueryFilter** are **both needed** in order to **create a SelectionSet**
  - This is what the **open diamond symbol** in the diagram to the right means (that multiple objects are needed to create another)



Using selection sets

- When you **have the required objects**, you can **make a selection set** by running the **Select method on IFeatureClass**:

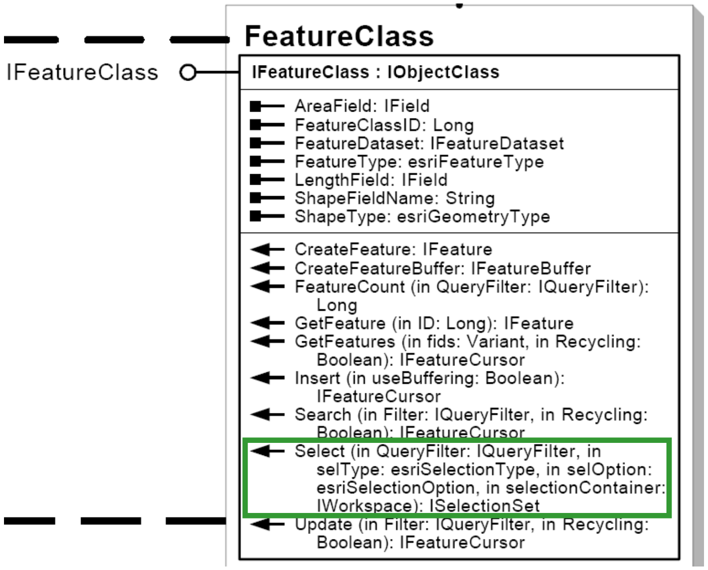


- The **Select method** takes **four arguments**, and **returns** the **ISelectionSet interface** of a selection set

Using selection sets

- The **four arguments** the Select method requires are:

1. A **query filter**
2. A **selection type**, which can be:
   - **esriSelectionTypeIDSet** – ID numbers of the feature are **written to a database table**
   - **esriSelectionTypeSnapshot** – ID numbers of the features are **held in computer memory instead** of being written
   - **esriSelectionTypeHybrid** – ArcGIS **automatically decides** which to use based on the size of the selection set



FeatureClass

IFeatureClass

IFeatureClass : IObjectClass

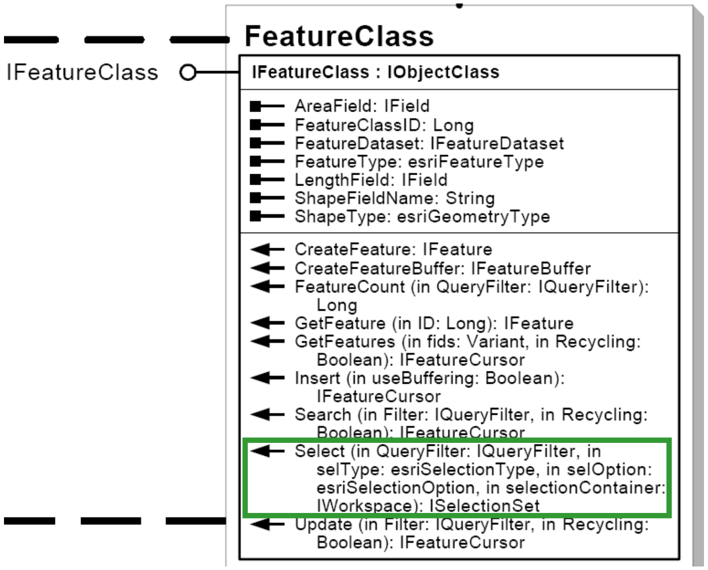
AreaField: IField
FeatureClassID: Long
FeatureDataset: IFeatureDataset
FeatureType: esriFeatureType
LengthField: IField
ShapeFieldName: String
ShapeType: esriGeometryType

CreateFeature: IFeature
CreateFeatureBuffer: IFeatureBuffer
FeatureCount (in QueryFilter: IQueryFilter): Long
GetFeature (in ID: Long): IFeature
GetFeatures (in fids: Variant, in Recycling: Boolean): IFeatureCursor
Insert (in useBuffering: Boolean): IFeatureCursor
Search (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor
Select (in QueryFilter: IQueryFilter, in selType: esriSelectionType, in selOption: esriSelectionOption, in selectionContainer: IWorkspace): ISelectionSet
Update (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor

Using selection sets

- The **four arguments** the Select method requires are (cont.):

3. A **selection option**, which can be:
    - **esriSelectionOptionNormal** – all features that meet the specified criteria
    - **esriSelectionOptionOnlyOne** – only the first feature that meets the specified criteria
    - **esriSelectionOptionEmpty** – An empty selection is created (and it is unclear when you would want this ?!)
4. A **workspace** for **saving the table** created by the 2nd argument
    - The value **Nothing** puts it in the same place as the **feature class**; the **argument is required** even when it **seems unnecessary** (storing results in memory)



FeatureClass

IFeatureClass

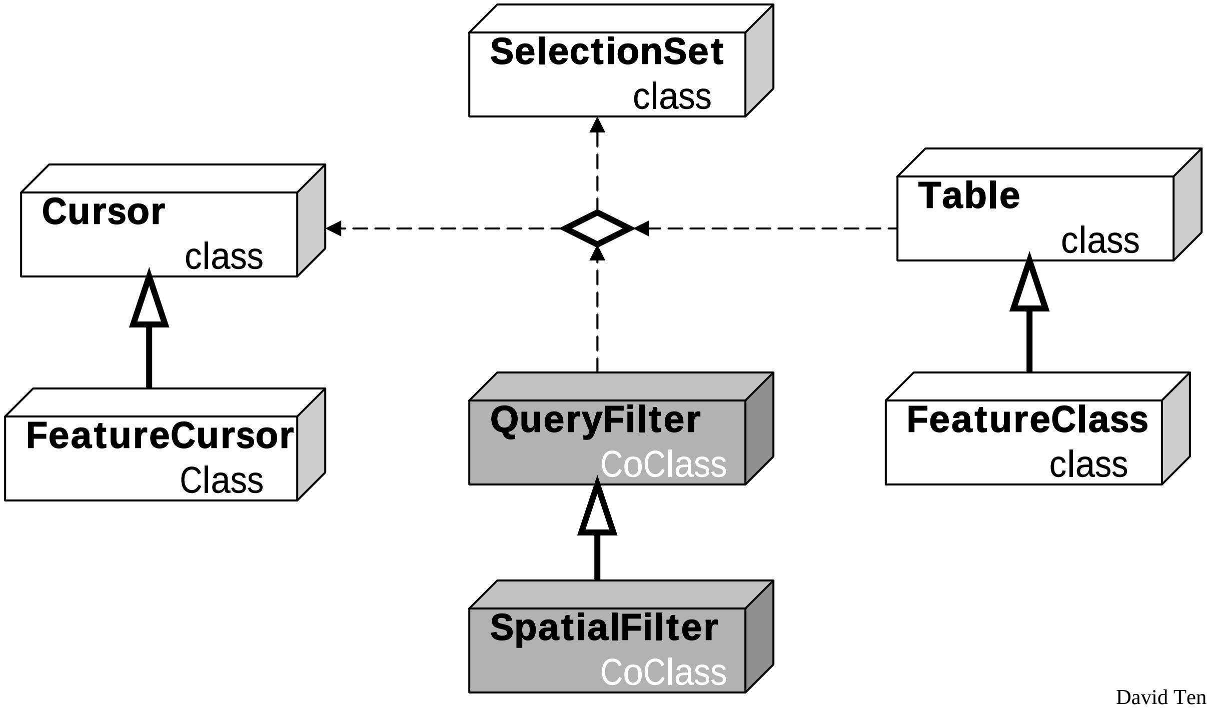
IFeatureClass : IObjectClass

AreaField: IField
FeatureClassID: Long
FeatureDataset: IFeatureDataset
FeatureType: esriFeatureType
LengthField: IField
ShapeFieldName: String
ShapeType: esriGeometryType

CreateFeature: IFeature
CreateFeatureBuffer: IFeatureBuffer
FeatureCount (in QueryFilter: IQueryFilter): Long
GetFeature (in ID: Long): IFeature
GetFeatures (in fids: Variant, in Recycling: Boolean): IFeatureCursor
Insert (in useBuffering: Boolean): IFeatureCursor
Search (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor
Select (in QueryFilter: IQueryFilter, in selType: esriSelectionType, in selOption: esriSelectionOption, in selectionContainer: IWorkspace): ISelectionSet
Update (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor

Using cursors

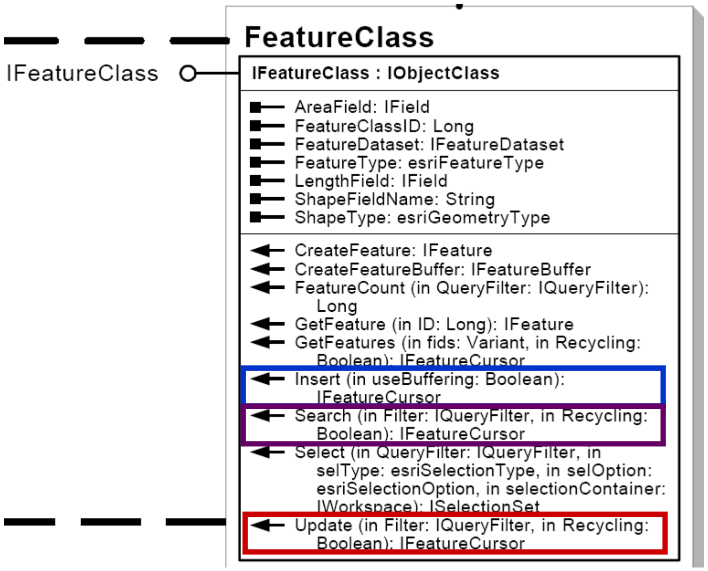
- A **cursor** can be used to **obtain and modify a feature’s spatial and attribute information**
  - It is a **group of records organized in rows**, like a table
  - It is **created** using a **query filter** and a **table**
  - A **FeatureCursor** is a type of cursor for use with features



Using cursors

- The **IFeatureClass interface** (which we used to make selection sets) also has **three methods** to **make a feature cursor**:

1. The **Insert method** lets you **add new features** to a feature class
2. The **Update method** lets you **edit existing features**
3. The **Search method** makes a cursor that **contains all features satisfying a query statement**
   – This is useful when you want to **get information** about features but **do not want to make any new features**


FeatureClass
IFeatureClass
IFeatureClass : IObjectClass
AreaField: IField
FeatureClassID: Long
FeatureDataset: IFeatureDataset
FeatureType: esriFeatureType
LengthField: IField
ShapeFieldName: String
ShapeType: esriGeometryType
CreateFeature: IFeature
CreateFeatureBuffer: IFeatureBuffer
FeatureCount (in QueryFilter: IQueryFilter): Long
GetFeature (in ID: Long): IFeature
GetFeatures (in fids: Variant, in Recycling: Boolean): IFeatureCursor
Insert (in useBuffering: Boolean): IFeatureCursor
Search (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor
Select (in QueryFilter: IQueryFilter, in selType: esriSelectionType, in selOption: esriSelectionOption, in selectionContainer: IWorkspace): ISelectionSet
Update (in Filter: IQueryFilter, in Recycling: Boolean): IFeatureCursor

Using cursors

- Provided you have a **feature class** and a **query filter**, you can **create a search cursor** with two lines of code:

```
Dim pFCursor as IFeatureCursor
Set pFCursor = pStateFClass.Search(pFilter, True)
```

- To **move through the features** in a cursor, use the **NextFeature method**, which simply **increments** through the cursor **one feature at a time**
  - Initially, think of the cursor having a pointer that points to a (hypothetical) spot **before the first feature** (i.e. if the first feature is the 0th feature, it begins by pointing to the -1th feature), so **the first time you run the NextFeature method**, it **moves to the first feature**:

```
Dim pFeature as IFeature
Set pFeature = pFCursor.NextFeature
```

Using cursors

- The **NextFeature method** returns a feature’s **IFeature interface**, giving access to the its **spatial properties**
- Included amongst these is the **Extent property**, which returns a feature’s **Envelope** (a.k.a. its minimum bounding rectangle)
- In Exercise 17B, we will be **working with polygon features**, and **zooming to their extents**, so it will be useful to be able to **obtain a polygon’s Envelope** like so:

```
Dim pEnvelope As IEnvelope
Set pEnvelope = pFeature.Extent
```

- We can use this to **set the zoom of a Map’s ActiveView**:

```
pMapsActiveView as IActiveView
Set pMapsActiveView = pMap
pMapsActiveVoew.Extent = pEnvelope
```

Next Topic:

Working with layouts
and editing data