Using Geoprocessing Tools

Getting Started with Geoprocessing

About Getting Started with Geoprocessing
ArcGIS 9.2 includes a VB Project reference called ESRI Geoprocessing Object Library containing the interface IGeoProcessor and the object class GeoProcessor.

Geoprocessing is a fundamental part of ArcGIS. Whether you’re a new user or an old pro, geoprocessing will become an essential part of your day-to-day work with ArcGIS. Geoprocessing provides the data analysis, data management, and data conversion tools necessary for all GIS users.

Geoprocessing consists of operators, called tools, that operate on the data within ArcGIS (tables, feature classes, rasters, TINs, and so on), and perform tasks that are necessary for manipulating and analyzing geographic information across a wide range of disciplines. Each geoprocessing tool performs an essential operation on geographic data, such as projecting datasets from one map projection to another, adding fields to a table, or creating buffer zones around features. ArcGIS includes over 400 such geoprocessing tools. A geoprocessing tool takes ArcGIS datasets as inputs (such as feature classes, tables, rasters, and CAD files), applies an operation against this data, and creates a newly derived dataset as output.

>> Using the Geoprocessor
A Geoprocessing tool is executed by the Geoprocessor. The Geoprocessor class is the main class that simplifies the task of executing geoprocessing tools. The IGeoProcessor interface Geoprocessor provides access to properties and methods which make it possible to execute tools, set global environment settings, and examine the resulting messages. A GeoProcessor object is the single access point for the execution of any geoprocessing tool in ArcGIS, including extensions.

For an example please see the How to Execute Geoprocessing Tools in Samples of Developers Help.

Accessing tools within a VB6 program

In VB, you execute any tool through the Execute method of the GeoProcessor object. In order to access any method or property of the GeoProcessor object you first need to declare an IGeoProcessor variable and set it to GeoProcessor object. You set the parameters of a tool through a variant array. Populate the variant array with each parameter in the same sequence as the usage of tool would show. You must ensure all the properties marked as required are set; you may also wish to set properties marked as Optional. See the example below showing how to access the Erase tool and set its parameters:
''//Initialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = New GeoProcessor

''//Declare 3 variables for input and output parameters
Dim in_features As String
Dim erase_features As String
Dim out_features As String

''//Define the input/output parameter variables
in_features = "C:\data\erase\Input.shp"
erase_features = "C:\data\erase\Erase.shp"
out_features = "C:\data\erase\Output.shp"

''//Declare and set a variant array to hold the parameters
Dim parameters As IVariantArray
Set parameters = New VarArray

''//Populate the variant array with the parameters
parameters.Add (in_features)
parameters.Add (erase_features)
parameters.Add (out_features)

''//Now execute the Erase tool
GP.Execute "Erase", parameters, Nothing

Running a tool
Each geoprocessing tool has a fixed set of parameters that provides the tool with the
information it needs for execution. Tools usually have input parameters that define the
dataset or datasets that will typically be used to generate new output data. Parameters
have several important properties:

? Name: Each tool parameter has a unique name.
? Type: The type of data expected, such as feature class, integer, string, and raster.
? Direction: The parameter defines input or output values.
? Required: Either a value must be provided for a parameter or it is optional.

When a tool is used in a program, its parameter values must be correctly set so it can
execute when the program is run. The documentation of each tool clearly defines its
parameters and properties. Once a valid set of parameter values are provided the tool is
ready to be executed.

Parameters are specified either as strings or objects. Strings are text values that uniquely
identify a parameter value, such as a path to a dataset or a keyword.

Most tool parameters can be specified as a simple string. In the following code example,
the required parameters for the Buffer tool are defined. In this case, strings are used to
define the input, the output, and the buffer distance properties of Buffer so the call to the
tool is easier to read.
The basic structure of a tool execution remains same as shown in above example. What varies is populating the variant array object, especially the sequence of adding parameters to the VarArray object.

'//Initialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = New GeoProcessor

Dim parameters As IVariantArray
Set parameters = New VarArray
parameters.Add (in_features)
parameters.Add (out_features)
parameters.Add ("12000 Meters") ' //the distance parameter

'//Execute the buffer tool
GP.Execute "Buffer", parameters, Nothing

Below are the system toolbox names and their aliases:

? Analysis Tools— analysis
? Conversion Tools— conversion
? Data Management Tools— management
? 3D Analyst Tools— 3d
? Cartography Tools— cartography
? Geocoding Tools— geocoding
? Geostatistical Analyst Tools— ga
? Linear Referencing Tools— lr
? Multidimension Tools— md
? Network Analyst Tools— na
? Samples— samples
? Spatial Analyst Tools— sa
? Spatial Statistics Tools— stats

>> Running Custom Tools
In addition to using the existing tools and toolboxes provided by ESRI, it is also possible to execute your custom tools such as model tools and script tools which exist in custom toolboxes. The procedure is the same as with system tools. You need to add the custom toolbox to the GeoProccessor first and then, use the name of the custom tool along with the variant array populated with parameters for the Execute method. Here is an example of executing a custom tool (in this case, this is a script tool):

'//Add the custom toolbox to the GeoProccessor object
GP.AddToolbox "C:\Data\MyTools\Toolbox.tbx"

'//Declare and set a variant array to hold the parameters
Dim parameters As IVariantArray
Set parameters = New VarArray

'//Populate the variant array with parameters
parameters.Add "C:\Data\MyTools\streets.shp"

'//'There is a custom script tool named MyScript in the custom toolbox '//'Toolbox. Execute the tool with its name and parameters GP.Execute "MyScript", parameters, Nothing

**Tool Names and Name Conflicts**

Tools have a name and label property. A tool name must be unique for the toolbox containing the tool, but the label has no restrictions. For example, a tool may be named CalculateBestPath, while its label is Calculate Best Path. There must be no other tool in that toolbox named CalculateBestPath, but other tools may have the same label. The tool label is used for displaying the tool in an ArcGIS application and for labeling the tool's dialog box. The tool name is used to execute the tool in the command line, within scripts, and within VB. The name must not have spaces or other restricted characters, such as percent symbols or slashes.

A program typically uses tools from more than one toolbox. When using multiple toolboxes, it is possible that two or more toolboxes will contain a tool with the same name. When this happens, the Geoprocessor is unable to determine which tool in which toolbox should be executed when the tool is referenced in a program. If the Geoprocessor determines that the tool name is ambiguous, VB will provide an error or inform you that the tool is ambiguous.

All toolboxes have an Alias property. The alias is a short, alternative name for the toolbox. The alias is also used to avoid confusion when running tools within a program, when tools have the same name but are stored in different toolboxes. For example, there are two tools named Clip, one in the Analysis toolbox (alias name Analysis) and one in the Data Management toolbox for (alias name Management). When you enter Clip in the command line, you are required to choose either Clip_analysis or Clip_management.

The following are tips for avoiding tool name conflicts:

? Avoid referencing too many toolboxes in your program, since it increases the possibility of a tool name conflict. Add a toolbox when you need its tools; remove it when you no longer need them.

? In the **Execute** method where you specify the tool name, use the Toolbox alias name along with the tool name. Here is an example of executing a tool by tool using the alias:

'//' Execute the tool using alias GP.Execute "Clip_analysis", parameters, Nothing
Tool Return Values

The Geoprocessor always returns the output values of the tool when it is executed. The return value is a variant. Typically, this variant the return value is the path to the output dataset produced or updated by the tool, but it may be other value types, such as a number or a Boolean. If an output for a tool is a multivalue parameter, the values are returned in a string, which consists of multiple strings separated by a semicolon.

Output values are always returned to the program from the Geoprocessor. Return values are necessary when a tool has no output dataset, instead, it has an output scalar value, such as an integer or Boolean. They are also helpful when working with multi-user databases, since output names are qualified with a user's name and table name when outputs are written into the database.

Below are several examples of how return values are captured and what their values could be:

Example 1: Return value is a string representing path to the output

```vba
Dim retValue As Variant
retValue = GP.Execute("Buffer", parameters, Nothing)
```

Example 2: Return value is a double

```vba
Dim vGridIndex As Variant
vGridIndex = GP.Execute("CalculateDefaultGridIndex_management", parameters, Nothing)
```

Listing Tools and Environments

When you initialize the Geoprocessor in your program, you have access to all the tools contained in your system toolboxes. In addition, you may have your own custom toolboxes added. Thus, you may have access to hundreds of tools. The Geoprocessor has several appropriately named methods to return a list of tools (ListTools) and environments (ListEnvironments).

Each method has a wildcard option, and they return a GeoProcessing enumeration list that can be looped through. Here are examples of listing tools and environments:

```vba
'//List Tools – all tools containing the name _analysis
Dim pList As IGPEnumList
Dim sTool As String
Set pList = GP.ListTools("*_analysis")
pList.Reset
While Not sTool = ""
    Debug.Print sTool
    sTool = pList.Next
Wend

'//List all environments by name
Dim pGpEnumList As IGPEnumList
Set pGpEnumList = GP.ListEnvironments("*")

Dim sEnv As String
sEnv = pGpEnumList.Next
```
While Not sEnv = ""
    Debug.Print sEnv
    sEnv = pGpEnumList.Next
Wend

**Using ArcObjects as Tool Input**

ESRI ArcObjects is the development platform for the ArcGIS family of applications, such as ArcMap, ArcCatalog, and ArcScene. The ArcObjects software components expose the full range of functionality available in ArcInfo, ArcEditor and ArcView to developers. These objects may be used to manage geographic data, such as the contents of a geodatabase, shapefiles, and coverages, to name a few. If you are accustomed to working with ArcObjects, you can continue with that object model when working with the Geoprocessor. An ArcObject may be used instead of an ArcCatalog path when defining an input parameter. For example, an IFeatureClass object may be used to define the input to the Clip tool in the Analysis toolbox, while an IRasterDataset object may be used as input to the Slope tool. The CreateFeatureClass tool will not accept an ArcObject as the input location, which could be a folder, geodatabase workspace, or geodatabase feature dataset, since none of these data types may be represented as a layer. An ArcCatalog path must be used to define newly created output. Tools such as Append, updates existing data, so its output may be defined using an ArcObject, such as an IFeatureClass.

Dim GP As IGeoProcessor
Set GP = New GeoProcessor
GP.OverwriteOutput = True
GP.SetEnvironmentValue "workspace", "C:\cities.gdb"

Dim GPU As IGPUtilities
Set GPU = New GPUtilities

'// create the input feature class object
Dim infc As IFeatureClass
Set infc = GPU.OpenFeatureClassFromString ("C:\cities.gdb\redlands")

Dim parameters As IVariantArray
Set parameters = New VarArray

'// add input feature class object to the parameters list
parameters.Add infc
'// add other two parameters as string as path and name are better represented as string
parameters.Add "C:\data\shapefiles"
parameters.Add "outfeats.shp"

'// execute the tool with the parameters as set above
GP.Execute "FeatureClassToFeatureClass_Conversion", parameters, Nothing
Using Environment Settings

Each tool has a set of parameters it uses to execute an operation. Some of these parameters are common among all tools, such as a tolerance or output location. These parameters may obtain their default values from a geoprocessing environment that all tools utilize during their operation. When a tool is executed, the current environment settings may also be used as global input parameter values. Settings, such as an area of interest, the coordinate system of the output dataset, and the cell size of a new raster dataset, may all be specified in the geoprocessing environments.

Here is an example of retrieving and setting environment values:

```vbscript
'//Get environment value: the current workspace is returned
Dim sWksEnv As String
sWksEnv = GP.GetEnvironmentValue "workspace"

'//Set and get the output coordinate system
GP.SetEnvironmentValue "outputCoordinateSystem", "C:\Program Files\ArcGIS\Coordinate Systems\Projected Coordinate Systems\UTM\Nad 1983\NAD 1983 UTM Zone 21N.prj"
Dim strCoordSys As String
strCoordSys = GP.GetEnvironmentValue("outputCoordinateSystem")
```

Access to Licensing and Extensions

Whenever a tool is executed in a program, an ArcGIS Desktop license is required. Tools from ArcGIS extensions, such as ArcGIS Spatial Analyst, require an additional license for that extension. If the necessary licenses are not available, a tool will fail and return the appropriate error messages.

When using an ArcView or ArcEditor license, a program must explicitly use AoInitialize and set the product to ArcView or ArcEditor or the program will fail. By default, the Geoprocessor will always assume an ArcInfo license is required for execution of a script, so if a program does not explicitly set the product required by the tools it executes, an ArcInfo license will be initialized.

```vbscript
'// Initialize the license
Dim pAoInitialize As IAoInitialize
Set pAoInitialize = New AoInitialize

Dim licenseStatus As esriLicenseStatus
licenseStatus = pAoInitialize.Initialize(esriLicenseProductCodeArcInfo)

'// Checkout spatial analyst extension
licenseStatus = pAoInitialize. CheckOutExtension
(esriLicenseExtensionCode.esriLicenseExtensionCodeSpatialAnalyst)
```

Dim GP As IGeoProcessor
Set GP = New GeoProcessor
GP.OverwriteOutput = True

Dim in_raster As String
Dim out_raster As String
in_raster = "C:\data\demlatgrd"
out_raster = "C:\data\aspect03"

Dim parameters As IVarArray
Set parameters = New VarArray

parameters.Add in_raster
parameters.Add out_raster

'// Execute the slope tool
GP.Execute "Slope_3d", parameters, Nothing

'// Release license
licenseStatus = pAoInitialize.CheckInExtension (esriLicenseExtensionCode.esriLicenseExtensionCodeSpatialAnalyst)
pAoInitialize.Shutdown
Set pAoInitialize = Nothing

**Batch Processing with Geoprocessing**

**About Batch Processing**
Companies and organizations that use a GIS, typically have large amounts of information stored in many different data formats. Hundreds of shapefiles, geodatabase feature classes, coverages, grids, and tables may form the backbone of the organization and are used every day. New data may also be created at a staggering rate by converting data from one format to another. The tasks required to maintain current databases and create data suitable for analysis are often daunting because of the volume of data required for processing. Programs are used to meet these requirements, since they are an efficient way of organizing data and executing operations. Programs may be run outside an ArcGIS application at any time. By using an operating system's scheduling mechanism, you may have a program run at a regular date and time.

**Listing Data**
One of the foremost tasks in a batch processing program is cataloging the data that is available so it can iterate through the data during processing. The Geoprocessor has a number of methods built specifically for creating such lists.

These methods work with all different types of data and provide flexibility for restricting a search by name or data category. Below is a list of these methods and their syntax.

- **ListDatasets** (WildCard As String, DatasetType As String)
  Returns an enumeration of datasets in the current workspace based on a query string and type.

- **ListFeatureClasses** (WildCard As String, FeatureType As String, Dataset As String)
  Returns an enumeration of feature classes in the current workspace or optional dataset, based on a query string and type.
ListRasters (WildCard As String, RasterType As String)

Returns an enumeration of rasters in the current workspace based on a query string and type.

ListTables (WildCard As String, TableType As String)

Returns an enumeration of tables in the current workspace based on a query string and type.

ListWorkspaces (WildCard As String, WorkspaceType As String)

Returns an enumeration of workspaces in the current workspace based on a query string and type.

The methods listed above require the workspace environment to be specified previously to identify the location from which the enumeration will be created. A workspace is a directory, database, or dataset containing geographic data (for example, geodatabase, geodatabase feature dataset, coverage, and folder) and other supporting data. The wild card option uses an asterisk to mean any character. More than one asterisk may be used in the wild card string. For example, the wild card *road* can be used to find items that have the word road in their names.

The result of each of these methods is a pointer to IGPEnumList interface. Once the GPEnumList pointer has been obtained with the values you want, you can loop through it in your program to work with each individual value.

All methods have a wild card parameter, which is used to restrict the objects or datasets listed by name. A wild card defines a name filter, and all the contents in the newly created list must pass that filter. For example, you may want to list all the feature classes in a workspace that start with the letter d. The following example shows how this is done:

```vbscript
'//List all feature classes in the workspace starting with d
GP.SetEnvironmentValue "workspace", "C:\data\shapefiles"
Dim pGpEnumList As IGPEnumList
Set pGpEnumList = GP.ListFeatureClasses("d*", ",", ")
pGpEnumList.Reset
Dim sFCname As String
'//First feature class of the list
sFCname = pGpEnumList.Next

'//Access each feature class one by one
While Not sFCname = ""
    Debug.Print sFCname
    sFCname = pGpEnumList.Next
Wend
```

The enumeration may also be restricted to match certain data properties, such as only polygon feature classes, integer fields, or coverage datasets. This is what the Type parameter is used for in all the methods. In the next example, the feature classes in a workspace are filtered using a wild card and a data type, so only polygon feature classes that start with the letter d are in the resulting enumeration:
'//List all feature classes in the workspace starting with d
'//and type polygon.
Dim pGpEnumList As IGPEnumList
Set pGpEnumList = GP.ListFeatureClasses("d*", "POLYGON", ")
pGpEnumList.Reset

Following is another example of how to use an IGPEnumList. The script is used to create
raster pyramids for all the rasters that are TIFF images within a folder.

'//List all TIFF files in the workspace and build pyramids
GP.SetEnvironmentValue "workspace", "C:\data\raster.mdb"
Dim pGpEnumList As IGPEnumList
Set pGpEnumList = GP.ListFeatureClasses("*", "TIF", ")
pGpEnumList.Reset

Dim parameters As IVariantArray
Set parameters = New VarArray

Dim sRaster As String
sRaster = pGpEnumList.Next
While Not sRaster = ""
    Debug.Print sRaster
    parameters.Add (sRaster)
    GP.Execute "Build Pyramids", parameters, Nothing
    sRaster = pGpEnumList.Next
Wend

The default behavior for all list methods is to list all supported types. A keyword is used
to restrict the returned collection to a specific type. The type keywords for each method
are listed in the table below.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListDatasets</td>
<td>All, Feature, Coverage, RasterCatalog, CAD, VPF, TIN, Topology</td>
</tr>
<tr>
<td>ListFeatureClasses</td>
<td>All, Point, Label, Node, Line, Arc, Route, Polygon, Region</td>
</tr>
<tr>
<td>ListFields</td>
<td>All, SmallInteger, Integer, Single, Double, String, Date, OID, Geometry, Blob</td>
</tr>
<tr>
<td>ListWorkspaces</td>
<td>All, Coverage, Access, SDE, Folder</td>
</tr>
<tr>
<td>ListTables</td>
<td>All, dBASE, INFO</td>
</tr>
<tr>
<td>ListRasters</td>
<td>All, ADRG, BIL, BIP, BSQ, BMP, CADRG, CIB, ERS, GIF, GIS, GRID, STACK, IMG, JPEG, LAN, SID, SDE, TIF, RAW, PNG, NITF</td>
</tr>
</tbody>
</table>
A common conversion task is the large scale transfer of data from one format to another. The following program section takes advantage of ListWorkspaces, ListFeatureClasses, ListTables, and ListDatasets to identify all personal geodatabases in a folder and converts the contents of each to file geodatabases.

```vba
Dim GP As IGeoProcessor
Set GP = New GeoProcessor
GP.OverwriteOutput = True

GP.SetEnvironmentValue "workspace", "C:\GP\coredata\global\pgdb"

Dim outwks As String
outwks = "C:\data\fgdb.gdb"

'// identify personal geodatabases in the workspace
Dim workspaces As IGpEnumList
Dim workspace As String
Set workspaces = GP.ListWorkspaces("*", "Access")
workspace = workspaces.Next
While workspace <> ""
    GP.SetEnvironmentValue "workspace", workspace

    Dim parameters As IVariantArray
    Set parameters = New VarArray

    '// identify feature classes and copy to file geodatabase
    Dim FCs As IGpEnumList
    Dim FC As String
    Set FCs = GP.ListFeatureClasses("*", ",", ",")
    FC = FCs.Next
    While FC <> ""
        parameters.Add FC
        parameters.Add outwks & FC
        GP.Execute "Copy_management", parameters, Nothing
        parameters.RemoveAll
        FC = FCs.Next
    Wend

    '// identify feature datasets and copy to file geodatabase
    Dim FDs As IGpEnumList
    Dim FD As String
    Set FDs = GP.ListDatasets("*", ",")
    FD = FDs.Next
    While FD <> ""
        parameters.Add FD
        parameters.Add outwks & FD
        GP.Execute "Copy_management", parameters, Nothing
        parameters.RemoveAll
        FD = FDs.Next
    Wend

    '// identify tables and copy to file geodatabase
    Dim tables As IGpEnumList
    Dim table As String
```

---

**Translation**

```vba
Dim GP As IGeoProcessor
Set GP = New GeoProcessor
GP.OverwriteOutput = True

GP.SetEnvironmentValue "workspace", "C:\GP\coredata\global\pgdb"

Dim outwks As String
outwks = "C:\data\fgdb.gdb"

'// identify personal geodatabases in the workspace
Dim workspaces As IGpEnumList
Dim workspace As String
Set workspaces = GP.ListWorkspaces("*", "Access")
workspace = workspaces.Next
While workspace <> ""
    GP.SetEnvironmentValue "workspace", workspace

    Dim parameters As IVariantArray
    Set parameters = New VarArray

    '// identify feature classes and copy to file geodatabase
    Dim FCs As IGpEnumList
    Dim FC As String
    Set FCs = GP.ListFeatureClasses("*", ",", ",")
    FC = FCs.Next
    While FC <> ""
        parameters.Add FC
        parameters.Add outwks & FC
        GP.Execute "Copy_management", parameters, Nothing
        parameters.RemoveAll
        FC = FCs.Next
    Wend

    '// identify feature datasets and copy to file geodatabase
    Dim FDs As IGpEnumList
    Dim FD As String
    Set FDs = GP.ListDatasets("*", ",")
    FD = FDs.Next
    While FD <> ""
        parameters.Add FD
        parameters.Add outwks & FD
        GP.Execute "Copy_management", parameters, Nothing
        parameters.RemoveAll
        FD = FDs.Next
    Wend

    '// identify tables and copy to file geodatabase
    Dim tables As IGpEnumList
    Dim table As String
```
Set tables = GP.ListTables("*", ")
table = tables.Next
While table <>"
   parameters.Add table
   parameters.Add outwks & table
   GP.Execute "Copy_management", parameters, Nothing
   parameters.RemoveAll
   table = tables.Next
Wend

Using Multiple Inputs
Tools may accept a single input or many inputs, depending on the operation. Tools that
convert or overlay data may accept multiple datasets as input because of the nature of the
operation. In a program, inputs are passed to these tools as a multivalue string, which
uses a semicolon to separate each input within the string. A multivalue string is easy to
construct within a program using the string manipulation functions built into the
language. In this example, a multivalue string is created just by separating three inputs by
semicolons and bounded by a pair of double-quotes.

'//Multiple input feature classes
GP.SetEnvironmentValue "workspace", "C:\data\shapefiles"
Dim parameters As IVarArray
Set parameters = New VarArray

'//There are three input feature classes, they are delimited by ;
parameters.Add ("redlands_citylimit.shp; redlands_faultzn.shp;
   redlands_flood.shp")
'//But only one output feature class
parameters.Add ("redlands_union.shp")

GP.Execute "Union", parameters, Nothing

Using Value Tables
A value table is a flexible object that may be used as input for a multivalue parameter.
The previous examples of multivalue parameter values focus on the text value of the
parameter, which may become difficult to use when there are numerous values with
complex paths. The value table is used to organize the values into a table, so values may
be easily added or removed, eliminating the complexity of parsing a multivalue text
string. The table may be thought of as a virtual matrix of values that is not persisted as an
actual table because it is a device for managing many values in a program.
The number of columns a value table contains should be specified when it is created (the
default is a single column). Each column corresponds to a value in the parameter being
defined. Union, for example, requires the path to a dataset or a layer name, and an
optional priority rank for each input entry. A value table for Union will require two
columns - one for the data and one for the priority rank. The following example shows
how to create and populate a value table for Union. For this example, you will need to
add a reference to the ESRI Geodatabase Object Library to access the data type and value objects.

'//Set the workspace
GP.SetEnvironmentValue "workspace", "C:\data\shapefiles"

Dim parameters As IVariantArray
Set parameters = New VarArray

Dim pValueTable As IGpValueTableObject
Set pValueTable = New GpValueTableObject

'// value table contains the inputs and their priority
'// the first column for input, the second for priority
pValueTable.SetColumns (2)
pValueTable.AddRow ("citylimit.shp 2")
pValueTable.AddRow ("faultzn.shp 1")
pValueTable.AddRow ("flood.shp 3")

parameters.Add (pValueTable.ExportToString)
parameters.Add ("redlands_union.shp")

GP.Execute "Union_Analysis", parameters, Nothing

A value table can be populated with a multivalue string that has been passed to a program from a script as an argument, making it easy to extract each record. This example below shows how to do this:

Dim inputvalues As String
'// The input values will come from the script parameters
inputvalues = "citylimit.shp 2;faultzn.shp 3;flood.shp 1"

pValueTable.LoadFromString inputvalues

Mapping Fields
A common geoprocessing task is to merge many datasets into a new or existing dataset to create a single dataset covering a larger area or a table containing a greater number of records. Often, the attributes, or fields, are the same for all the inputs that are used in a merge or append operation, but sometimes they do not match and the relationships between fields of different names and types have to be mapped. The Merge tool in the Management toolbox facilitates this mapping of relationships so data is placed in the desired output fields with the correct values.

The IGPFieldMap object provides a field definition and a list of input fields from a set of tables or feature classes that provide its values. The IGPFieldMapping object is a collection of IGPFieldMap objects, and it is used as the parameter value for tools that perform field mapping, such as Merge, Append, Feature Class To Feature Class and Table to Table. The easiest way to work with these objects is to first create an IGPFieldMapping object, then initialize its IGPFieldMap objects by adding the input
feature classes or tables that are to be combined. Once all inputs are provided, the
IGPFieldMapping object will contain one IGPFieldMap object, or output field, for each
unique field name from all the inputs. This list may be modified by adding new fields,
altering the properties and/or contents of an output field, or removing any unwanted
output fields.

The properties of the IGPFieldMap object include the start and end position of an input
text value, so a new output value may be created using a slice of an input value. If an
IGPFieldMap object contains multiple input fields from the same table or feature class,
each record's values will be merged using the MergeRule property. This is a convenient
way to join values, such as a street name that is held in one field and a street type that is
held in another, for example, Eureka and Street. The Delimiter property of an
IGPFieldMap is used if the MergeRule value of Join is specified. Any set of characters,
such as a space, may be used as a delimiter. For the above example, this would create a
value of Eureka Street.

In the following example, a table containing U.S. census data will be converted to a new
table. One of the input attributes found in the input table is a text field, STFID. This 15-
digit value is a unique identifier for all census blocks for the United States. The value
may be broken into four components. The first two digits provide the state code, the next
three indicate the county, the following six identify the census tract, and the last four
identify the census block. The value 360899912001006 represents the census block
(1006) containing the State University of New York at Potsdam in upstate New York
(36), within census tract 991200 of the county of St. Lawrence (089). This sample will
convert this table and also create a new field, TRACTID, because the input data only has
the STFID attribute. To do this, the GPFielMapping object is initialized using the
AddTable method to enter the input table. Then the default IGPFieldMapping object is
modified by creating a new IGPFieldMap object, populating its properties, and adding
them to the IGPFieldMapping object.

Sub MappingFields()
On Error GoTo EH:

    '//initialize license
    Dim aoi As I AoInitialize
    Set aoi = New AoInitialize
    aoi.Initialize esriLicenseProductCodeArcInfo

    '// create the GeoProcessor
    Dim GP As IGeoProcessor
    Set GP = New GeoProcessor
    GP.OverwriteOutput = True

    Dim strTableA As String
    strTableA = "C:\Data\citiblocks.gdb\census"

    Dim pDataElement As IDataElement
    Dim pDETableA As IDETable
Dim pGPU As IGPUtilities
Set pGPU = New GPUtilities

Set pDataElement = pGPU.MakeDataElement(strTableA, Nothing, Nothing)
Set pDETableA = pDataElement

Dim pArray As IArray
Set pArray = New esriSystem.Array
pArray.Add pDETableA

Dim pFieldMapping As IGPFieldMapping
Set pFieldMapping = New GPFieldMapping

'//initialize the field mapping
pFieldMapping.Initialize pArray, Nothing

'//create a new output field
Dim TrackIDField As IFieldEdit
Set TrackIDField = New Field
TrackIDField.Name = "TRACTID"
TrackIDField.Type = esriFieldType.esriFieldTypeString
TrackIDField.Length = 20

'//create a new FieldMap
Dim TrackID As IGPFieldMap
Set TrackID = New GPFieldMap
Set TrackID.OutputField = TrackIDField

'//find field map STFID containing the input field STFID
Dim FieldMap_Index As Long, Field_Index As Long
FieldMap_Index = pFieldMapping.FindFieldMap("STFID")

Dim STFID_FieldMap As IGPFieldMap
Set STFID_FieldMap = pFieldMapping.GetFieldMap(FieldMap_Index)

Field_Index = STFID_FieldMap.FindInputField(pDETableA, "STFID")

Dim inputField As IField
Set inputField = STFID_FieldMap.GetField(Field_Index)

TrackID.AddInputField pDETableA, inputField, 5, 10

'//add the new field map to the field mapping
pFieldMapping.AddFieldMap TrackID

'//execute Table to Table tool using the FieldMapping
Dim parameters As IVariantArray
Set parameters = New varArray
parameters.Add (strTableA)
parameters.Add ("C:\Data\citiblocks.gdb")
parameters.Add "census2"
parameters.Add ""
parameters.Add pFieldMapping
GP.Execute "TableToTable_conversion", parameters, Nothing
MsgBox GP.GetMessages(0)
Exit Sub
EH:
   Debug.Print GP.GetMessages(2)

End Sub

**Geoprocessing Messages**

**About Geoprocessing Messages**

During execution of a tool, messages are relayed back to the Geoprocessor. These messages include such information as when the operation started, what parameter values are being used, the operation's progress, and warnings of potential problems or errors. All communication between a tool, the Geoprocessor, and the user is conducted via this messaging. Geoprocessing programs are frequently run in an automated fashion without user interaction. Since geoprocessing tools do not assume there is direct interaction with a user, they never use message boxes or other types of dialog boxes during their execution.

A tool message will be classified as an informative message, a warning, or an error. A message's type is indicated by its severity property, which is a numeric value. An informative message may be used to indicate any event that does not reflect a problem or possible error. Typical informative messages indicate a tool's progress, what time a tool started or completed, output data characteristics, or tool results. A severity value of 0 is used for informative messages, since they require no action from a user or program. Warning messages are generated when a tool experiences a situation that may cause a problem during its execution or when the result may not be what the user expects. A user or program may choose to take action when a warning is returned, such as canceling the tool's execution or making another parameter choice. Defining a coordinate system for a dataset that already has a coordinate system defined, for example, will generate a warning. The tool will still execute, but it may not create the desired result if the user did not intend to alter the existing coordinate system. A severity value of 1 is given to warning messages.

Error messages indicate a critical event that will prevent a tool from executing. Errors are generated when one or more parameters have invalid values or when a critical execution process or routine has failed. Paths to data that does not exist, an invalid keyword, or corrupted data are examples of situations that will cause an error. Errors have a severity value of 2, indicating an action is required by the program or user.

Most programming languages have built-in error handling, allowing programs to continue execution in a logical fashion when an error occurs. When an error message from a tool is returned to the Geoprocessor, it generates a system error, which may be caught by a program’s error handling routine. If a program does not have an error handling routine, it will fail immediately, which decreases its robustness. Use error handling routines to manage errors and improve a program's usability.
Messages from the last tool the Geoprocessor executed are maintained as a list of messages. A GPMessage class contains a message’s text and its severity value. Using the GPMessage class, you may access the message list to retrieve, add, or clear messages.

### Getting Returned Messages

When tools are executed in a script, you may want to access the resulting messages if an error or warning occurs or simply show the progress of an operation. The returned messages may be filtered to only those with a certain severity using the severity option. Here is an example that returns all the tool’s error messages:

```vbscript
Dim sMsgs As String
'// get the messages with severity 2
sMsgs = GP.GetMessages(2)
Debug.Print sMsgs
```

Individual messages can be retrieved using the GetMessage method. This method has one parameter, which is the index of the message in the geoprocessor's message list or array. The MessageCount property maintains the number of messages in the geoprocessor's message array. Here is an example that returns all the messages:

```vbscript
'// Execute Union
GP.Execute "Union_analysis", parameters, Nothing
If GP.MessageCount > 0 Then
    For count = 0 To count <= GP.MessageCount - 1
        Debug.Print GP.GetMessage( count )
    Next
End If
```

### Setting output messages

When a script tool is executed, messages often need to be returned to the user, especially when problems arise. To support the full integration of program executables as tools, the GeoProcessor has AddMessage, AddWarning and AddError methods for adding messages, which are then available to the user. Messages added to the Geoprocessor are immediately returned to the application or program executing the tool. These methods add a new GPMessage to the collection. The GPMessage can be one of the following types: Informative, Warning, and Error.

Once an error message is added, the Geoprocessor immediately generates a system error that halts the execution of the tool. The next example, a simple script tool, buffers the point features from an input feature class. It adds two messages first and then, depending on the success or failure of the execution buffer adds more message, warning or error. Error handling is used to catch any problems and return messages; otherwise, messages of success are returned during execution.

```vbscript
Dim GP As IGeoProcessor
Set GP = New GeoProcessor
```
GP.OverwriteOutput = True

Dim strFC As String, strOut As String
strFC = "C:\data\buffer\nfld_wells40.shp"
strOut = "C:\data\buffer\nfldwells_buffer.shp"

GP.AddMessage "Input : " & strOut
GP.AddMessage "Output : " & strFC

Dim parameters As IVariantArray
Set parameters = New VarArray
parameters.Add strFC
parameters.Add strOut
parameters.Add "20000 Meters"

GP.Execute "Buffer_Analysis", parameters, Nothing

If GP.MaxSeverity = 0 Then
    GP.AddMessage "MyBuffer Tool executed successfully."
ElseIf GP.MaxSeverity = 1 Then
    GP.AddWarning "Tool executed with warning: " & GP.GetMessages(1)
Else
    GoTo ErrorHandler
End If

ErrorHandler:
If GP.MaxSeverity = 2 Then
    GP.AddError "Error occurred: " & GP.GetMessages(2)
End If

There are times when you may want a script to return messages from a tool it has
executed. Using an index parameter, the AddReturnMessage method will return a
message from the geoprocessor's message array. The example below shows how to return
all of a tool's messages:

GP.Execute "Buffer_Analysis", parameters, Nothing

Dim i As Integer
For i = 0 To GP.MessageCount - 1
    GP.AddReturnMessage (i)
Next

About Script Tools

Many programs are written for specific datasets and scenarios. One of the advantages of
programming is its flexibility to perform additional operations not provided by ESRI and
extend the ArcGIS framework. Some programs perform a generic operation that may be
used repeatedly because they use arguments to change tool parameters and behavior.
These programs may rarely change and are used by many people. ArcGIS Desktop users
are able to use the geoprocessing framework to create tools where program executables
are used as the source. These types of tools are called Script Tools.
Using Script Tool Parameters

Geoprocessing tools typically have parameters. Parameters, or arguments, make programs more generic and flexible, so they can be used to vary input data, set other tool parameters used in the program, or control the program's processing logic.

When a program is added to a toolbox as a script tool, a number of parameters may be defined. These parameters correspond to a program's parameters, representing input and output values.

Typically, programs only have input values, since they are usually run in an independent fashion. Script tools must define their outputs so tools work as expected in models built with ModelBuilder or on the command line in the geoprocessing window. Models need an output parameter so they can be used as input for other tools, while tools on the command line allow users to specify the name and location of a tool's output. System tools, built with ArcObjects, populate the properties of output variables in a model to aid subsequently connected tools. These properties are not required for building models but aid in their construction. System tools do this with a validation routine that is called each time a parameter value is changed for a tool. Script tools do not have a validation routine, just one for execution, which is the script itself. The geoprocessing framework does some validation for script tools automatically, such as ensuring a parameter value matches what is expected for a parameter, including a number for a numeric parameter and a path to a feature class for a feature class parameter. By defining dependencies between parameters, certain behavior may also be built into a tool.

For example, a dependency between a feature class parameter and a field parameter in which the field parameter is dependent on the feature class means the field list will automatically be populated on the tool when the feature class is given a valid value. Another program may not actually have a new output because it alters a dataset specified as an input parameter. In this case, the script tool would still declare an output parameter, with the type set to derived with a dependency on the input dataset. The script tool will automatically set the output parameter's value to be the same as the input dataset when used in a model so the tool may be used in a work flow.

>> Getting input parameter values

Programming languages typically provide a mechanism for accessing arguments passed to the program from the caller. In VB, for example, a variable of type Variant holds the arguments passed to a method. A program must use these mechanisms if it is not the source of a script tool, as shown in the example below:

```vbnet
'// set the workspace environment
GP.SetEnvironmentValue "workspace", GP.GetParameterValue(0)
```

The following is an example of how to call the program executable in the example above, specifying an input workspace:

```bash
ClipData.exe "C:\soils\Newfoundland"
```
If a program is the source of a script tool, it may use the Geoprocessor to access the input parameter values. Here is an example of getting parameters from a user. This script tool gets the input and output parameters from the user and runs the Select tool:

```vbscript
Dim gp As IGeoProcessor
Set gp = New GeoProcessor

gp.SetEnvironmentValue "workspace", "C:\data\select"
gp.OverwriteOutput = True

'//declare variables for input parameters
Dim in_features As String
Dim out_features As String

'//Get the parameters from the user
in_features = gp.GetParameterValue(0)
out_features = gp.GetParameterValue(0)

'//declare and set a variant array to hold the parameters
Dim parameters As IVariantArray
Set parameters = New VarArray

'//populate the variant array with the parameters
parameters.Add (in_features)
parameters.Add (out_features)

gp.Execute "Select", parameters, Nothing
```

**>> Setting output parameter values**

Output parameter values may not be known until the program is executed. The program must evaluate or calculate an output value based on its input, so script tools must have a way to specify output values after execution so the values may be used in a model workflow.

The SetParameterValue method will set the value of an output parameter using either an object, such as a value table, or a text string or Boolean value. An index value is required to indicate which parameter will be updated.

```vbscript
'// Initialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = new Geoprocessor

'// Check the feature classes and set the output parameters
GP.SetEnvironmentValue "workspace", GP.GetParameterValue(0)

If GP.ListFeatureclasses("*","POLYGON").Next Then
    GP.AddMessage "Feature type is POLYGON"
    GP.SetParameterValue 1, True
    GP.SetParameterValue 2, False
Else If GP.ListFeatureclasses("*","ARC").Next Then
    GP.AddMessage "Feature type is LINE"
    GP.SetParameterValue 1, False
```
GP.SetParameterValue 2, True
Else
    GP.AddMessage "Coverage has neither polygons nor lines"
    GP.SetParameterValue 1, False
    GP.SetParameterValue 2, False
End If

Data Properties and Access
Geoprocessing tools work with all types of data, such as geodatabase feature classes, shapefiles, rasters, tables, topologies, and networks. Each piece of data has properties that may be used to control the flow of a program or the parameters of a tool. For example, the output feature type of an intersect operation is dependent on the type of data being intersected. When the Intersect tool is run within a program on a list of input datasets, it must be able to determine the data types used so the correct output type can be set. Using the Geoprocessor's GetDataElement method, a dataset's properties may be determined and used to make decisions.

GetDataElement returns a pointer to IDataElement interface that can be used to describe the given value. All data element classes support the IDataElement interface. Click here, for a list of all the data element classes that implement IDataElement.

Dim pType As IGPDataType
Set pType = Nothing

Dim pDataElement As IDataElement
Set pDataElement = gp.GetDataElement("C:\GP\fgdb.gdb\cities\Redlands", pType)

Dim defc As IDEFeatureClass
Set defc = pDataElement

If (defc.ShapeType = esriGeometryPolygon) Then
    MsgBox "Shapetype is POLYGON"
End If

Checking for Existence
Programs often use paths to data, which may be problematic if the data being referenced does not exist. Data may be deleted or moved between executions of a script, which will cause errors if the path is used as a geoprocessing tool parameter. If there is a possibility of a referenced dataset not existing during a program's execution, the Geoprocessor's Exists method should be used.

The function simply returns a Boolean value for the existence of a dataset or object at the time of execution. Objects, such as a cursor, spatial reference, or any other object managed by the Geoprocessor, may also be used as input to Exists. Exists will work with any type of data available in ArcCatalog or with any system file or folder. An ArcCatalog path must be used for this and any other method of the Geoprocessor when referring to
GIS data. If the data resides in an enterprise geodatabase, the name must be fully qualified.

The default behavior for all tools is to not overwrite any output that already exists. This behavior may be changed by changing the OverwriteOutput setting to true. Attempting to overwrite when the OverwriteOutput option is false will cause a tool to fail.

```GP.OverwriteOutput = True```

### Working with Geodatabases

Geodatabases are relational databases that contain geographic information. Geodatabases contain feature classes and tables. Feature classes can be organized into a feature dataset; they can also exist independently in the geodatabase.

Feature classes store geographic features represented as points, lines, or polygons and their attributes; they can also store annotation and dimensions. All feature classes in a feature dataset share the same coordinate system. Tables may contain additional attributes for a feature class or geographic information, such as addresses or x,y,z coordinates.

The geodatabase model defines a generic model for geographic information. This generic model can be used to define and work with a wide variety of different use - or application-specific models. By defining and implementing a wide variety of behavior on a generic geographic model, a robust platform is provided for the definition of a variety of user data models.

The geodatabase supports a model of topologically integrated feature classes similar to the coverage model. However, it extends the coverage model with support for complex networks, topologies, relationships among feature classes, and other object-oriented features. The ESRI ArcGIS Desktop applications (ArcMap, ArcCatalog, and ArcGlobe) work with geodatabases as well as coverages and shapefiles. Learn how to build and edit data in a geodatabase

Successfully implementing a multi-user GIS system with ArcInfo and ArcSDE, starts with a good data model design and database tuning. How the data is stored in the database, the applications that access it, and the client/server hardware configurations are all key factors to a successful multi-user GIS system.

A critical part of a well-performing geodatabase is the tuning of the database management system (DBMS) in which it is stored. This tuning is not required for personal geodatabases; however, it is critical for ArcSDE geodatabases. For more information on tuning your database for ArcSDE and the geodatabase, see the Configuration and Tuning Guide for <DBMS> PDF file.

Designing a geodatabase is a critical process that requires planning and revision until you reach a design that meets your requirements. Once you have a design, you can create the geodatabase and its schema using geoprocessing tools. There are tools for creating,
modifying, and analyzing your geodatabase schema, such as Create Feature Class, Compress, and Add Subtype.

Note that personal geodatabases are designed for single use, utilizing MS Access as the database. Multi-user geodatabases use RDBMS packages, such as Oracle, SQL Server, or IBM DB2. Generic geoprocessing programs should be written to work with either type of geodatabase using the various validation and qualification Geoprocessor methods.

Validating Table and Field Names
Geodatabases use various relational database management systems to maintain the many tables that comprise a geodatabase. All tables in a geodatabase must have a unique name, so a mechanism for checking if a table name is unique is essential when creating data in a geodatabase. The default behavior for geoprocessing tools is to overwrite output that already exists, so there is potential for accidentally overwriting data if the script doesn't ensure that the new output name is unique. Using the ValidateTableName method, a program can determine if a specific name is valid and unique to a specific workspace.

Specifying the workspace as a parameter allows the Geoprocessor to check all the existing table names and determine if there are naming restrictions imposed by the output workspace. If the output workspace is an RDBMS, it may have reserved words that may not be used as a table name. It may also have invalid characters that are not to be used in a table or field name. ValidateTableName will return a string representing a valid table name that may be the same as the input name if the input name is valid. The example below shows how to validate the name of each feature class before copying them to a SDE database.

```vbscript
' // Initialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = New GeoProcessor

' // Set the input workspace environment and list all shapefiles.
GP.SetEnvironmentValue "workspace", "C:\data\shapefiles"
Dim shapefiles As IGeoprocessor
Dim count As Integer
Dim parameters As IVariantArray
Set parameters = New varArray
shapefiles = GP.ListFeatureClasses("*", "", "")
shapefiles.Reset

' // Output workspace
Dim outputWorkspace As String
outputWorkspace = "Database Connections\pasta1.sde"

' // Get first feature class name
Dim shapefile As String
shapefile = shapefiles.Next

Dim outfc As String
Dim count As Integer
Dim parameters As IVariantArray
Set parameters = New varArray
```
While (shapefile <> "")
    '// Validate the output name so it is unique and not empty
    outfc = GP.ValidateTableName(shapefile, outputWorkspace)

    '// Set input and output feature classes
    parameters.Add shapefile
    parameters.Add outputWorkspace & "\" & outfc

    '// Execute CopyFeatures
    GP.Execute "CopyFeatures_management", parameters, Nothing

    If (GP.MessageCount > 0) Then
        For count = 0 To GP.MessageCount - 1
            Debug.Print GP.GetMessage(count)
        Next
        '// Get next feature class
        shapefile = shapefiles.Next
    End If
Wend

>> Validating field names

Each database may have naming restrictions for field names in a table. Objects such as feature classes or relationship classes are stored as tables in an RDBMS, so these restrictions affect more than just standalone tables. These restrictions may or may not be common between various database systems, so programs should check all new field names to ensure that a tool does not fail during execution.

The example below will ensure that a field will be added no matter what the input name is, using the ValidateFieldName method:

    '// Intialize the Geoprocessor
    Dim GP As IGeoProcessor
    Set GP = New GeoProcessor

    '// Get the input feature class
    Dim inputFeatureClass As String
    inputFeatureClass = "C:\data\mexico.gdb\land\lakes"

    '// Get the field name
    Dim fieldname As String
    fieldname = "Well-Depth"      '// invalid for file geodatabase.

    '// Validate field name for entry into a file geodatabase feature class
    fieldname = GP.ValidateFieldName(fieldname, "C:\data\mexico.gdb\land")

    '// create VarArray and populate it with parameters
    Dim parameters As IVariantArray
    Set parameters = New varArray
    parameters.Add inputFeatureClass
    parameters.Add fieldname
    parameters.Add "TEXT"

    GP.Execute "AddField_management", parameters, Nothing
Whenever a program is updating a dataset, such as a feature class or table, be careful to avoid situations in which the dataset is locked. Microsoft Access always locks a database for update operations when a table is accessed, so if you have a personal geodatabase open in ArcCatalog, a program will not be able to update any of the geodatabase's contents until it is deselected and the folder is refreshed or ArcCatalog is closed.

**Qualifying table and field names**

Geodatabases that reside in an RDBMS, such as Oracle, SQL Server, or IBM DB2, use standard database naming conventions for identifying tables for specific users. An RDBMS may contain thousands of tables from many different users; therefore, users of an enterprise geodatabase must understand how to correctly qualify the name of an object so the correct feature class, relationship class, feature dataset, or table is used during an operation. If an unqualified name is specified as input to a tool, the Geoprocessor will qualify it automatically using the currently connected user name, which is specified as a property of the connected workspace. If a program needs to access data from a number of users, it should qualify the name of the table using the Geoprocessor's QualifyTableName method so the syntax of the qualified name is correct.

It requires the name of a table, the username and the path to a geodatabase. Output parameter values do not need to be qualified because a tool's output is always created by the connected user of the workspace. The example below shows how a script qualifies table names so it can use tables from a number of users:

```vbscript
'// Initialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = New GeoProcessor

'// Set the input workspace environment and list all shapefiles.
GP.SetEnvironmentValue "workspace", "Database Connections/Calgary.sde/transportation"
Dim shapefiles As IGPEnumList
Set shapefiles = GP.ListFeatureClasses("*", ",", ",")

'// Get first feature class name
Dim shapefile As String
shapefile = shapefiles.Next

Dim outfc As String
Dim outputWorkspace As String
Dim clipfc As String

outputWorkspace = "Database Connections/Calgary.sde/clip"

Dim parameters As IVariantArray
Set parameters = New varArray
```
While (shapefile <> "")
    outfc = GP.ValidateTableName(shapefile, outputWorkspace)
    '// Qualify the clip feature class name
    clipfc = GP.QualifyTableName(shapefile, outputWorkspace)
    parameters.Add shapefile
    parameters.Add clipfc
    parameters.Add outfc
    GP.Execute "Clip_analysis", parameters, Nothing
    shapefile = shapefiles.Next
Wend

SQL queries may contain the names of many fields from two or more tables. It is not uncommon for the same field name to exist in separate tables, especially when working with foreign and primary keys. Qualified field names must also be used when a table contains certain field names that are also used by ArcSDE. To resolve the ambiguity between duplicate names, the field name must be qualified with the table or view name. The QualifyFieldName method allows a database independent mechanism for creating fully qualified field names, using the table and field name as input. The example below ensures that a program will always create a correct SQL statement, regardless of the underlying database type:

'// Intialize the Geoprocessor
Dim GP As IGeoProcessor
Set GP = New GeoProcessor

'// Set the input workspace environment
Dim sde As String
sde = "Database Connections/vampire.sde/LegalFabric"
GP.SetEnvironmentValue "workspace", sde

Dim parameters As IVariantArray
Set parameters = New varArray

'// create the layer for selection
parameters.Add "taxparcels"
parameters.Add "parcels"
GP.Execute "MakeFeatureLayer_management", parameters, Nothing
parameters.RemoveAll

'// join the parcel updates table using parcel_id field
parameters.Add "parcels"
parameters.Add "parcel_id"
parameters.Add "parcel_updates"
parameters.Add "parcel_id"
GP.Execute "AddJoin_management", parameters, Nothing
parameters.RemoveAll

'// qualify the identically named fields
Dim id As String
Dim update_id As String
id = GP.QualifyFieldName("parcels", "parcel_id")
update_id = GP.QualifyFieldName("parcel_updates", "parcel_id")
'//select the parcels using the qualified names
parameters.Add "parcels"
parameters.Add "New_Selection"
parameters.Add id & " = " & update_id  'not clear
GP.Execute "SelectLayerByAttribute_management", parameters, Nothing
parameters.RemoveAll

'//calculate the update flag to be true for the selected parcels
parameters.Add "parcels"
parameters.Add "updateflag"
parameters.Add "True"
GP.Execute "CalculateField_management", parameters, Nothing

Tips:

- ArcGIS desktop applications, such as ArcMap and ArcCatalog, always present fully qualified feature class, table, and field names. If you are unsure of the owner of a table or which tables are accessible, use ArcCatalog to view the contents of the geodatabase.
- Field names, such as Area, Entity, and Len are used by ArcSDE system tables, and they require programs to fully qualify fields with these names. Refer to the ArcSDE Administration Guide for more information on ArcSDE field names.

Parsing table and field names

Programs should use the Geoprocessor's ParseTableName and ParseFieldName methods to split the fully qualified names for a dataset or for a column in a table into its components (database, owner, table, and column). Programs that need to be RDBMS independent should not assume that '.' is the delimiter used to separate the components of a fully qualified dataset name. ArcGIS applications always use fully qualified names, so if your program is being used as the source of a script tool, any feature class name, for example, will need to be parsed if the program needs to determine the name of the feature class without the user and database names. In the example below, the program is checking the input feature class for specific user names. The program could use the parsing functions of VB, C# or VB.NET to split the qualified name, but it would then assume a specific syntax, which may change if another database type is used. ParseTableName returns a single string with the database name, owner name, and table name separated by commas. ParseFieldName returns the table name and the field name, also separated by commas. A program can then reliably parse the returned string, since these methods always return the same formatted string. A string in VB has a number of native methods for manipulation. In this example, the split method is used to create a variant array, using the comma as the delimiter.

Dim GP As IGeoProcessor
Set GP = New GeoProcessor
GP.SetEnvironmentValue "workspace", "Database Connections/vampire.sde"
Dim sFCs As IGPEnumList
Set sFCs = GP.ListFeatureClasses("*", ",", ",")
Dim FC As String
FC = sFCs.Next

Dim fullname As String
fullname = GP.ParseTableName(FC, "Database Connections/vampire.sde")

Dim namelist As Variant
namelist = Split(fullname, ",")

Dim owner As String
Dim database As String
Dim featureclass As String

database = namelist (0)
owner = namelist (1)
featureclass = namelist (2)