



Hitchhiker's Guide to Geoprocessing

Jason Pardy & Corey Tucker

Workshop Outline



- Geoprocessing Overview
- Developing Tools with Models and Scripts/Programs
- Executing Tools in VB6, .NET, Scripting, and C++
- Developing Tools with ArcObjects
- A Look Ahead at ArcGIS 9.2

Who we think you are:



- You add value to ArcGIS community
 - Through your understanding of GIS and expertise in a particular domain (wastewater, urban planning, etc.)
- You develop new functions for ArcGIS
 - VB, C++, C#, .NET, JAVA, Scripts
 - Extend behavior of the system (i.e., new feature class behavior)
 - Extend functionality of the system (new toolbars and tools)

...and why you should know about geoprocessing

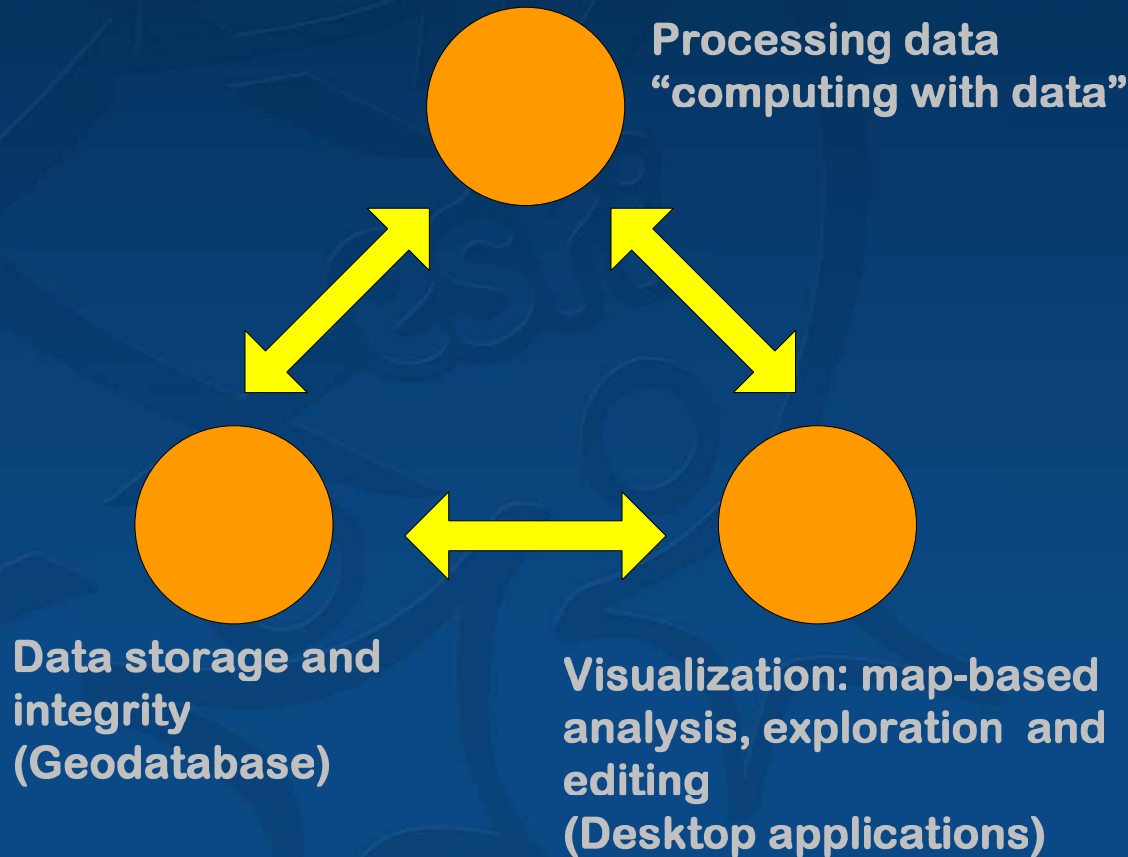


- Empowers GIS professionals to implement workflows
 - Reduces barriers between GIS professionals and software developers
- A complete platform for delivering solutions
 - Universal capability that can be used and deployed by all GIS users to automate their work, build repeatable and well-defined methods and procedures, and to model important geographic processes.
- *Significant* reduction of your development cycle

Geoprocessing is...



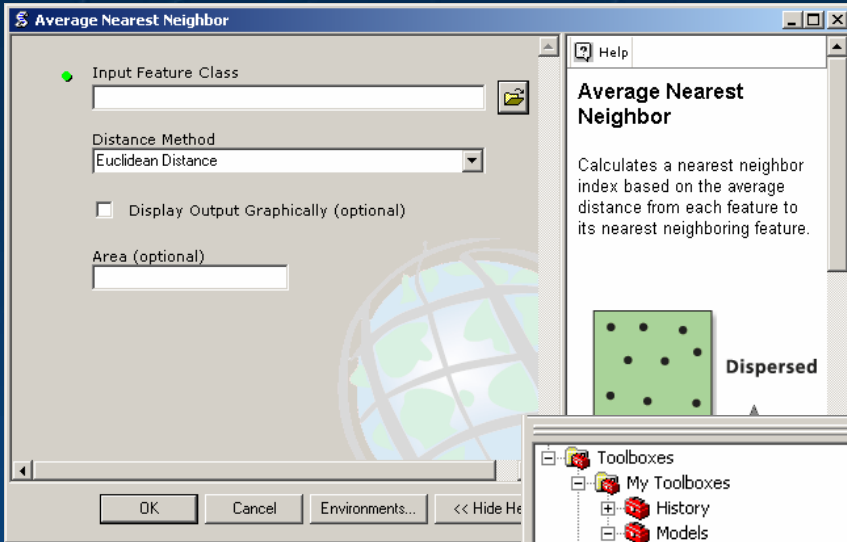
...one of the 3 critical components of a GIS



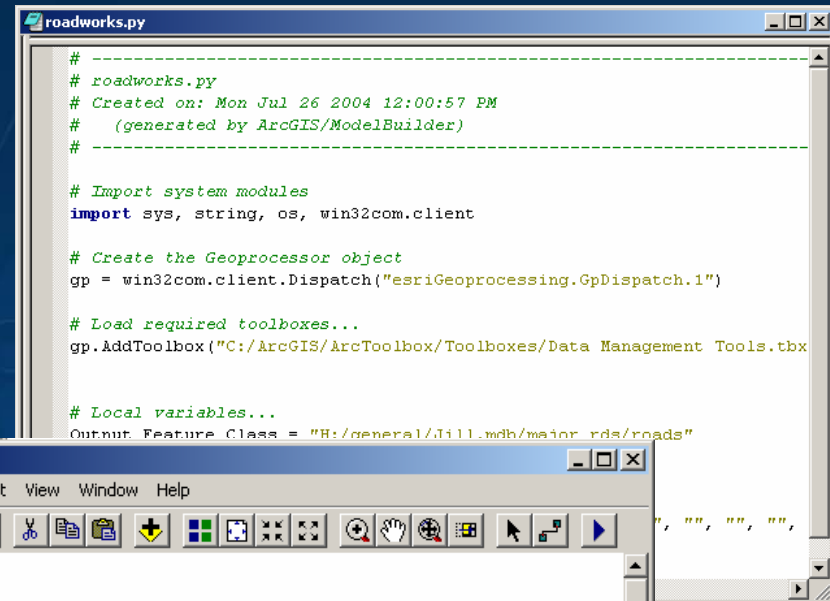
Geoprocessing Framework



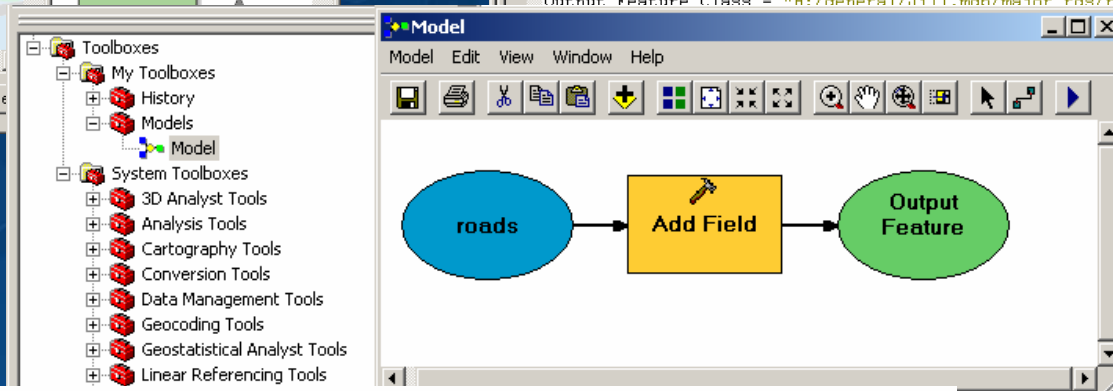
As a dialog...



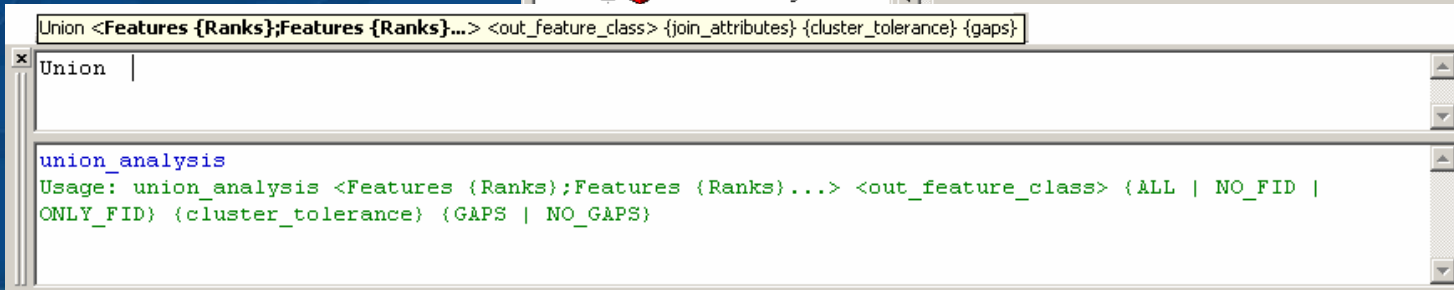
As a script...



As a model...



As a command...



ArcToolbox



- Presents toolboxes and tools in a functionally ordered tree
- Dockable window in all ArcGIS applications – no longer a separate application
- Custom toolboxes can easily be added to access custom tools (models & scripts)

Geoprocessing Tools

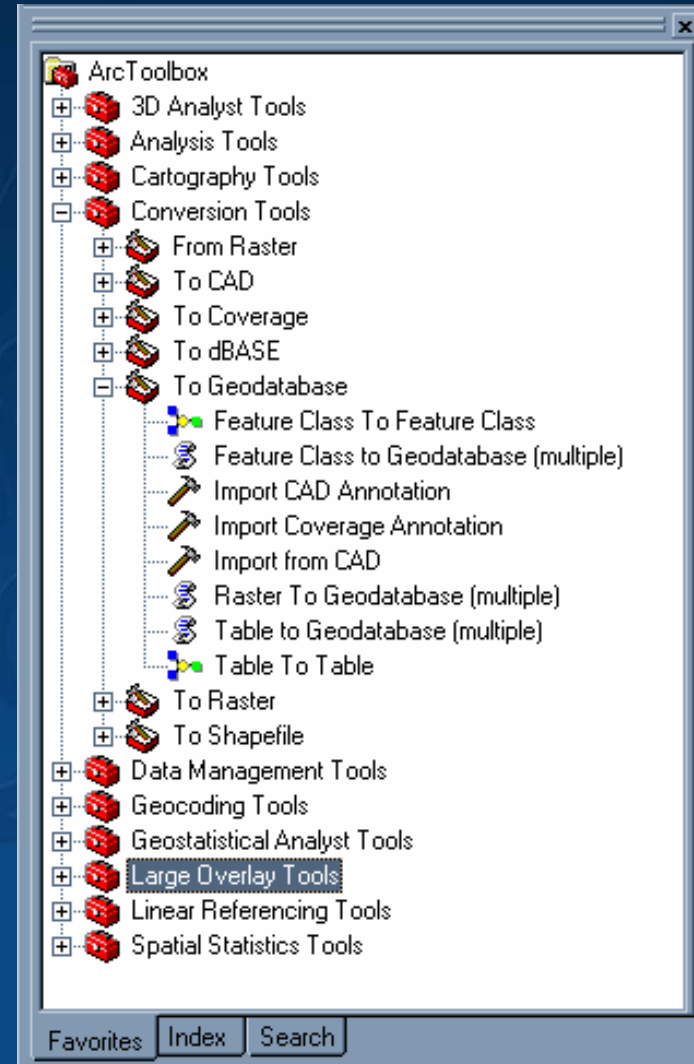


Tools (450+ tools)

- ArcGIS tools
- Models
- Scripts

Toolboxes

- 3D Analyst tools - 45
- Analysis tools - 16
- Cartography tools - 3
- Conversion tools - 25
- Data management tools - 118
- Geocoding tools - 7
- Geostatistical Analyst tools - 1
- Linear Referencing tools - 7
- Spatial Analyst tools - 160
- Spatial Statistics tools - 18



Geoprocessing Tools



- Tools work with:
 - Personal geodatabase feature classes and tables
 - SDE feature classes and tables
 - Shapefiles
 - Coverages
 - GRIDs
 - Rasters
 - Plug-in datasources (SDC, VPF, FME, etc)

Geoprocessing Tools



- Tools support metadata and have built in documentation
 - Users can create documentation for custom tools using a new geoprocessing tool documentation editor
 - Custom tools use the same documentation style as tools created by ESRI

Creating Tools



- Model Tool
 - FeatureClassToFeatureClass Tool
- Script Tool
 - FeatureClassToGeodatabase Tool
 - Spatial Statistics Tools
- Function Tool (System Tool)
 - Most Geoprocessing Tools are Function Tools
- Custom Tool
 - i.e. Data Interoperability Extension

ModelBuilder



Model Tools

The screenshot displays the ArcMap interface with two ModelBuilder workflow windows open. The top window, titled "Find Impacted Habitat", shows a workflow starting with two input polygons: "Input Proposed Road" and "Input Vegetation". "Input Proposed Road" is processed by a "Buffer" tool to create "Buffer Zones". "Input Vegetation" is processed by a "Find Potential" tool to create "Output Habitat Locations". Both "Buffer Zones" and "Output Habitat Locations" are then used as inputs for a "Clip" tool, resulting in the final "Output Impacted Habitat".

The bottom window, titled "Find Potential Habitat 1", shows a more complex workflow. It starts with three input polygons: "roads_100", "roads_200", and "roads_300". Each is processed by a "Buffer" tool to create "Roads Buffer", "Roads Buffer", and "Roads Buffer" respectively. These three buffers are combined in a "Merge" tool. Simultaneously, "roads_100" and "roads_200" are processed by "Select" tools to create "Roads Selection" and "Roads Selection". These two selections are combined in a "Merge" tool. The output of the "Merge" tool (from the three buffers) and the output of the second "Merge" tool (from the two selections) are then processed by a "Cross" tool. The output of the "Cross" tool is processed by a "Select" tool, followed by a "Dissolve" tool, a "Dissolve" tool, a "Merge" tool, a "Merge" tool, and finally a "Select" tool to produce the "Output Habitat Locations".

The ArcMap interface includes a tool palette on the left with categories like Extract, Overlay, Proximity, Statistics, Cartography Tools, Conversion Tools, Data Management Tools, Geocoding Tools, Geostatistical Analyst Tools, Linear Referencing Tools, My Habitat Potential Tools, Find Impacted Habitat, Find Potential Habitat 1, and My_Analysis_Tools. The status bar at the bottom shows the scale as 6250293.53 1870292.04 Feet and the time as 9:10 PM.

Script Tools



- Two basic language categories: system & scripting
 - System languages:
 - C++, .Net, Java.
 - Create applications from scratch, using low-level primitives and raw resources of the computer
 - Scripting languages:
 - Python, VBScript, Java script, Perl
 - Useful for gluing applications together, using higher level functions of the computer, masking the nuts and bolts
 - Easy to learn and use—a basic understanding of programming is all that's needed to be productive.
- Scripts are analogous to models, in that they create new tools.
 - ModelBuilder: visual programming language ModelBuilder
 - Scripts: text-based language and text editors.

Developing Script Tools



- Scripts and executables may be added to toolboxes as a script tool
 - Amls, python scripts, and program executables (exe) can be used as source to script tools
- Once added as a tool, the script behaves like other tools, it can be run using a dialog, in ModelBuilder, at the command line, or in another script/program.

Scripts and Executables as Tools



- Script tool source is pathname to script/program file.
- Define input and output parameters

Map Color Properties

General | Source | Parameters | Help

Display Name	Data Type
@ Input Features	Feature Layer
Color Field	Field
Output Features	Feature Layer
	Feature Layer

Click any parameter above to see its properties below.

Parameter Properties

Property	Value
Type	Required
Direction	Input
MultiValue	No
Default	
Environment	
Domain	
Dependency	

To add a new parameter, type the name into an empty row in the name column, click in the Data Type column to choose a data type, then edit the Parameter Properties.

OK Cancel Apply

Running Geoprocessing Tools



- Tools have a fixed set of parameters required for execution.
- Parameter values must be correctly set so it can execute when the program is run.
- Parameters are specified either as strings or objects (i.e. IFeatureClass, ISpatialReference).
- Tool with the same name can exist in different toolboxes. Use toolbox alias to avoid tool name conflicts.
 - E.g. Clip_analysis and Clip_management

Running Tools in VBA, VB6, and VB.NET



- Tools are executed using the GPDDispatch object
- Tools are accessed as methods on the GPDDispatch object
- Environments and settings are accessed as properties on the GPDDispatch object

Create a shapefile: VB & ArcObjects



```
Public Sub CreateShapefile()  
    Const strFolder As String = "D:\DATA"  
    Const strName As String = "MyShapeFile" ' Dont include .shp extension  
    Const strShapeFieldName As String = "Shape"  
  
    ' Open the folder to contain the shapefile as a workspace  
    Dim pFWS As IFeatureWorkspace  
    Dim pWorkspaceFactory As IWorkspaceFactory  
    Set pWorkspaceFactory = New ShapefileWorkspaceFactory  
    Set pFWS = pWorkspaceFactory.OpenFromFile(strFolder, 0)  
  
    ' Set up a simple fields collection  
    Dim pFields As IFields  
    Dim pFieldsEdit As IFieldsEdit  
    Set pFields = New esriGeoDatabase.Fields  
    Set pFieldsEdit = pFields  
  
    Dim pField As IField  
    Dim pFieldEdit As IFieldEdit  
  
    ' Make the shape field  
    ' it will need a geometry definition, with a spatial reference  
    Set pField = New esriGeoDatabase.Field  
    Set pFieldEdit = pField  
    pFieldEdit.Name = strShapeFieldName  
    pFieldEdit.Type = esriFieldTypeGeometry
```

Wait! There's more!



```
Dim pGeomDef As IGeometryDef
Dim pGeomDefEdit As IGeometryDefEdit
Set pGeomDef = New GeometryDef
Set pGeomDefEdit = pGeomDef
With pGeomDefEdit
    .GeometryType = esriGeometryPolygon
    Set .SpatialReference = New UnknownCoordinateSystem
End With
Set pFieldEdit.GeometryDef = pGeomDef
pFieldsEdit.AddField pField

' Add another miscellaneous text field
Set pField = New esriGeoDatabase.Field
Set pFieldEdit = pField
With pFieldEdit
    .Length = 30
    .Name = "MiscText"
    .Type = esriFieldTypeString
End With
pFieldsEdit.AddField pField

' Create the shapefile
' (some parameters apply to geodatabase options and can be defaulted as Nothing)
Dim pFeatClass As IFeatureClass
Set pFeatClass = pFWS.CreateFeatureClass(strName, pFields, Nothing, _
                                         Nothing, esriFTSimple, strShapeFieldName, "")
End Sub
```

Better alternative: Use the geoprocessing object!



```
' ~~~ VBA/VB6 CODE ~~~
```

```
' Create Geoprocessing object
```

```
Dim pGP as Object
```

```
Set pGP = CreateObject("esriGeoprocessing.GPDispatch.1")
```

```
'Set the OverwriteOutput setting to True
```

```
pGP.overwriteoutput = True
```

```
' use Geoprocessing tools
```

```
pGP.CreateFeatureClass_management("C:\St_Johns", "roads.shp")
```

```
pGP.AddField_management("C:\St_Johns\roads.shp", "ROAD_NAME", "TEXT")
```

GPDispatch – Geoprocessing Object



- A **coarse grained** ArcObject which provides access to all tools and environments
- Can be used in scripting languages which support the COM IDispatch interface: Python, Perl, VBScript, Jscript, etc...
- Can also be used in programming languages: VB, VBA
 - The object is late-bound, so intelli-sense is not supported for all calls



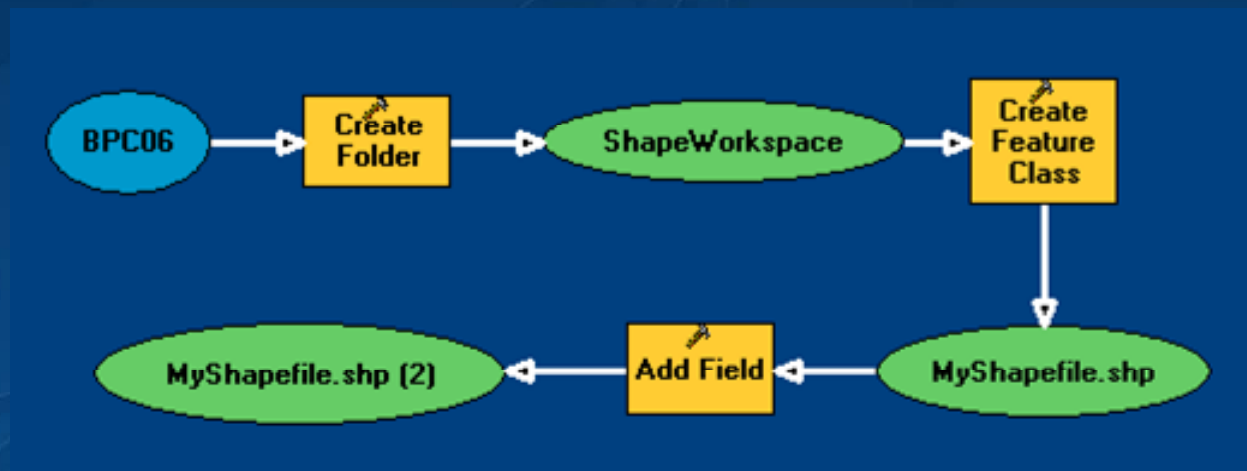
Key points

- “Compute with data” – simple and relevant syntax for data operations
- GIS Professionals can easily express their ideas
 - Some boilerplate code to create the GP object, and you’re off and running
- Ubiquitous – the GP object is available in any compliant language
- Consistent user interface, documentation, and sharing

Running Model and Script Tools



- It is also possible to execute your custom tools such as model tools and script tools which exist in custom toolboxes.



Running Model and Script Tools



~~~ VBA/VB6 CODE ~~~

‘ Create Geoprocessing object

Dim pGP as Object

Set pGP = CreateObject("esriGeoprocessing.GPDispatch.1")

‘ Add Custom Toolbox

pGP.AddToolbox "C:\Data\BestPath.tbx"

‘Set the OverwriteOutput setting to True

pGP.overwriteoutput = True

‘Set the Workspace Environment

pGP.Workspace = "C:\Data"

‘ Run CalculateBestPath Model tool

pGP.CalculateBestPath "souce.shp", "destination.shp", "bestpath.shp"

# Running Tools in C++ and .NET



- IDispatch is not well supported.
  - Use IGPCoMHelper to execute tools
  - Same idea as GPDispatch
    - Populate a property set of parameters values
    - A single place for tool execution and environment settings

# Running Tools in C++ and C#.NET



## // Create required objects

```
IGPComHelper pHelper = new GpDispatchClass();  
IPropertySet pPropertySet = new PropertySetClass();  
ITrackCancel pTC = new CancelTrackerClass();  
IGPMessages pMessages = new GPMessagesClass();
```

## // Input feature class

```
string input_fc = "C:\\workspace\\near1.shp";  
pPropertySet.SetProperty("in_features", input_fc);
```

## // Output feature class

```
string out_fc = "c:\\workspace\\nearbuff.shp";  
pPropertySet.SetProperty("out_feature_class", out_fc);
```

## // Buffer distance

```
string buff_dist = "100";  
pPropertySet.SetProperty("buffer_distance_or_field", buff_dist);
```

## //Add the Toolbox

```
pHelper.AddToolbox("C:\\ArcGIS\\ArcToolbox\\Toolboxes\\Analysis Tools.tbx");
```

## //Execute Buffer Tool

```
pMessages = pHelper.Execute("buffer_analysis", pPropertySet, pTC);
```

# Geoprocessing Messages



- Executing a tool will produce messages. These can be:
  - Informative messages
  - Or warning messages
  - Or error messages
- Trap for errors (using the programming language's built in error handling).

# Running Tools - Summary



- GpDispatch object provides access to all tools and environments.
  - Used in scripting languages, VBA, VB6, and VB.NET.
- GPCOMHelper – used to execute tools in C++ and C#.NET.
- Use Model and Script tools to create new geoprocessing tools that can be called in your program.

# Developing Function Tools



- Most ESRI Geoprocessing tools are implemented as COM function tools.
- Requires ( a minimum) of implementing two interfaces:
  - *IGPFunction*
  - *IGPFunctionFactory*
- Many tools can be included in a single DLL.

# Developing Function Tools - Usage



- Determine the tool usage:
  - What is the tool name?
  - What are the parameters and their properties?
    - Name
    - Type – Required, Optional, Derived
    - DataType – FeatureClass, Raster
    - Value – FeatureClass, Raster
    - Direction – Input, Output
    - Domain – set of values
    - Dependencies – one parameter dependent on another

# Usage Example



- **Clip** <in\_features> <clip\_features>  
<out\_feature\_class> {cluster\_tolerance}
- **AddField** <in\_table> <field\_name> <LONG | TEXT |  
FLOAT | DOUBLE | SHORT | DATE | BLOB>  
{field\_precision} {field\_scale} {field\_length} {field\_alias}  
{NULLABLE | NON\_NULLABLE} {NON\_REQUIRED |  
REQUIRED} {field\_domain}

<> = required

{ } = optional

UPPERCASE = Keyword

# Parameters



- Defines the characteristics of the inputs and outputs to the function tool.
- IGPFFunction::ParameterInfo property
  - Define the parameters
  - Returns an array (IArray) of parameter objects (IGPPParameter).



# Parameter Type

- Required
  - <input\_featureclass>
- Optional
  - {cluster\_tolerance}
- Derived
  - Used to indicate when the tool updates the input parameter i.e. AddField
  - Does not show up on the dialog or command line
  - Used in ModelBuilder to indicate changed state of the value and to provide proper chaining.

# Parameter Type - Derived



'VB Code

' Define the datatype of the Derived Parameter

```
Dim pGPParameterEdit as IGPParameterEdit
```

```
Set pGPParameterEdit = New GPPParameter
```

```
Set pGPParameterEdit.DataType = New DEFeatureClassType
```

'Value object is DEFeatureClass

```
Set pValue = New DEFeatureClass
```

```
Set pParamEdit.Value = pValue
```

'Set Output Parameter properties

```
pParamEdit.Direction = esriGPPParameterDirectionOutput
```

```
pParamEdit.DisplayName = "Output FeatureClass"
```

```
pParamEdit.Enabled = True
```

```
pParamEdit.Name = "out_featureclass"
```

```
pParamEdit.ParameterType = esriGPPParameterTypeDerived
```

```
pParamEdit.AddDependency "input_featureclass"
```

# Parameter DataType



- Defines the datatype of each parameter
  - DataTypes: FeatureClass, Table, Rasters, ...
  - Basic Types: String, Double, Boolean, ...
  - Geoprocessing Structures: spatial reference, extent, cell size, remap table, ...
  - Complex Structures: list of values, composite datatype (either | or)

'VB Code

' Define the datatype of the Input Parameter

```
Dim pGPPParameterEdit as IGPPParameterEdit
```

```
Set pGPPParameterEdit = New GPPParameter
```

```
Set pGPPParameterEdit.DataType = New DEFeatureClassType
```

# Parameter DataType



- Supporting layers and tables in ArcMap and ArcGlobe:
  - GPTableViewType, GPFeatureLayerType, GPRasterLayerType...

‘ Define the datatype of the Input Parameter to accept layers

```
Dim pGPPParameterEdit as IGPPParameterEdit
```

```
Set pGPPParameterEdit = New GPPParameter
```

```
Set pGPPParameterEdit.DataType = New GPFeatureLayerType
```

# Parameter DataType - Lists



- GPValueTypeType – a table of one or more datatypes.
  - Define the datatype for each column in the value table (Union Tool)

## 'GPValueTypeType

```
Dim pValueTypeType As IGPValueTypeType  
Set pValueTypeType = New GPValueTypeType
```

## 'GPValueTable

```
Dim pGPValueTable As IGPValueTable  
Set pGPValueTable = New GPValueTable
```

## 'First DataType

```
Dim pDataType1 As IGPFeatureLayerType  
Set pDataType1 = New GPFeatureLayerType  
pValueTypeType.AddDataType pDataType1, "input_Features", 100, Nothing
```

## 'Second DataType

```
Dim pDataType2 As IGPDoubleType  
Set pDataType2 = New GPDoubleType  
pValueTypeType.AddDataType pDataType2, "double", 100, Nothing
```

## 'Add DataTypes to the GPValueTable

```
pGPValueTable.AddDataType pDataType1  
pGPValueTable.AddDataType pDataType2
```

# Parameter DataType - Composite



- GPCCompositeDataType – this datatype is used when the parameter can be more than one datatype.
  - i.e. Append Tool

## 'Create the GPCCompositeDataType

```
Dim pGPCCompositeDataType As IGPCCompositeDataType  
Set pGPCCompositeDataType = New GPCCompositeDataType
```

## 'Create the DataTypes that are permitted as input

```
Dim pDataType3 As IGPDatatype  
Set pDataType3 = New GPTableViewType  
Dim pDataType4 As IGPDatatype  
Set pDataType4 = New GPRasterLayerType
```

## 'Add datatypes

```
pGPCCompositeDataType.AddDataType pDataType3  
pGPCCompositeDataType.AddDataType pDataType4
```

## 'Set the Input Parameter

```
Set pParamEdit = New GPParameter  
Set pParamEdit.DataType = pGPCCompositeDataType  
Set pParamEdit.Value = New GPTableView
```

# Parameter Value



- For each DataType there is a Value object (IGPValue).
  - FeatureClass, FeatureLayer, Table ...
- Used as actual inputs to each function.
  - Contains the path to the FeatureClass
- Arrays of values are created based upon the same order as the parameters.
- This array is input to the validate and execute methods of the function.

# Parameter Domain



- Used to filter values
  - Feature type: Polygon, Polyline, Point, ...
  - Field type: string, double, ...
  - Coded Value domain (fixed list of values)
  - Range domain (range of numeric values)
- For a complete list of Domain objects, refer to the ArcGIS Developer Help and search for *IGPDomain*.

# Parameter Dependency



- Field parameters are commonly dependent on another table/featureclass parameter.
  - E.g. The list of fields for a field parameter is dependent on the input table.

# IGPFunction::Validate



- Performs *light-weight verification* that a given set of values are of the appropriate number, type, and value.
- Returns an array of messages, identifying warnings/errors, with a one-to-one correspondence to the array of parameters.
- Three important roles of Validate:
  - Basic validation – number of required values, datatype, etc. ( use `GPUutilities::InternalValidate` )
  - Complex interaction between parameters
  - Populate properties of the output parameter



# Validate - InternalValidate

- Method available on GPUtilities object.
- Tests the validity of a set of values against a set of parameters.
  - Have all the required parameter values been supplied?
  - Are the values of the appropriate data types?
  - Does the input exist?
    - This does not happen in ModelBuilder when the output value is the input to another tool (i.e. a derived value)

# Validate – Parameter Interaction



- Set rules for parameters.
  - Adding a field
    - limit the length of the field name. Sets maximum number of allowable characters.
- Enable/Disable parameters.
  - AddField
    - Selecting FieldType of Text enables the Field Length parameter.

# Validate – Update Output Properties



- Update the value (e.g. schema) of the output parameter.
  - i.e. AddField, Overlay operations
- This is important when using the tool in ModelBuilder.
- Use `GPUtilities::UnPackGPValue()` and `GPUtilities::PackGPValue()`

# IGPFunction::Execute



- Method to execute the tool given the array of parameter values.
- Execute should do the following:
  - Call Validate
    - basic validation
  - Open the datasets
    - do this after calling validate
    - Create objects from inputs (IFeatureClass, IField, etc)
  - Check if overwrite output is true or false
    - Delete the output if overwrite output is true
  - Perform the operation

# GPEnvironmentManager



- Object for managing all environments and settings used by the tools.
- GPEnvironmentManager is passed to Validate() and Execute().
- The tool will then have access to all of the current environments and settings.

# GPMessages Object



- Manages an array of GPMessage objects.
- Contains methods to generate and replace message objects.
- Validate() returns a new GPMessages object, with the same number and order as the parameter definition.
  - GPUilities::InternalValidate() creates/returns a new GPMessages object based upon a given parameter definition.
- Execute() is passed an existing GPMessages object and simply adds new messages as needed.

# GPMessage Object



- GPMessage objects are composed of a message type, error code, and description properties.
- GPMessage contains the following methods:
  - IsInformational
  - IsError
  - IsWarning
  - IsAbort

# GPUtilities Object



- COM object which contains useful helper methods for the function writer
  - UnPackGPValue() to get the GPValue from the parameter.
  - PackGPValue() to put the GPValue into the parameter.
  - Exists
  - Delete
  - OpenDataset to open the datasource associated with a value
  - Others...

# IGPFunctionFactory



- “Serves” up the functions.
  - Factory is responsible for handing out the function name objects for each function.
- Registered in the CATID\_GPFunctionFactories
- Logical grouping of functions.

# Function Tools - Summary



- Function Tools are built using ArcObjects.
- Required to implement *IGPFunction* and *IGPFunctionFactory*
- Functions in most cases are a single operation composed of parameters (recommended).
- Define the Tool's usage
  - Set the type, datatype, direction, domains, dependencies, etc.
- Each parameter is a specific data type.

# Function Tools - Summary



- Validate – 3 important roles of Validate()
- Execute – Call Validate(), Open Datasets, check for fields, perform operation.
- *IGPFunctionFactory/IGPDataTypeFactory* serves up the functions and datatypes.
- GPUilities provide helper functions
  - UnpackGPValue, PackGPValue, OpenDataset, etc.

# Custom Tools - IGPToolExtension



- Similar, in concept, to a IGPFFunction
- Function tools have static definition (i.e., behavior & parameters)
- Custom tools have dynamic definition (e.g., like model tools)
- Requires implementing **four** interfaces:
  - IGPToolExtension,
  - IGPToolExtensionFactory
  - IClone
  - IPersistStream

# Custom Tools



- **VB6 users** must implement:
  - IGPToolExtensionGen
  - IGPToolExtensionFactoryGen
- The ParameterInfo is similar to IGPFFunction, but the parameter array may be dynamic.
- Validate and Execute are equivalent to IGPFFunction
- Can override the Create and Edit methods to invoke as a special dialog to edit this tool type

# Custom Tools - Summary



- Tool Extensions are similar to function tools except the parameter definition can be dynamic.
- Necessary when wanting to invoke a custom dialog or have dynamic parameter definition.
- Tool Extensions usually have state, therefore implement persistence and cloning.
- ToolExtensionFactory serves up the ToolExtension.

# What's New with ArcGIS 9.2



- IGeoprocessor interface
  - Recommended way to execute tools.
  - Coarse grain object similar to GPDispatch.
  - For COM and C++ developers
  - Clean and easy way to execute tools, set environments, and get messages.

# IGeoprocessor – C++



```
// Intialize the Geoprocessor COM Object
CoInitialize(NULL);
IGeoProcessorPtr ipGP(CLSID_GeoProcessor);

// Overwrite output set to true
ipGP->put_OverwriteOutput(VARIANT_TRUE);

// Generate BSTR input value
_bstr_t input(L"C:\\Workspace\\near1.shp");

// Build the variant input
_variant_t vInput(input);

// Generate BSTR output value
_bstr_t output(L"C:\\Workspace\\near1 copy.shp");

// Build the variant output
_variant_t vOutput(output);

// Build the array of variant parameters
IVariantArrayPtr ipVarArray(CLSID_VarArray);
ipVarArray->Add(vInput);
ipVarArray->Add(vOutput);

HRESULT hr = ipGP->Execute(L"CopyFeatures_management", ipVarArray, 0, 0, 0);
```

# Geoprocessor - .NET and Java



- New Object called the *Geoprocessor*.
- A Geoprocessing tool is executed by passing the tool process as input to the Geoprocessor.
- Tool members have parameter information and documentation.

# Running a Tool in .NET



```
using ESRI.ArcGIS.Geoprocessor;  
using ESRI.ArcGIS.AnalysisTools;
```

```
// Intialize the Geoprocessor
```

```
Geoprocessor GP = new Geoprocessor();
```

```
Erase eraseTool =
```

```
new Erase (@"C:\Data\Input.shp", @"C:\Data\Erase.shp", @"C:\Data\Output.shp");
```

```
GP.Execute(eraseTool, null);
```

```
Or
```

```
Erase eraseTool = new Erase();
```

```
eraseTool.in_features = @"C:\Data\Input.shp";
```

```
eraseTool.erase_features = @"C:\Data\Erase.shp";
```

```
eraseTool.out_featureEclass = @"C:\Data\output.shp";
```

# Running a Tool in Java



```
// Create the Geoprocessor
```

```
GeoProcessor gp = new GeoProcessor();
```

```
GetCount countTool = new GetCount("C:/data/usa/usa.gdb/states");
```

```
gp.execute(countTool);
```

```
System.out.println("Count: " + countTool.getRowCount());
```

# Summary



- Geoprocessing is for GIS Professionals
  - Tools perform essential and elemental tasks
    - Some of these tools go back > 25 years
  - Computing with data – express ideas quickly and easily
  - Framework for creating, managing, executing, documenting, and sharing tools
- Geoprocessing is for developers
  - Easy access to a rich set of tools from any language
  - Reduce barriers between GIS professionals and software developers
  - Ubiquitous platform for delivering software

# Additional Support



- Building Geoprocessing Function Tools
  - [http://arcgisdeveloperonline.esri.com/ArcGISDeveloper/ArcGISDevHelp/TechnicalDocuments/Geoprocessing/Build\\_GP\\_Functions/Build\\_GP\\_Functions.htm](http://arcgisdeveloperonline.esri.com/ArcGISDeveloper/ArcGISDevHelp/TechnicalDocuments/Geoprocessing/Build_GP_Functions/Build_GP_Functions.htm)
- For additional information about geoprocessing including service pack information, knowledge base articles, and up to the minute information about issues go to:
  - <http://support.esri.com/geoprocessing>

# Session Evaluations Reminder



Session Attendees:

Please turn in your session evaluations.

*... Thank you*