Use Geoprocessing Tools in ArcObjects Applications

This approach saves time and reduces code

By Jason Pardy, ESRI Product Engineer

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

Within programs it is often necessary to manipulate and analyze geographic data by transforming the coordinate system by projecting datasets from one projection to another, adding fields to a table, or creating buffer zones around features. ArcGIS 9 includes hundreds of geoprocessing tools for performing such tasks. This article describes how ArcGIS Desktop and ArcGIS Engine users can use geoprocessing tools in ArcObjects applications.

The Geoprocessor

The geoprocessor is the main object that simplifies the task of executing a geoprocessing tool. This object is the single access point for the execution of any geoprocessing tool in ArcGIS. The geoprocessor is a coarse-grained object containing properties and methods that make it possible to

- Execute tools.
- Set global environment settings.
- Examine the resulting messages.
- Perform batch processing.
- Access data properties.

Toolboxes define the set of tools for the geoprocessor. Toolboxes can be added and removed from the geoprocessor.

The geoprocessor object can be accessed via any system language including Microsoft Visual Basic, Microsoft .NET, and Java. For the purposes of this article, all code listings use C#. Links to the help for Java developers are also provided in "Additional Resources for Programming with Geoprocessing Tools" (see the PDF document that accompanies the online version of this article at www.esri.com/arcuser.)

ArcGIS 9.2 includes a new .NET assembly called the ESRI.ArcGIS.Geoprocessor. This assembly contains a managed class called the Geoprocessor. Each toolbox provided by ESRI is also represented by a managed assembly. In each toolbox assembly, there are classes representing each geoprocessing tool. Tool classes are used to set up and run a tool with the geoprocessor. The code in Listing 1 shows how to set up the buffer tool and execute it with the geoprocessor.

Running Custom Tools

Custom tools, such as model tools and script tools, can be executed from custom toolboxes. Using the integrated development environment (IDE) framework in Visual Studio, .NET users can generate a geoprocessing assembly to represent any custom toolbox by using the ArcGIS Toolbox reference dialog box.

Execute a Tool by Name

Generating a geoprocessing assembly is not required to represent a custom toolbox. There is an alternative way to use the Execute() method on the geoprocessor. The Execute() method is overloaded *[i.e., multiple methods in the same class that share the same name but have different parameter lists]* and has an additional argument list that allows users to execute a tool by specifying the tool name, the parameters of the tool, and the ITrackCancel interface. In this case, the toolbox must be added to the geoprocessor and the parameters are provided as a variant array where the parameters are supplied in the same order as the tool usage. Listing 2 shows an example of executing the CalculateBestPath model tool, which is located in the BestPath toolbox.

Geoprocessing Results and Messages

All geoprocessing tools generate results and messages. The Execute() method returns an IGeoProcessorResult object that manages the results. The result object will have the return value of a tool when executed. Return values are necessary when a tool has no output dataset but instead has an output scalar value such as an integer or Boolean. Listings 3 and 4 show how to retrieve the return values.

When a tool is executed, it produces messages. Geoprocessing messages can be retrieved from the geoprocessing result object or directly from the geoprocessor. If a tool fails during execution, the geoprocess-

Method	Description
ListDatasets (string Wild Card, string Dataset Type)	Returns an enumeration of datasets in the current workspace based on a query string and type
ListFeatureClasses (string Wild Card, string Feature Type, string Dataset)	Returns an enumeration of feature classes in the current workspace or optional dataset based on a query string and type
ListRasters (string Wild Card, string Raster Type)	Returns an enumeration of rasters in the current workspace based on a query string and type
ListTables (string Wild Card, string Table Type)	Returns an enumeration of tables in the current workspace based on a query string and type
ListWorkspaces (string Wild Card, string Workspace Type)	Returns an enumeration of workspaces in the current workspace based on a query string and type

Table 1: Geoprocessor methods for cataloging data

Resources for programming with geoprocessing tools can be found at www.esri.com/arcuser.

ing result object is null and it is necessary to get the messages from the geoprocessor.

A tool message will be classified as an informative message, a warning message, or an error message. A message's type is indicated by its severity property, which is a numeric value. The GetMessages() method will return all the messages for a specified severity. Listing 5 shows how to retrieve messages by severity. Listing 6 shows how to retrieve individual messages using the GetMessage() method. The result object is necessary to support geoprocessing with ArcGIS Server.

Batch Processing

Often, geoprocessing tasks must be repeated many times. A classic scenario for batch processing would be using the Clip tool to clip multiple input datasets to a predetermined study area polygon. The geoprocessor provides a number of methods for cataloging the available data so it can iterate through the data during processing. These methods work with all different types of data and provide flexibility for restricting a search by name or data category. Table 1 lists these methods and their syntax.

The methods listed in this table require the workspace environment to be previously specified to identify the location from which the enumeration will be created. The result of each of these methods is an IGpEnumList. Once the enumeration has been created with the desired values, loop through it in the program to work with each individual value. The example in Listing 7 shows how to list all TIFF files in a workspace and build pyramids.

Describing Data

Geoprocessing tools work with all types of data—geodatabase feature classes, shapefiles, rasters, tables, topologies, and networks. Data properties may be used to control the flow of a program. Using the Geoprocessor's GetDataElement() method, a dataset's properties can be determined and used to make decisions. The GetDataElement() method returns an IDataElement object allowing a program to determine the data properties such as

- Spatial reference
- Extent of features
- List of fields
- Shape type (e.g., point, polygon)
- Data type (e.g., coverage, shapefile)

Listing 8 describes an input feature class and returns the shape type.

Additional Examples

The examples in Listings 9 and 10 highlight how geoprocessing tools can save time and reduce the amount of code necessary to perform an operation. For links to additional information on topics covered in this article as well as information about getting started with geoprocessing and information on individual geoprocessing tools, see "Additional Resources for Programming with Geoprocessing Tools," available with the online version of this article.

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

// Initialize the Geoprocessor
GeoProcessor GP = new Geoprocessor();

// Set workspace environment
GP.SetEnvironmentValue("workspace", @ "C:\Data\Nfld.
gdb");

// Initialize the Buffer Tool
Buffer bufferTool = new Buffer();
bufferTool.in_features = "roads";
bufferTool.out_feature_class = "roads_500";
bufferTool.buffer_Distance_or_field = "500 METERS";

// Execute the buffer
GP.Execute(bufferTool, null)
```

Listing 1: Execute the Buffer tool

```
using ESRI.ArcGIS.Geoprocessor;
using ESRI.ArcGIS.esriSystem;

public void SampleCalculateBestPathTool()
{

  // Initialize the geoprocessor.
  Geoprocessor GP = new Geoprocessor();

  // Add the BestPath toolbox.
  GP.AddToolbox(@"C:\SanDiego\BestPath.tbx");

  // Generate the array of parameters.
  IVariantArray parameters = new VarArrayClass();
  parameters.Add(@"C:\SanDiego\source.shp");
  parameters.Add(@"C:\SanDiego\destination.shp");
  parameters.Add(@"C:\SanDiego\bestpath.shp");

  // Execute the model tool by name.
  GP.Execute("CalculateBestPath", parameters, null);
```

Listing 2: Execute a model tool by name

Continued on page 32

Use Geoprocessing Tools in ArcObjects Applications

Continued from page 31

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

// TODO: Initialize Buffer and provide parameter
values.

// The return value is an Object of type string
IGeoProcessorResult pResult = (IGeoProcessorResult)
GP.Execute(bufferTool, null);
object path = pResult.ReturnValue;

IFeatureClass pFC = GP.Open(path);
```

Listing 3: Return the ArcCatalog path to the output feature class

```
using ESRI.ArcGIS.Geoprocessing;
using ESRI.ArcGIS.Geoprocessor;
using ESRI.ArcGIS.Geoprocessing;
using ESRI.ArcGIS.DataManagementTools;

// TODO: Initialize Calculate Grid Index and provide
parameter values.

// The return value is an Object of type double.
IGeoProcessorResult pResult = GP.Execute(calculateGri
dIndexTool, null);
object defgrid = pResult.ReturnValue;
```

Listing 4: Return the default grid size

```
IGeoProcessorResult  result = (IGeoProcessorResult )
GP.Execute(bufferTool, null);

object infoMsgs = 0;
object errorMgs = 2;

if (result == null)
{
   // Informative messages
   Console.WriteLine(GP.GetMessages(ref errorMgs));
}
else
{
   // Error messages
   Console.WriteLine(result.GetMessages(ref in-
foMsgs));
}
```

Listing 5: Get all tool messages by severity

```
// Execute Union
IGeoProcessorResult result = GP.Execute(uniontool,
null);

if (result. != null) {
   for (int Count = 0; Count <= result.MessageCount -
1; Count++)
   {
      Console.WriteLine(result.GetMessage(Count));
   }
}
```

Listing 6: Get individual messages

```
// List all TIFF files in the workspace and build
pyramids
GP.SetEnvironmentValue("workspace", @"C:\Ccm\Data");

IGpEnumList rasters = GP.ListRasters("*", "TIFF");
string raster = rasters.Next();

// Intialize the BuildPyramids tool
BuildPyramids pyramids = new BuildPyramids();

while (raster != "") {
       // Set input raster dataset pyramids.
       pyramids.in_raster_dataset = raster;
       GP.Execute(pyramids, null);
       raster = rasters.Next();
}
```

Listing 7: Iterate through a list of TIFF files and build pyramids

```
// Describe the input feature class.
object dtype = "";
IDataElement dataelement = GP.GetDataElement(@"C:\GP\
PortlandOR.gdb\streets", ref dtype);

// Open the feature class data element and get the
shape type.
IDEFeatureClass defc = dataelement as IDEFeature-
Class;
if (defc.ShapeType == esriGeometryType.esriGeometry-
Polyline)
 Console.WriteLine("ShapeType is polyline.");
```

Listing 8: Describe a feature class to obtain the properties

```
using ESRI.ArcGIS.Geoprocessor;
using ESRI.ArcGIS.DataManagementTools;

// Adding a Field - Geoprocessing
Geoprocessor GP = new Geoprocessor();
AddField addfieldtool = new AddField(@"C:\nfld.gdb\
wells", "Depth", "DOUBLE");

// Execute tool
IGeoProcessorResult result = (IGeoProcessorResult )
GP.Execute(addfieldtool, null);
```

Listing 9: Add a field to a table

```
// Assign a Domain to a Field
Geoprocessor GP = new Geoprocessor();
AssignDomainToField domainToField = new
AssignDomainToField(@"C:\nfld.gdb\wells", "Diameter",
"Diameter");

// Execute the tool
IGeoprocessorResult result = (IGeoProcessorResult )
GP.Execute(domainToField, null);
```

Listing 10: Add a domain to a field