Contents

[ArcGIS.Desktop.Core.Geoprocessing 1](#_Toc417241137)

[Overview 1](#_Toc417241138)

[How to execute a tool – basic concept 2](#_Toc417241139)

[Passing .Net non-string objects to ExecuteToolAsync 2](#_Toc417241140)

[A trick to get (or visualize) correct parameter sequence 3](#_Toc417241141)

[Supported native .NET types 3](#_Toc417241142)

[How to execute a tool – details 4](#_Toc417241143)

[Setting geoprocessing environment 4](#_Toc417241144)

[Example 1 5](#_Toc417241145)

[Example 2 5](#_Toc417241146)

[Open a geoprocessing tool dialog. 6](#_Toc417241147)

[Example 1 6](#_Toc417241148)

[Example 2 6](#_Toc417241149)

[Using the delegate GPToolExecuteEventHandler 7](#_Toc417241150)

[Still to do 8](#_Toc417241151)

[ShowMessageBox 8](#_Toc417241152)

The geoprocessing public APIs are exposed through the ArcGIS.Desktop.Core.Geoprocessing namespace in the ArcGIS Pro API Reference Guide.

The core useful methods, enumerators and interfaces for geoprocessing tasks are available in this namespace.

# ArcGIS.Desktop.Core.Geoprocessing

## Overview

You will use the ExecuteToolAsync method to run any geoprocessing tool.

## How to execute a tool – basic concept

To run (or execute) a tool, in the most case, you just pass two required (along with some optional) arguments to ExecuteToolAsync method, which are:

1. Name of the tool: tool name passed as a string and is specified as follows:

toolbox\_alias.ToolName (alias and tool name are separated by a dot)

You can get the correct alias and tool name from drag-dropping a tool in Python window or from tool’s help page.

Example (python window and .Net):

Python window: arcpy.Buffer\_analysis(arg, arg, arg …)

.Net: “analysis.Buffer” // a string

1. Arguments to ExecuteToolAsync: Pass all the parameter values of the geoprocessing tool in an IEnumerable of type string.

Make sure the sequence of parameter values of the tool matches exactly with the sequence specified in tool’s reference (help) page. You can also get the sequence by drag-dropping the tool in Python window.

Note: Tool parameters are arranged on tool’s dialog as per GUI design and do not always follow the sequence specified in tool help page.

Example (python window and .Net)

Python window: input\_features, out\_feature\_class, buffer\_distance, ….

// pass values to Enumerable with same sequence as in Python window.

.Net: IEnumerable<string> args = new Enumerable(“path\_to\_output”, “out\_fc\_name”, “Polyline”)

So, a call to ExecuteToolAsync will be:

Geoprocessing.ExecuteToolAsync(“management.CreateFeatureClass”, args)

## Passing .Net non-string objects to ExecuteToolAsync

In many scenarios, passing a Layer, geometry created while interacting with the map, Envelope or Extent, SpatialReference or a Table as an argument to a geoprocessing tool may be necessary. In such cases, MakeValueArray method, which converts such objects to string, can be used.

By using the params keyword, you can specify a [method parameter](https://msdn.microsoft.com/en-us/library/8f1hz171.aspx) that takes a variable number of arguments. See this link also: <https://msdn.microsoft.com/en-us/library/w5zay9db.aspx>

Say, we have to pass the first argument as a Layer object and the 2nd and 3rd as strings, then we can pack the arguments in a List<string> either of two ways:

1. Convert the layerObject to a string and combine it with other two strings in a List<string> - the result is a 3-string enumerable:

MakeValueArray(layerObject), “second\_param\_value”, “third\_param\_value”

1. Convert all three (one object and two strings) to strings at once – the result is same – a 3-string enumerable:

MakeValueArray(layerObject, “second\_param\_value”, “third\_param\_value”)

Note:

1. MakeValueArray( var1, val2) 🡪 returns 2 strings
2. MakeValueArray(var1, list(a, b, c) 🡪 also returns 2 strings

Code example – if you add objects to the list then you’ll need to convert the args to an array as shown:

List<object> args = new List<object>();

// populate args

// if all arguments are string, you won’t even need to call MakeValueArray

ExecuteToolAsync(“management.CreateFeatureClass”, Geoprocessing.MakeValueArray(args.ToArray()));

### Supported native .NET types

The following .Net types will be supported in ArcGIS Pro 1.1:

Scalars – long, short, float, double, date, string

ArcGIS.Core.Geometry.SpatialReference

ArcGIS.Core.Geometry – point, line, polygon

ArcGIS.Core.Geometry.Envelope – supporting GP types of GPExtentEnv, GPExtent, GPEneveope

ArcGIS.Core.Data.Field – supporting GPField and list of fields for GPFieldList

ArcGIS.Desktop.Mapping – Layer, StandaloneTable

ArcGIS.Core.Data.Dataset – Table, FeatureClass

## How to execute a tool – details

Until now only the required parameters of ExecuteToolAsync is discussed. However, there are 4 optional parameters. The full syntax is:

(awaitable) Task<IGPResult> Geoprocessing.ExecuteToolAsync( string toolpath,

IEnumerable<string> parameters,

[ IEnumerable< ……>> environments = null ], \\ ask Dima

[ CancellationToken? cancelToken = null ],

[ GPToolExecuteEventHandler callback = null ],

[ GPExecuteToolFlags flags = GPExecuteToolFlags.Default ])

Usage:

IGPResult x = await ExecuteToolAsync( …………. );

## Setting geoprocessing environment

The third parameter of ExecuteToolAsync method is an IEnumerable. You pass pair(s) of values as argument to set the environment:

This parameter is optional – if you don’t want to set any specific environment, just pass null.

System.Collections.Generic.KeyValuePair<string, string>[] envs = new System.Collections.Generic.KeyValuePair<string, string>[]

{

new System.Collections.Generic.KeyValuePair<string, string>("Output Workspace", @"c:\temp"),

new System.Collections.Generic.KeyValuePair<string, string>("Extent", "0 0 12 14")

};

Note: Example nn shows the usage of the last three optional parameters.

## FAQ

1. What are the permission types for parameter types? For example, how do I know which type I need to pass as buffer distance?

Example 1: Simplest tool execution

private async Task<IGPResult> ExecuteGetCount()

{

string[] args = new string[] {

@"C:\data\Hydrants.shp"

};

var gp\_result = await Geoprocessing.ExecuteTool("GetCount\_management", args);

if (!gp\_result.IsFailed)

{

int count = Convert.ToInt32(gp\_result.Values[0]);

System.Windows.Forms.MessageBox.Show("Result : " + gp\_result.Values[0]);

}

var messages = gp\_result.Messages;

string msgStr = "";

foreach (IGPMessage msg in messages)

{

msgStr += msg.Text + "\n";

}

System.Windows.Forms.MessageBox.Show(msgStr);

return gp\_result;

}