

ET GeoWizards

ET GeoWizards is a set of powerful functions that will help the GIS professionals to manipulate data with easy. It offers more than 100 functions for spatial data conversion, analysis, topological cleaning, sampling and many more. ET GeoWizards was initially developed as an extension for ArcGIS and for the last 15 years it became the most popular third party ArcGIS extension for data processing.

As from version 12.0 ET GeoWizards is available as a standalone application and can be used by all GIS professionals **no matter what is the GIS platform they are using**.

The functionality of ET GeoWizards is available in several ways:

- Stand-alone Windows application with user friendly interface and integrated User Guide.
- Seamlessly integrated in ArcGIS Desktop via
 - ArcGIS Desktop add-in that starts ET GeoWizards dialog from ArcMap
 - An ArcGIS toolbox that allows the functionality to be executed from ArcToolbox, included in a Model using the Model Builder or ArcPy scripts.
- Seamlessly integrated in ArcGIS Pro via
 - ArcGIS Pro add-in that starts ET GeoWizards dialog from within the ArcGIS Pro interface
 - An ArcGIS Pro Toolbox that allows the functionality to be executed from ArcToolbox, included in a Model using the Model Builder or ArcPy scripts.
- All the functions can be executed using Python (no third party software required)
- The tools can be integrated in custom .NET applications (no third party software required).
- The functionality can be executed directly from the DOS Command Prompt.

ET GeoWizards is a native 64-bit application which allows handling very large datasets at impressive speed. The functionality however can be executed from any 32-bit application (example - ArcMap).

ET GeoWizards is not a free program, but it has many free functions that can be used without purchasing a license and registering the software. Until registered ET GeoWizards runs in DEMO mode.

- The Demo mode has the following limitations
 - Many of the features are free - do not have any restrictions with the DEMO version. See [ET GeoWizards - free features](#) for a list
 - The rest of the functions have restriction of 100 features in the layer to be processed.
 - Note that the Demo Mode is available only when the functions are executed from the ET GeoWizards interface.
- See [How to Register ET GeoWizards](#) for registration information

Installation Instructions

System requirements

- ET GeoWizards 12 is a 64-Bit application
- It will run on Windows 7 and above 64-Bit and Windows Server 64-Bit

Note that you have to be logged as an Administrator on the machine when you are installing ET GeoWizards

- Download ET GeoWizards from <http://www.ian-ko.com>
- Unzip ETGeoWizardsXX.zip - two files will be extracted from the archive:
 - setup.exe
 - ETGeoWizardsXXSetup.msi
- Run setup.exe - a simple installation wizard will guide you through the process.
- A new program group called ET GeoWizards 12 with 6 items will be created
 - ET GeoWizards 12 - the main executable - opens ET GeoWizards interface
 - ET GeoWizards User Guide
 - Readme
 - Install .NET Toolbox for ArcGIS Desktop
 - Register ETGW Add In for ArcGIS Desktop
 - Register ETGW Add In for ArcGIS Pro

To find the ET GeoWizards 12 Program Group

- On Windows 7
 - Click on the Start button
 - Choose All Programs
 - Open the ETGeoWizards 12 folder
- On Windows 8
 - Click on the Start button
 - Click the down button from the Windows tile screen
 - Locate the ETGeoWizards 12 section
- On Windows 10
 - Click on the Start button
 - Choose All apps
 - Select ETGeoWizards 12 folder

If you want to use ET GeoWizards from within ArcMap you need to register the ET GeoWizards for Desktop Add-In.

- Find the ET GeoWizards 12 Program Group (see above)
- Click on Register ETGW Add In for ArcGIS Desktop
- If you have ArcGIS Desktop installed the Installation Utility will open
- Click on Install Add-In
- Open ArcMap

- The ET GeoWizards 12 toolbar should be open. Note that the toolbar was introduced in ET GeoWizards 12.1
- If the toolbar is not open: In ArcMap click on Customize ==> Toolbars and check the ET GeoWizards 12 toolbar.

If you want to use ET GeoWizards from within ArcGIS Pro you need to register the ET GeoWizards for Pro Add-In.

- Find the ET GeoWizards 12 Program Group (see above)
- Click on Register ETGW Add In for ArcGIS Pro
- If you have ArcGIS Pro installed the Installation Utility will open
- Click on Install Add-In
- Open ArcGIS Pro
- Click on the Add-In Tab. The ET GeoWizards button should be available for use.

Notes:

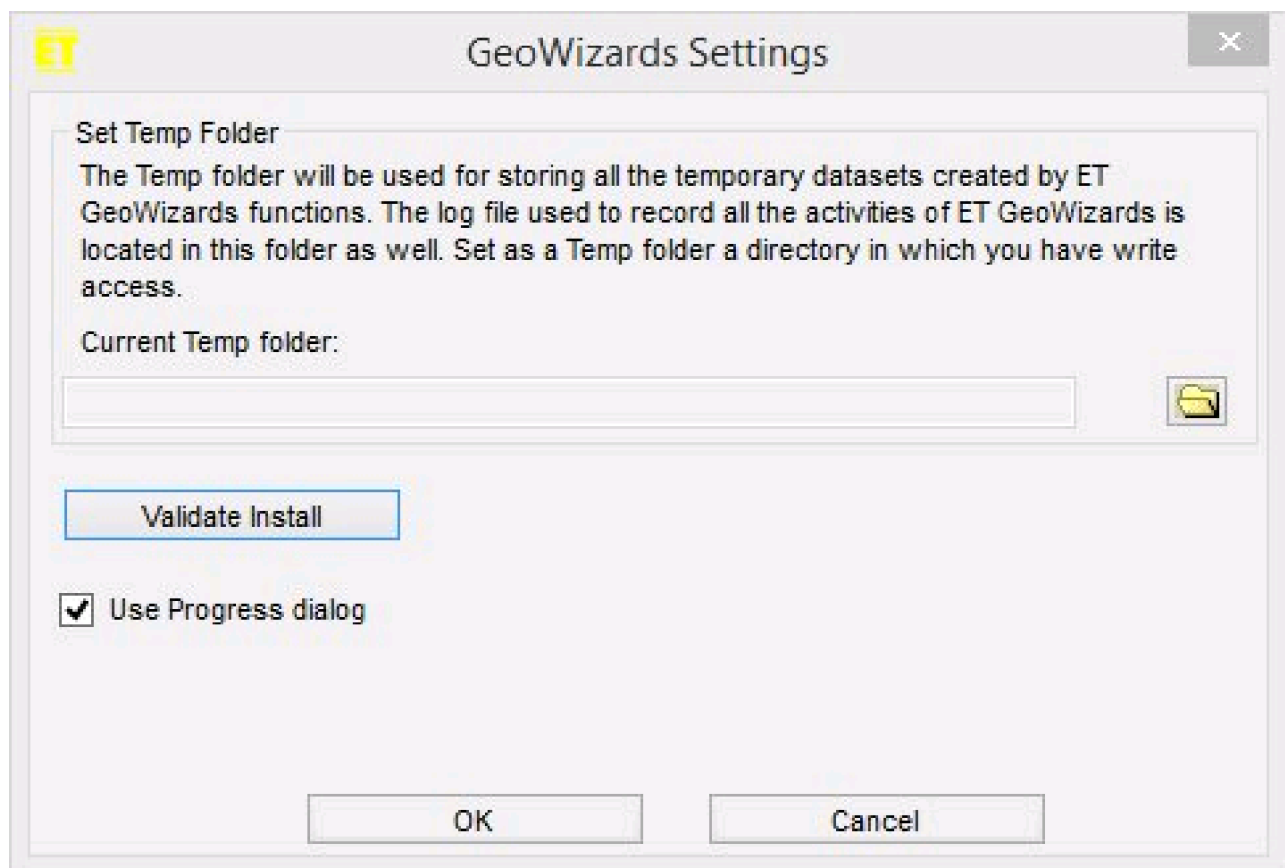
- For installing previous versions please see the user guide for ET GeoWizards 11.X
- The Demo mode has the following limitations
 - Many of the features are free - do not have any restrictions with the DEMO version. See [ET GeoWizards - free features](#) for a list
 - The rest of the functions have restriction of 100 features in the layer to be processed.
 - Note that the Demo Mode is available only when the functions are executed from the ET GeoWizards interface.
- See [How to Register ET GeoWizards](#) for registration information

Validate ET GeoWizards 12 Installation

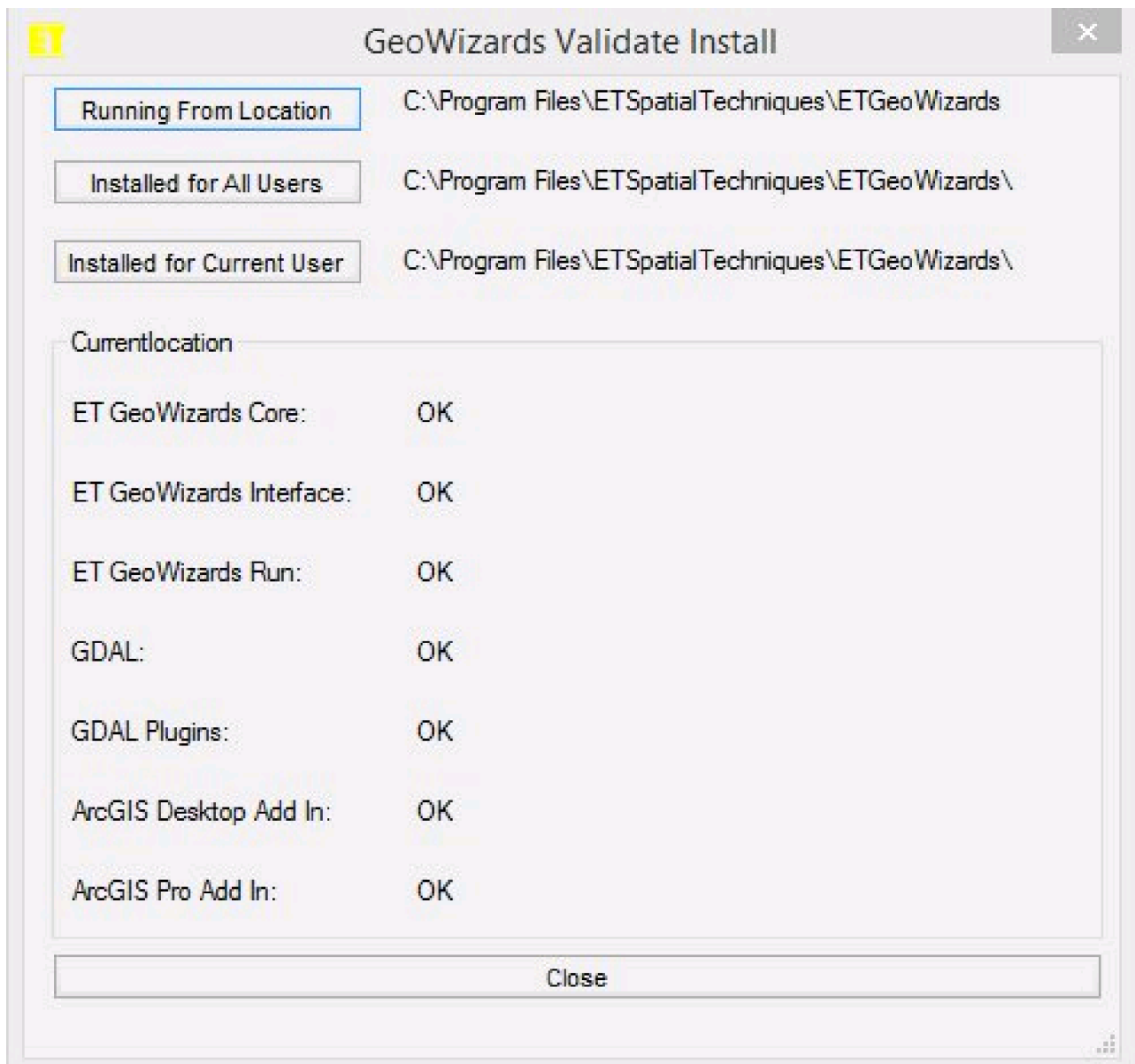
ET GeoWizards relies on the directory structure it creates when installed. So it is not recommended moving or deleting the files stored in the installation folders of ET GeoWizards. The exception to this are the ArcGIS toolboxes .PYT files which users can place in any folder.

To validate that ET GeoWizards is correctly installed:

- Start ET GeoWizards
- On the [Main Dialog](#) click on the Settings button. The settings dialog will open



- Click the Validate Install - the Validate dialog will open



- On this dialog you can check whether the installation of ET GeoWizards is correct and whether you are running it from the correct location

How to register ET GeoWizards

A. Single use (fixed) license

The registration process involves three steps:

1. Follow the links from <http://www.ian-ko.com> and purchase a license of ET GeoWizards. You will receive a reference number for your order.
2. On the [ET GeoWizards Main Dialog](#) go to Help ==> Request License Key button. Fill the small form - all the fields are required.

- User Name
- Company
- Reference number (see Step 1)

After filling the form there are two options to chose from:

- Create Key Request File will write all the information to a file (*.etr). Send this file to register@ian-ko.com and in maximum 24 hours you will receive a license key that will unlock the full version
- Send Key Request via e-mail. This option will open you default e-mail program with all necessary information. You just have to click the SEND button

Important note:

Do not change anything in the request file or the body of the generated message. It will cause the registration process to fail.

3. When you receive the Key File , save the attachment (*.etw file) to you hard disk. Click on Register button ([Main Dialog](#)). In the form click on Load Key File button. Select the received file. The ET GeoWizards dialog will close. When opened next time the program will be registered.

Important note:

Do not change anything in the Key File. It will cause the registration process to fail.

B. Concurrent license


ET LicenseManager should be installed on a PC on your network

1. Contact your system administrator and get the following information:
 - The Name or IP address of the PC where the ET LicenseManager is installed
 - The TCP port on which the ET License Manager communicates
2. On the ET GeoWizards toolbar click Help ==> Connect To License Server.
3. In the dialog fill
 - License Server - fill the network name or the IP Address of the license server
 - TCP Port - fill the port number
4. Click on the Test License Server button
5. If a connection to the license server is established, click OK to save the settings. You are ready to work.
6. If the test fails - contact your system administrator.

How to use ET GeoWizards as a stand alone application

[See this topic on how to use in ArcGIS Desktop and ArcGIS Pro](#)

A. Via the User Interface

1. Clicking on the ET GeoWizards 12 will introduce the [ET GeoWizards main dialog](#)
2. Select the appropriate group of functions in the navigation panel on the left.
3. Select the function required.
4. The appropriate topic of the User Guide will be displayed in the Help Window
5. To run the selected function click the GO button or the Run icon  next the the function name. You can also double click the function name.

Note:

The dialog for each function has a Help Tab that contains the full help topic for the current function.

B. In Python scripts.

All the functions of ET GeoWizards can be used in Python Scripts by calling ETGWRun.exe located in the installation folder of ET GeoWizards using the Subprocess Python module. ET GeoWizards 12 is a native 64-bit application, but since it runs in a separate process, it can be used from 32 and 64 bit Python.

Below is an example of using a function of ET GeoWizards within a Python script

Calling the script RotateShapes from Command Prompt

```
python.exe RotateShapes.py "c:\00\aaaa.shp" "c:\00\aaaaRotated.shp" "45" "" "5" "5"
```

```
import sys, subprocess
argList = sys.argv
print len(argList)
if len(sys.argv) < 5:
    print "Usage:" , "RotateShapes <input_dataset> <output_dataset> <rotation_angle>
{origin_point_dataset} {Origin_X} {Origin_Y}"
else:
    etgwPath = r"C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe"
    etFunction = "RotateShapes"
    inputDataset = argList[1]
```

```

outputDataset = argList[2]    rotation_angle = argList[3]
origin_point_dataset = argList[4]
Origin_X = "0"
Origin_Y = "0"
if len(sys.argv) >=6:
    Origin_X = argList[5]
if len(sys.argv) >=7:
    Origin_Y = argList[6]
print "Input: " , inputDataset
print "Output: " , outputDataset

retCode = subprocess.call([etgwPath, etFunction, inputDataset, outputDataset, rotation_angle,
origin_point_dataset, Origin_X,Origin_Y])
if retCode == 0:
    print "Success"
else:
    print retCode

```

The location of ETGWRun.exe can be derived from the registry using:

```

hKey = _winreg.OpenKey (_winreg.HKEY_LOCAL_MACHINE, r"SOFTWARE\ETSpatial
Techniques\ETGeo Wizards", 0, _winreg.KEY_READ|_winreg. KEY_WOW64_64KEY)
myPath = _winreg.QueryValueEx(hKey, "InstallPath")[0]
etgwPath = myPath+r"ETGWRun.exe"
_winreg.CloseKey(hKey)

```

C. In .NET applications

The functionality of ET GeoWizards can be also used in custom .NET applications. [See this topic for details](#)

D. From the Command Prompt

The functionality of ET GeoWizards can be executed directly from the DOS command prompt.
Example:

```
ETGWRun.exe "RotateShapes" "c:\00\aaaa.shp" "c:\00\aaaaRotated.shp" "45" "" "5" "5"
```


Notes:

- Please read the topic for a specific function for parameters description and syntax

How to use ET GeoWizards in ArcGIS

[See this topic on how to use as a Stand Alone application.](#)

A. As Add-In for ArcGIS Desktop and ArcGIS Pro

1. Register the corresponding Add-In - [see this topic for details](#).
2. Clicking on the ET button will introduce the [ET GeoWizards main dialog](#)
3. Select the appropriate group of functions in the navigation panel on the left.
4. Select the function required.
5. The appropriate topic of the User Guide will be displayed in the Help Window
6. To run the selected function click the GO button or the Run icon  next to the the function name. You can also double click the function name.

Note:

The dialog for each function has a Help Tab that contains the full help topic for the current function.

B. In ArcToolbox - ArcGIS Desktop:

Due to the fact that ArcGIS Desktop has problems with loading large Python Toolboxes (takes long time to load the toolboxes and slows down opening projects) ET GeoWizards 12.1 offers in addition to the Python Toolbox a .NET toolbox which integrates better with ArcGIS Desktop .

- .NET Toolbox.
 1. Find the ET GeoWizards 12 Program Group ([see this topic for details](#))
 2. Run the installation of ET GeoWizards 12 .NET Toolbox
 3. Right-click the ArcToolbox folder inside the ArcToolbox window and click Add Toolbox.
 4. Navigate to the folder where the ToolBox is installed (the default folder is "C:\Program Files (x86)\ET SpatialTechniques\ET GeoWizards 12 Toolbox for ArcGIS Desktop") and select "ETGeoWizards12_NET.tbx"

5. Click Open.
6. The ET GeoWizards 12 .NET toolbox will be loaded in ArcToolbox
7. Use the tools as any standard ArcToolbox tool.

- Python Toolbox.

1. Right-click the ArcToolbox folder inside the ArcToolbox window and click Add Toolbox.
2. Navigate to the folder where ET GeoWizards is installed and select :
 - For ArcGIS 10.1 and 10.2 - ETGeoWizards12_Desktop_101_102.pyt
 - For ArcGIS 10.3, 10.4 and 10.5 - ETGeoWizards12_Desktop_103_104_105.pyt
3. Click Open.
4. The ET GeoWizards toolbox will be loaded in ArcToolbox
5. Use the tools as any standard ArcToolbox tool.

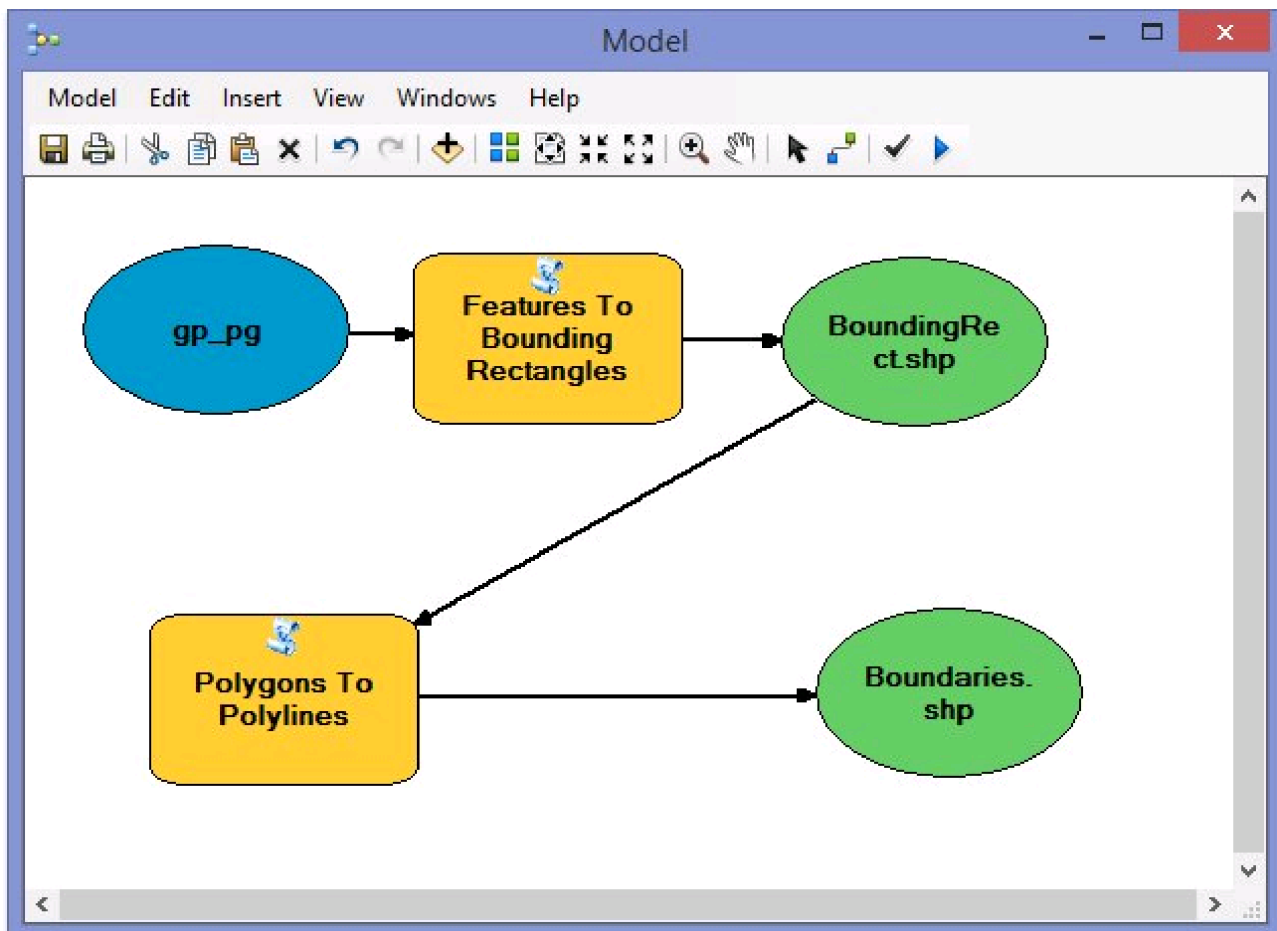
C. In ArcToolbox - ArcGIS Pro:

1. Click INSERT ==> Toolbox ==> Add Toolbox.
2. Navigate to the folder where ET GeoWizards is installed and select ETGeoWizards12_Pro.pyt
3. Click Open.
4. Open the Geoprocessing dialog - ANALYSIS ==> Tools
5. In the Geoprocessing window click on Toolboxes

6. Use the tools as any standard ArcToolbox tool.
7. Note that ArcGIS Pro might not allow you to drag a Tool to a Model. Just right click on the tool ==> Add to Model

D. In ArcPy scripts

The easiest way to get the syntax for an ArcPy script is to create a model in ArcGIS Desktop using the ET GeoWizards tools. Then export the model as a Python script. Unfortunately in ArcGIS Pro you cannot export your model as Python script. Below is a sample model using the ET GeoWizards tools and the corresponding python script.



```
# Import arcpy module
import arcpy

# Load ET GeoWizards Toolbox
arcpy.ImportToolbox("C:/00/testToolBox/ETGeoWizards12_Desktop_103_104.pyt")

# Local variables:
input = "C:\\00\\gp_pg.shp"
result1 = "C:\\00\\BoundingRect.shp"
result2 = "C:\\00\\Boundaries.shp"
```

```
# Process: Features To Bounding Rectangles
arcpy.FeaturesToRectangles_ETGeoWizards(input, result1, "LongestSegment")

# Process: Polygons To Polylines
arcpy.PolygonsToPolylines_ETGeoWizards(result1, result2, "false", "false")
```

Notes:

- The installation of ET GeoWizards places the toolboxes for ArcGIS Desktop and ArcGIS Pro in the installation folder of ET GeoWizards. Since ArcGIS Desktop and ArcGIS Pro tend to create a separate XML file for each tool in the toolbox, it is recommended to copy the required toolbox in an user folder with write access.
- Since the usage of the ET GeoWizards tools is exactly the same as the standard tools provided with ArcGIS, we highly recommend you to have a look at "Geoprocessing in ArcGIS" in the desktop help.
- For the parameters required by each tool and syntax, please have a look at the topic for the specific tool.

How to use ET GeoWizards functionality in .NET

All the functions of ET GeoWizards 12 can be used in custom applications developed in .NET. The syntax of each ET GeoWizards function is described in the main topic of the function ==> .NET implementation.

There are two ways to execute the functionality of ET GeoWizards 12 in .NET

A. As a separate process - The advantages are:

- Your application can be 32 Or 64-bit.
- You don't need to copy additional files in the folder of your application

Example:

Prerequisites

- ET GeoWizards 12 installed on the computer and registered.
- Microsoft Visual Studio

1. Start Visual Studio
2. Go to File ==> New ==> Project
 - Select Project Type (for the purpose of this example - Windows Forms Application)
3. Create a button on your form
4. Double click on the button to start editing the code
5. Paste the code below

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    'Set the path to ETGWRun.exe
    Dim ETRunPath As String = "C:\Program Files\ETSpatial Techniques\ETGeo
Wizards\ETGWRun.exe"
    'We are going to use the Explode Multipart Features for the example
    Dim sFunction As String = "ExplodeMultipart"
    'Set the input dataset with the full path to the shapefile. Note that the Argument separator is
```

space. If a string might contain a space, you need to double quote it.

```
Dim sInDataset As String = Chr(34) & "C:\testData\aaaaa.shp" & Chr(34)
'Set the full name of the output
Dim sOutFN As String = Chr(34) & "C:\testData\0000result3.shp" & Chr(34)
'Execute the ExplodeMultipart function
Dim sArgumentList As String = sFunction & " " & sInDataset & " " & sOutFN
Dim wProc As Diagnostics.Process = New Diagnostics.Process
Dim procParam As String = sArgumentList
wProc.StartInfo.FileName = ETRunPath
wProc.StartInfo.Arguments = procParam
wProc.Start()
Dim wProcID As Integer = wProc.Id
wProc.WaitForExit()
Dim iResult As Integer
If wProc.HasExited Then
    'Check the result - iResult = 0 - Success, any other returned value indicate that the function
    failed.
    iResult = wProc.ExitCode
    If iResult = 0 Then
        MsgBox("Function completed successfully!!!")
    Else
        MsgBox("Problems encountered during the execution!!! See the log file for details.")
    End If
End If
End Sub
```

B. Using directly the Core ET GeoWizards DLL

- Advantages:
 - Seamless integration.
 - You we'll be able to use some utility functions provided in the core ET GeoWizards DLL
- Disadvantages
 - Your application must be 64-bit
 - You have to copy several files in the folder from which your application will run

Example:

Prerequisites

- ET GeoWizards 12 installed on the computer and registered.
- Microsoft Visual Studio

1. Start Visual Studio
2. Go to File ==> New ==> Project
 - Select Project Type (for the purpose of this example - Windows Forms Application)
3. Go to Project ==> Properties ==> References:
 - Add reference ==> Browse ==> navigate to the installation folder of ET GeoWizards 12 and select ETGWOutX.dll. Make sure that "Copy Local" is set to true in the reference properties.
4. Make sure that your application is 64-Bit
 - Go to Project ==> Properties ==> Compile ==> Target CPU = x64
5. Save the Assembly and Build it.
6. Copy from the ET GeoWizards 12 installation folder to the folder where your application is the following files
 - the entire GDAL sub-folder
 - gdalconst_csharp.dll, gdalconst_wrap.dll, gdal_wrap.dll, ogr_wrap.dll, osr_wrap.dll
7. Create a button on your form
8. Double click on the button to start editing the code

9. Paste the code below

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    'Get reference to ETGWOutRun
    Dim ETGWOut As ETGWOutX.ETGWOutRun = New ETGWOutX.ETGWOutRun(True)
    'We are going to use the Explode Multipart Features for the example
    'Set the input dataset with the full path to the shapefile
    Dim sInDataset As String = "C:\testData\input.shp"
    'Set the full name of the output
    Dim sOutFN As String = "C:\testData\result.shp"
    'Execute the ExplodeMultipart function
    Dim iResult As Integer = ETGWOut.ExplodeMultipart(sInDataset, sOutFN)
    'Check the result - iResult = 0 - Success, any other returned value indicate that the function
    failed.
    If iResult = 0 Then
        MsgBox("Function completed successfully!!!")
    Else
        MsgBox("Problems encountered during the execution!!! See the log file for details.")
    End If
End Sub
```

ET GeoWizards and Projections

Some short definitions

- Projection - The two-dimensional representation of the three-dimensional space.
- Coordinate System - a reference system for measurements defined by the projection
 - Geographic Coordinate System - measures locations in degrees - latitude and longitude. Since latitude and longitude are angular measurements they are not suitable for measuring distances. The major parameter of a Geographic Coordinate System is its datum
 - Projected Coordinate System - uses a projection to transform the latitude and longitude to X and Y coordinates and makes the linear measurements more accurate. Each projected coordinate system is based on a Geographic Coordinate System
- Spatial Domain - the range and precision of coordinates that can be stored in a feature dataset
- Spatial Reference - contains information for the coordinate system and spatial domain extent for a feature dataset

Projections of the data

- Projected data - the data is explicitly projected in a Projected Coordinate System
- Unprojected data - the data is in a Geographic Coordinate System
- Data with unknown projection - the projection information is missing

ET GeoWizards works with data in any projection.

Notes:

- All the wizards preserve the Spatial Reference of the input data source unless an option is available and the user selects it. The assumption is that if the user keeps a dataset in certain projection he has reasons for that, and all the products of this data set must be in the same projection.

- When a distance input is required (Fuzzy tolerance, Dangling tolerance etc.) best results will be achieved if the data is projected in a specific projection, because the tolerance is compared directly with the data.
- Although possible, it is not recommended to manipulate Unprojected data (data in a Geographic Coordinate System) for reasons mentioned above.

ET GeoWizards spatial data formats

General

ET GeoWizards 12 uses Geospatial Data Abstraction Library (GDAL) for reading and writing spatial data. All functions accept as input and produce output in Shapefile or File Geodatabase (File GDB) which are the most common spatial data formats and most GIS products can read/write one or both data formats.

In order to use your spatial data in the ET GeoWizards functions you need to convert it to Shapefile or File GDB first

Shapefile

- consists of several files. The first 3 listed below are compulsory and the forth is very important if the data is to be used for spatial analysis.
 - *.shp - containing the geometry information
 - .dbf - containing the attribute information
 - .shx - positional index
 - .prj - contains the coordinate system and projection information of the data
 - etc.
- ET GeoWizards accepts as input the full name of the .shp file (for example c:\data\input.shp") and treats all the files with the same name and different extension as a single dataset. In a similar fashion if the user specifies as an output the full name to a .shp file (for example c:\data\output.shp") the functions of ET GeoWizards will produce a full set of files comprising a Shapefile dataset.
- Supports Z and M (measures) and is suitable for representing 3D geometries and used in linear referencing tasks.
- Does not support true curves. The curves are approximated using linear segments.
- The attributes are stored in a DBF file. Maximum length of field names - 10 characters

File Geodatabase (File GDB)

The Esri File Geodatabase is a collection of data layers stored in a file system. The File GDB format has emerged as a very common format for storing and exchanging spatial data. A File GDB is a folder that ends with .gdb extension.

- ET GeoWizards accepts as input the full name of a layer stored in File GDB (for example `c:\data\myData.gdb\input")`. The output is specified in a similar way (for example `c:\data\myData.gdb\output."`)
- Supports Z and M (measures) and is suitable for representing 3D geometries and used in linear referencing tasks.
- Supports true curves. The current version of GDAL (2.1) cannot read true curves stored in File GDB. As a result ET GeoWizards 12.0 does not support true curves.
- ET GeoWizards will read File Geodatabases created with ArcGIS 10.0 and above. Previous versions are not supported by GDAL.
- ET GeoWizards can create a new File GDB
- A Feature Dataset is a subfolder in the File GDB folder (for example `c:\data\myData.gdb\myNetwork\layer1")`. A Feature Dataset is normally used to group layers for a specific purpose. With ET GeoWizards it is possible to create a Feature Dataset and store the output data in it. It is however not recommended (and sometime not possible) to save output layers in a Feature Dataset created in ArcGIS).

Temporary feature classes.





Many of the functions of ET GeoWizards perform complex spatial operations and in the process need to create one or more intermediate datasets. The temporary datasets can be stored in Shapefiles or File GDB. The location (temp folder) for storing temp datasets can be set using the ET GeoWizards Main Dialog ==> Settings). If the temp folder has not be set Et GeoWizards sets it to `"c:\temp\ET_Temp"`.

Maintenance of the temp folder

All functions are designed to maintain the ET GeoWizards folder by removing the intermediate feature classes after completion. In some cases however some of the functions cannot delete the intermediate datasets. This might cause the size of the temporary folder to grow after long use of the software. The easiest way to avoid problems with large temp folder is simply to delete the contents of this folder on regular basis.

ET GeoWizards Main Dialog

The Main Dialog of ET GeoWizards gives access to all the functions of the software. Selecting the tab for specific category of functions will display a list of all functions in this category. Next to each function there is an icon indicating the status of the function

-  indicates that the function is available with no limitations
-  indicates that the software is not registered and if you run the function the limitations of the unregistered software apply
-  appears when a licensed (or free) function is selected. Clicking on the icon will execute the function.
-  appears when a non-licensed function is selected. Clicking on the icon will execute the function with the applicable to the unregistered software limitations.

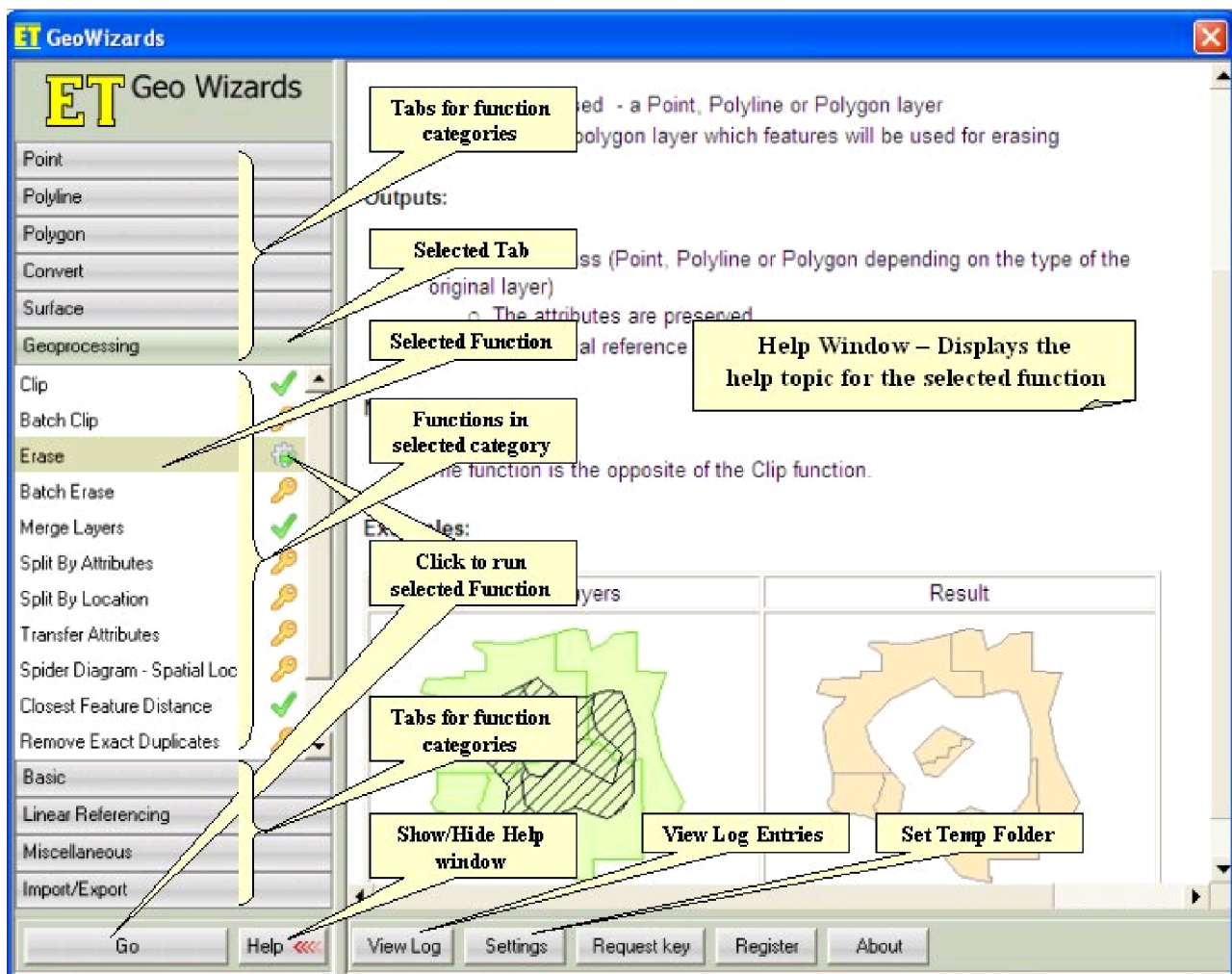
On the registered software only  and  icons should appear.

Clicking on the GO button will execute the selected function.

The User Guide is embedded in the main dialog - whenever you select a function, the Help Window will display the appropriate help topic. You can use the Help button to hide or show the Help Window.

The View Log button displays the entries recorded in the ET GeoWizards log file. The dialog allows deleting the current entries. It is recommended to clean the log file on regular intervals.

The settings button opens the settings dialog of ET GeoWizards. On this dialog you can view the current temp folder (where all intermediate datasets created by the functions of ET GeoWizards are stored) or set a new folder to be used for such purposes. ET GeoWizards cleans the temp folder automatically, but it is a good practice to delete all the contents of this folder from time to time.



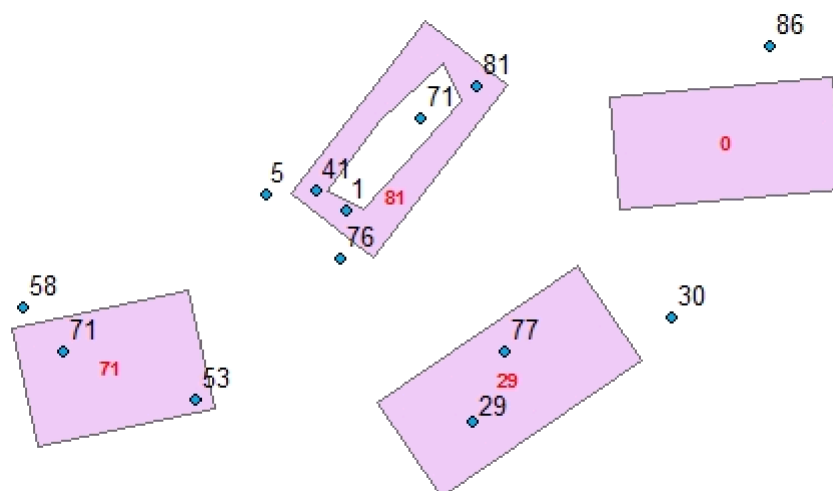
Spatial Join

[Running programmatically](#)

Joins the attributes of the Join Layer to the attribute table of the Target Layer based on spatial location.

Inputs:

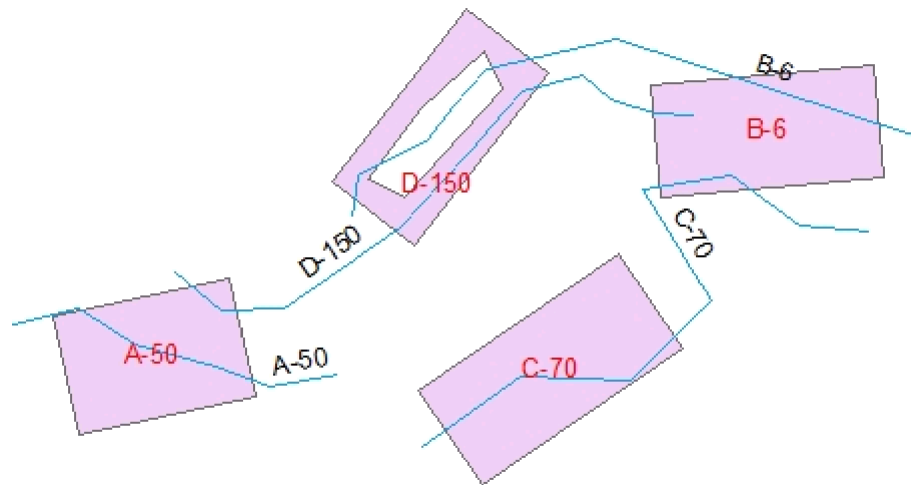
- Target Dataset - Point, Multipoint, Polyline or Polygon
- Join Dataset - Point, Multipoint, Polyline or Polygon
- Join Type
 - Join Type = "One To Many" - one feature from the Join Dataset can be joined to many features of the target dataset, but each feature from the Target Dataset will receive the attributes of only ONE of the features from the Source Dataset:
 - Point to Polygon - the polygon will get the attributes of the deepest point inside the polygon (farthest to the polygon boundary) or if there is no point inside the closest point to the polygon (within the search tolerance)
Example: Points to Polygons - Search Tolerance = 0



- Point to Point, Point To Polyline, Point To Multipoint - the target will get the attributes of the closest polyline
- Polyline to Polygon - the polygon will get the attributes of the polyline with the longest intersection with the polygon or if there is no intersecting

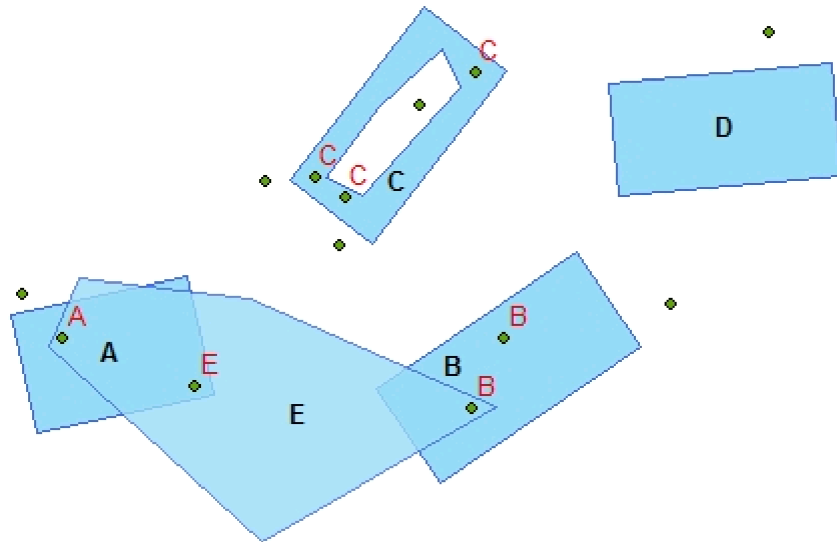
polyline the closest polyline to the polygon (within the search tolerance)

Example: Polylines to Polygons - Search Tolerance = 0

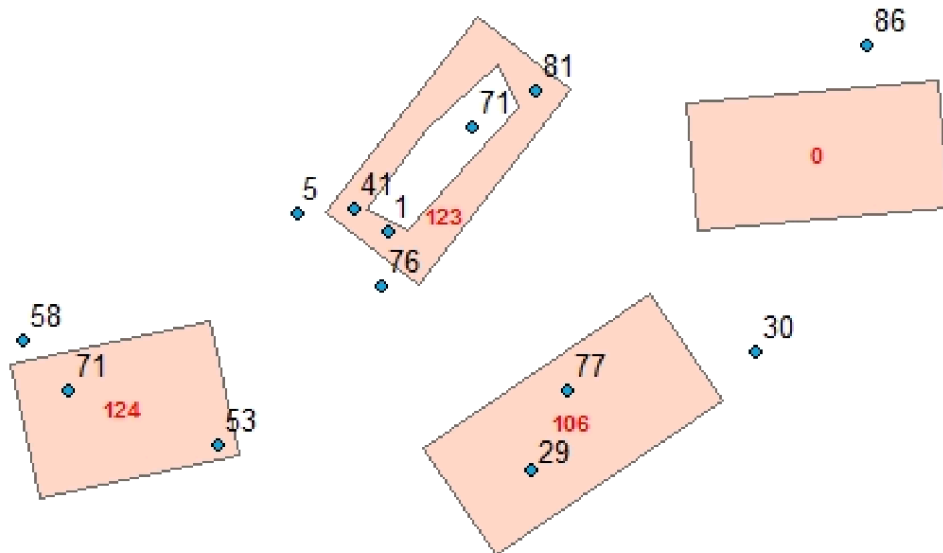


- Polyline to Point, Polyline To Polyline, Polyline To Multipoint - the target will get the attributes of the closest point
- Polygon to Point - the point will get the attributes of the polygon:
 - If the point is within the polygon - the polygon with largest distance to the boundary
 - If the point is not in any polygon - the closest polygon within the search tolerance.

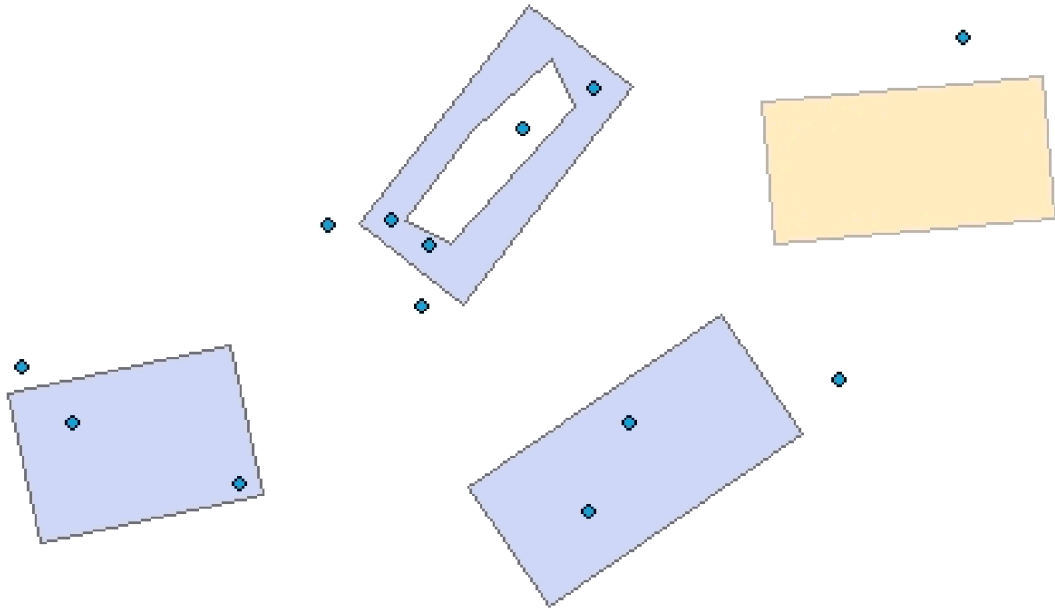
Example: Polygons to Points - Search Tolerance = 0 - Update Rule = "Sum"



- Polygon to Polyline - the polyline will get the attributes of the polygon with longest intersection. If the polyline does not intersect any polygon - the closest polygon within the search tolerance.
 - Polygon to Polygon - the polygon will get the attributes of the polygon with largest area of intersection. If the polygon does not intersect any source polygon - the closest polygon within the search tolerance.
 - Join Type = "Many To Many" - the target feature will get the attributes of all source features within the search tolerance according to the user specified update rule for each field to be joined.
- Example: Points to Polygons - Search Tolerance = 0



- Cut-off distance (Search Tolerance) - the maximum distance for which the attributes of the features in the Join layer will be joined to the target features (in the units of the spatial reference of the Target Dataset)
- Join Fields - the fields from the Join Layer that will be added to the Target layer. If the Join Type is "One To Many" - the values of the joined feature will be stored. If the Join Type is "Many To Many" - the values for each joined field will be added according to the user defined update rules.
- Keep All Target Features - if selected all features of the target layer will be exported to the output dataset. If not selected only the target features that were joined to the source features will be saved in the output. This option is only valid to "One To Many" Join Type. If the Join Type is "Many To Many", all features of the target dataset are preserved.
Example: Points to Polygons - Search Tolerance = 0 Keep All Target Features = FALSE.
The feature with no joins is missing in the output.



- Add statistics fields - used only if the Join Type is "Many To Many". Statistics for the Spatial Join will be added to the output attribute table depending on the type of the Target and Source datasets
 - [ET_Count] - the number of source features joined to the target feature.
 - [ET_CountIn] - if the target is polygon or polyline - the number of the joined features that intersect each target feature. Note that if the search tolerance is larger than 0 the number of the joined features might be larger than the number of intersecting features
 - [ET_LengthIn] - if the target is polygon and the source polyline - the sum of the length of the polylines intersecting the target polygon
 - [ET_AreaIn] - if the target and the source are polygons - the sum of the area of the polygons intersecting the target polygon

Outputs:

- New layer
- The attributes of the target layer will be preserved in the output dataset

- The join fields from the Source layer will be added to the output and the values will be populated as described above
- If Add statistics fields = TRUE, statistics fields will be added to the output and the values will be populated as described above

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SpatialJoin
<Target Dataset>	A String representing the target layer.
<Join Dataset>	A String representing the join layer.
<output dataset>	A String - the full name of the output layer.
< Join Type>	A String representing the Join Type to be used. Valid values: "OneToMany" and "ManyToMany"
<CutOff Distance>	A Double representing the Cut-Off Distance to be used. The units of the tolerance are the units of spatial reference of the Target Dataset
<Join Fields<	A String representing a list (separator ";") of the fields to transfer together with the method for each field. Valid values - "Sum", "Max", "Min" for number fields and "First", "Last" for string fields. Example: "Field1 Sum; Field2 First; Field3 Min"
{Keep All}	A Boolean indicated whether to keep all target features - see explanation above.
{Add Statistics}	A Boolean indicated whether to add statistics fields - see explanation above.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SpatialJoin", "Target Dataset", "Join Dataset", "output dataset", " Join Type", "CutOff Distance", "Join Fields", "Keep All", "Add Statistics"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SpatialJoin" "Target Dataset" "Join Dataset" "output dataset" "Join Type" "CutOff Distance" "Join Fields" "Keep All" "Add Statistics"</code>
.NET using ETGWOutX.dll	<code>SpatialJoin(Target Dataset,Join Dataset, output dataset, Join Type, CutOff Distance,Join Fields,Keep All,Add Statistics)</code>
ArcPy	<code>arcpy.SpatialJoin(Target Dataset, Join Dataset, output dataset, "Join Type", "CutOff Distance", "Join Fields","Keep All", "Add Statistics")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Near Feature

[Running programmatically](#)

Calculates the distance for each feature of the Input dataset to the closest feature from the Near dataset. In the attribute table of the output the distance is recorded together with the ID of the closest feature.

Inputs:

- Input Dataset - Point, Multipoint, Polyline or Polygon
- Near Dataset - Point, Multipoint, Polyline or Polygon
- Search tolerance - the maximum distance to search for features in the near layer in the units of the spatial reference of the Input Dataset

Outputs:

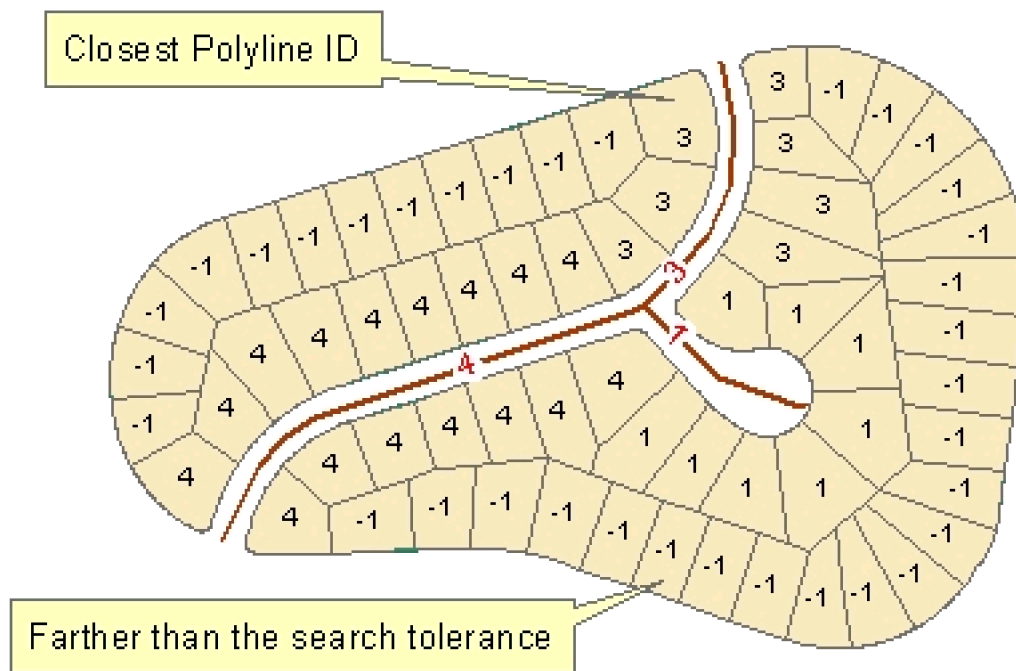
- New layer. The attribute table of the resulting feature class will have two new fields
 - [ET_Dist] - the distance from the input feature to the closest feature from the near layer
 - [ET_Closest] - the ID of the closest feature from the near layer

Notes:

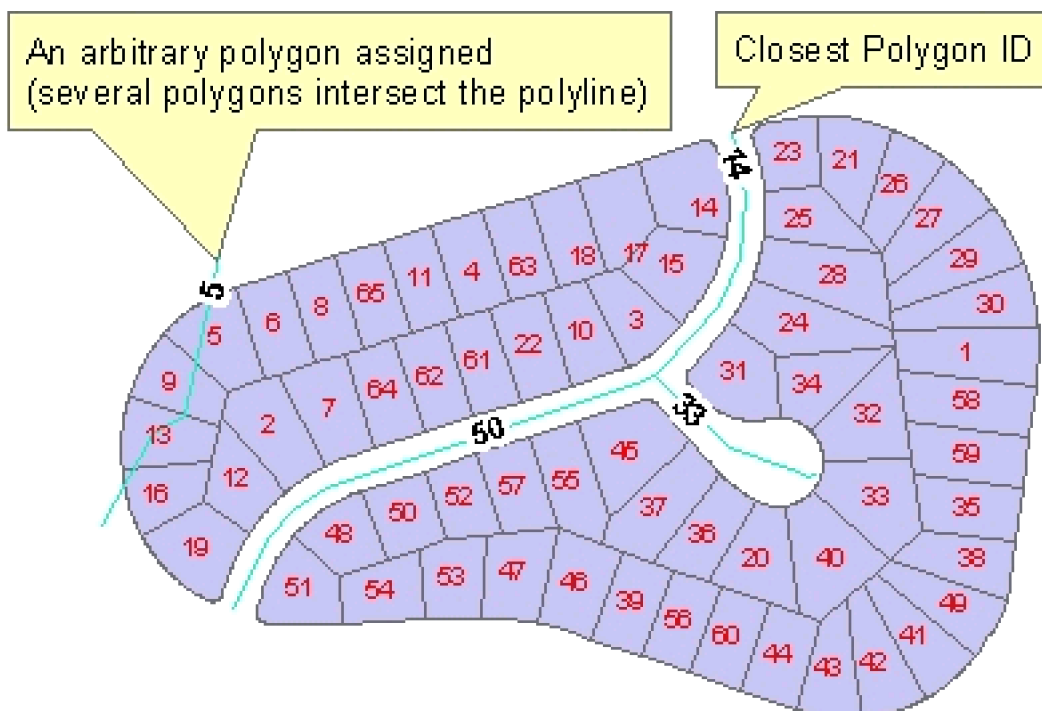
- If the distance from an input feature to the closest feature from the distance layer is larger than the Search Tolerance then the [ET_Dist] and [ET_Closest] will have a value of -1
- If an input feature intersects several features from the near dataset an arbitrary feature from the intersecting near features will be assigned as closest and the distance will be assigned to 0.
- If the input layer and the near layer have different Spatial References the distance is calculated in the Spatial Reference of the data input dataset.
- Keep Input Spatial Reference - If selected the output will have the spatial reference of the input dataset, else the spatial reference of the near dataset will be used

Examples:

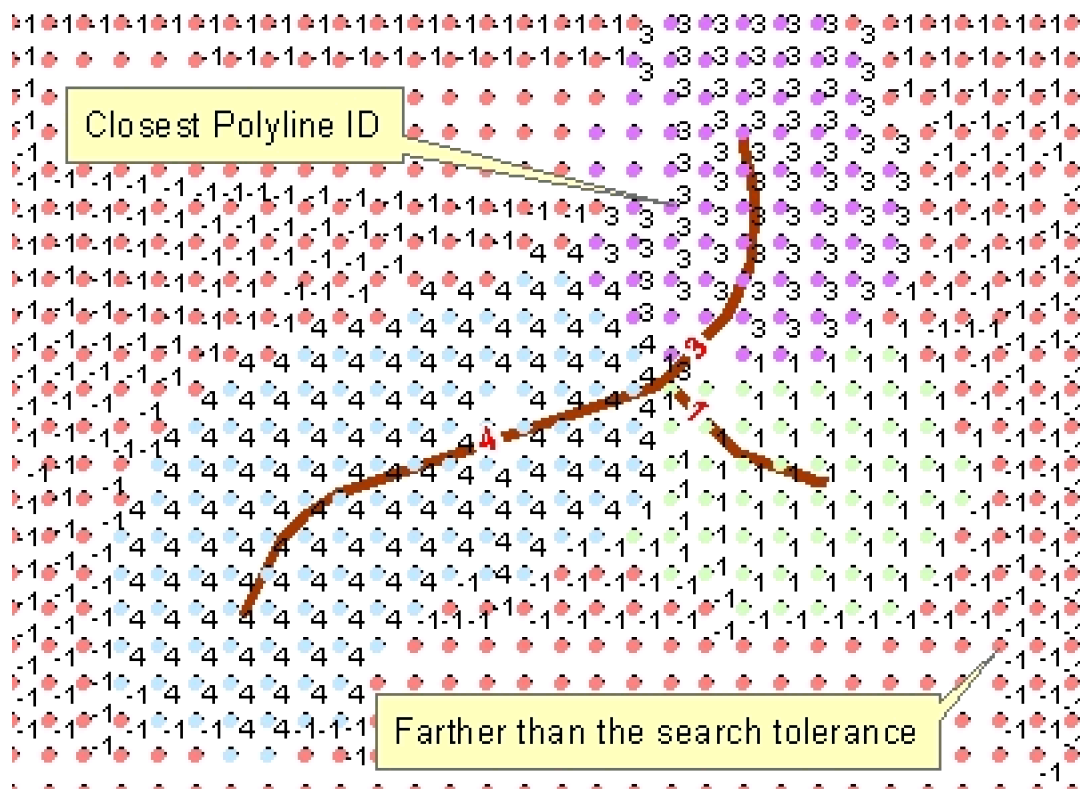
Input Dataset - Polygons ; Near Dataset - Polylines



Input Dataset - Polylines ; Near Dataset - Polygons



Input Dataset - Points ; Near Dataset - Polylines



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	NearFeature
<input dataset>	A String representing the input layer.
<Near Dataset>	A String representing the reference layer.
<output dataset>	A String - the full name of the output layer.
< CutOff Distance>	A Double representing the Cut-Off Distance to be used. The units of the tolerance are the units of spatial reference of the input dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the Near Dataset.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "NearFeature", "input dataset", "Near Dataset", "output dataset", " CutOff Distance", "KeepSourceSref"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "NearFeature" "input dataset" "Near Dataset" "output dataset" "CutOff Distance" "KeepSourceSref"</code>
.NET using ETGWOutX.dll	<code>NearFeature(input dataset,Near Dataset, output dataset, CutOff Distance, KeepSourceSref)</code>
ArcPy	<code>arcpy.NearFeature(input dataset, Near Dataset, output dataset, "CutOff Distance" , "KeepSourceSref")</code>

Notes:

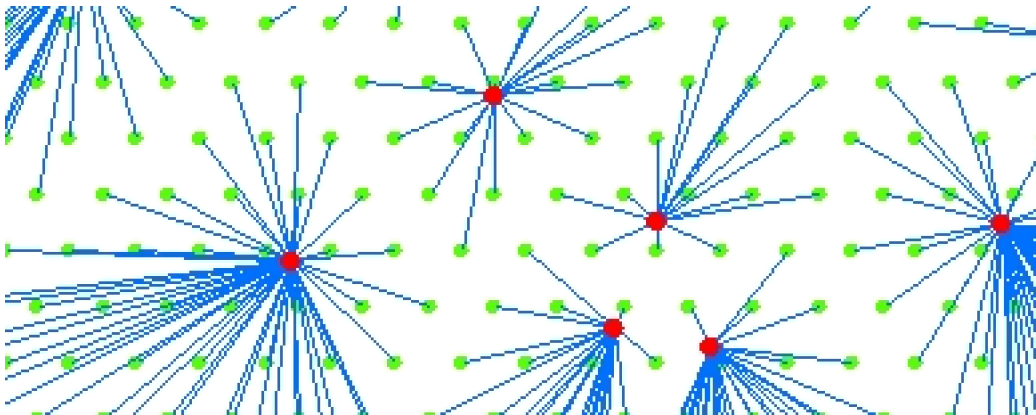
- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Spider Diagram

[Running programmatically](#)

Creates a polyline feature class representing the shortest distance between centers (point dataset) and Destinations (Point, Polyline or Polygon datasets). The Destinations are allocated to the closest Center.



Inputs:

- Point feature layer representing the Centers
- Point, Polyline or Polygon layer representing the destinations
- Cutoff distance - the maximum distance between a Center and a Destination to be used. Destinations that are further than this distance from any Center will not be assigned to a Center
- Output Spatial Reference from Centers or Destinations

Outputs:

- New Polyline feature class. The attribute table of the resulting feature class will have three new fields
 - [Center_ID] - the Feature ID of the Center point
 - [Dest_ID] - the Feature ID of the Destination feature
 - [ET_Dist] - the distance from the Center to the Destination

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SpiderDiagram
<Centers Dataset>	A String representing the input layer. Must be of Point type.
<Destinations Dataset>	A String representing the reference layer. Must be of Polyline type
<output dataset>	A String - the full name of the output layer.
< CutOff Distance>	A Double representing the Cut-Off distance to be used. The units of the tolerance are the units of spatial reference of the Centers Dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the Destinations Dataset.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SpiderDiagram", "Centers Dataset", "Destinations Dataset", "output dataset", " CutOff Distance", "KeepSourceSref"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SpiderDiagram" "Centers Dataset" "Destinations Dataset" "output dataset" "CutOff Distance" "KeepSourceSref"</code>
.NET using ETGWOutX.dll	<code>SpiderDiagram(Centers Dataset, Destinations Dataset, output dataset, CutOff Distance, KeepSourceSref)</code>

ArcPy

```
arcpy.SpiderDiagram(Centers Dataset, Destinations Dataset, output dataset,  
"CutOff Distance", "KeepSourceSref")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Spider Diagram Attribute Link

[Running programmatically](#)

Creates a Spider Diagram between the points of a Center Points dataset and the features in the destination layer (points, polygons, or polylines) based on the values in the link fields in both datasets. The created polylines will connect the Centers to the destination features if the values in the link fields are the same.

Inputs:

- Point dataset representing the Centers.
- Link field in the input point dataset.
- Point, Polyline or Polygon layer representing the destinations.
- Link field in the destination dataset.
- Optional - Depending on the type of the destination dataset the connector line is created between
 - Point - Center Point - Destination point
 - Multipoint - Center Point - Closest Destination point
 - Polyline - Center Point - Closest point on the polyline
 - Polygon - Center Point - Closest point on the polygon boundary
- Optional - Cutoff distance - the maximum distance between a Center and a Destination to be used. Destinations that are further than this distance from any Center will not be connected to a Center

Outputs:

- New Polyline feature class with single segmented polylines. The attributes of the centers dataset will be preserved

Notes:

- The spatial references of both input dataset must have the same geographic coordinate system.
- The output spatial reference is the one of the Centers dataset if Keep Centers Sref is TRUE

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SpiderDiagramAttributeLink
<Centers Dataset>	A String representing the input layer. Must be of Point type.
<Destinations Dataset>	A String representing the reference layer. Must be of Polyline type
<output dataset>	A String - the full name of the output layer.
<Centers Link>	A String - the name of the link field from the Centers Dataset.
<Destinations Link>	A String - the name of the link field from the Destinations Dataset.
< CutOff Distance>	A Double representing the Cut-Off distance to be used. The units of the tolerance are the units of spatial reference of the Centers Dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the Destinations Dataset.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPath, "SpiderDiagramAttributeLink", "Centers Dataset", "Destinations Dataset", "output dataset", "Centers Link" "Destinations Link" "CutOff Distance", "KeepSourceSref"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SpiderDiagramAttributeLink" "Centers Dataset" "Destinations Dataset" "output dataset" "Centers Link" "Destinations Link" "CutOff Distance" "KeepSourceSref"</code>
.NET using ETGWOuX.dll	<code>SpiderDiagramAttributeLink(Centers Dataset, Destinations Dataset, output dataset, Centers Link , Destinations Link, CutOff Distance, KeepSourceSref)</code>
ArcPy	<code>arcpy.SpiderDiagramAttributeLink(Centers Dataset, Destinations Dataset, output dataset, "Centers Link", "Destinations Link", "CutOff Distance", "KeepSourceSref")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Allocation

[Running programmatically](#)

Allocates a set of demand points (Customers) to user specified number of supply points (Facilities) out of a Facilities point dataset based on the Euclidian distance between the Customers and Facilities. In other words the function selects N Facilities out of K candidates to service a set of M Customer locations in such a way that each Customer is allocated to a single Facility (based on Euclidean distance) and the total distance between the Customers and selected Facilities is minimized.

The function uses heuristic vertex substitution algorithm modified from Teitz and Bart (1968) and can handle comparatively large problems (Number of Customers * Number Facilities < 5 Million)

Inputs:

- Point feature layer representing the Facilities (Centers).
- Facility name field (optional) - the values in this field are used to identify the facilities. If the field is not specified the FID will be used as a name
- Facility type field (optional) - the values of this field indicate whether a specific facility must be included in the selected set of facilities. Values of "1", "Required", "Existing" will force the inclusion of the Facility in the selected set of facilities. If the field is not specified all facilities will be considered as equal in the selection algorithm.
- Point feature layer representing the customers (demand points) that need to be allocated to the facilities.
- Customer name field (optional) - the values in this field are used to identify the facilities. If the field is not specified the FID will be used as a name
- Number of facilities to be selected.
- Cutoff distance (optional) - the maximum distance between a Facility and a Customer to be used. Note that some customers might not be allocated if too small cutoff distance is used.

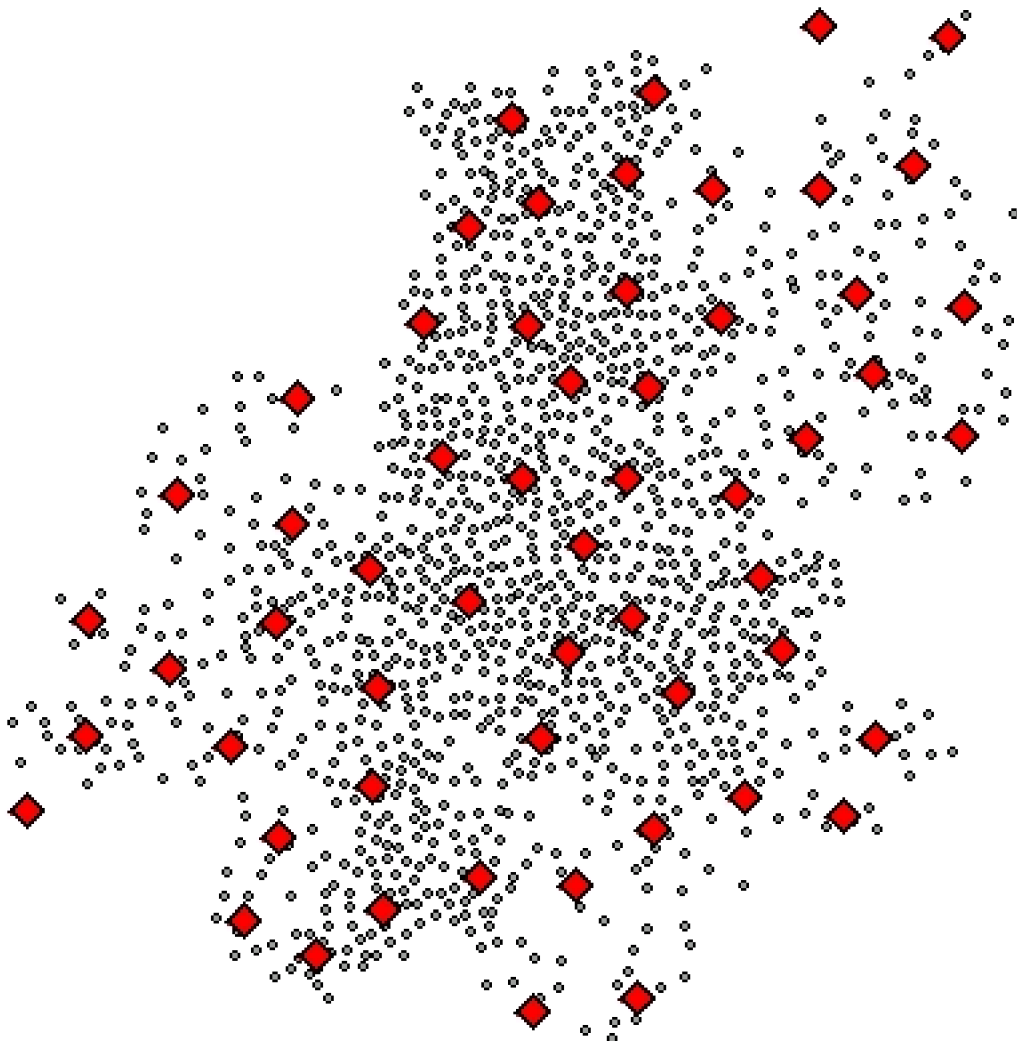
Outputs:

- New Point layer containing only the selected facilities. The attribute table of the resulting feature class will have the following fields
 - FacilityID - The original FID of the selected facility

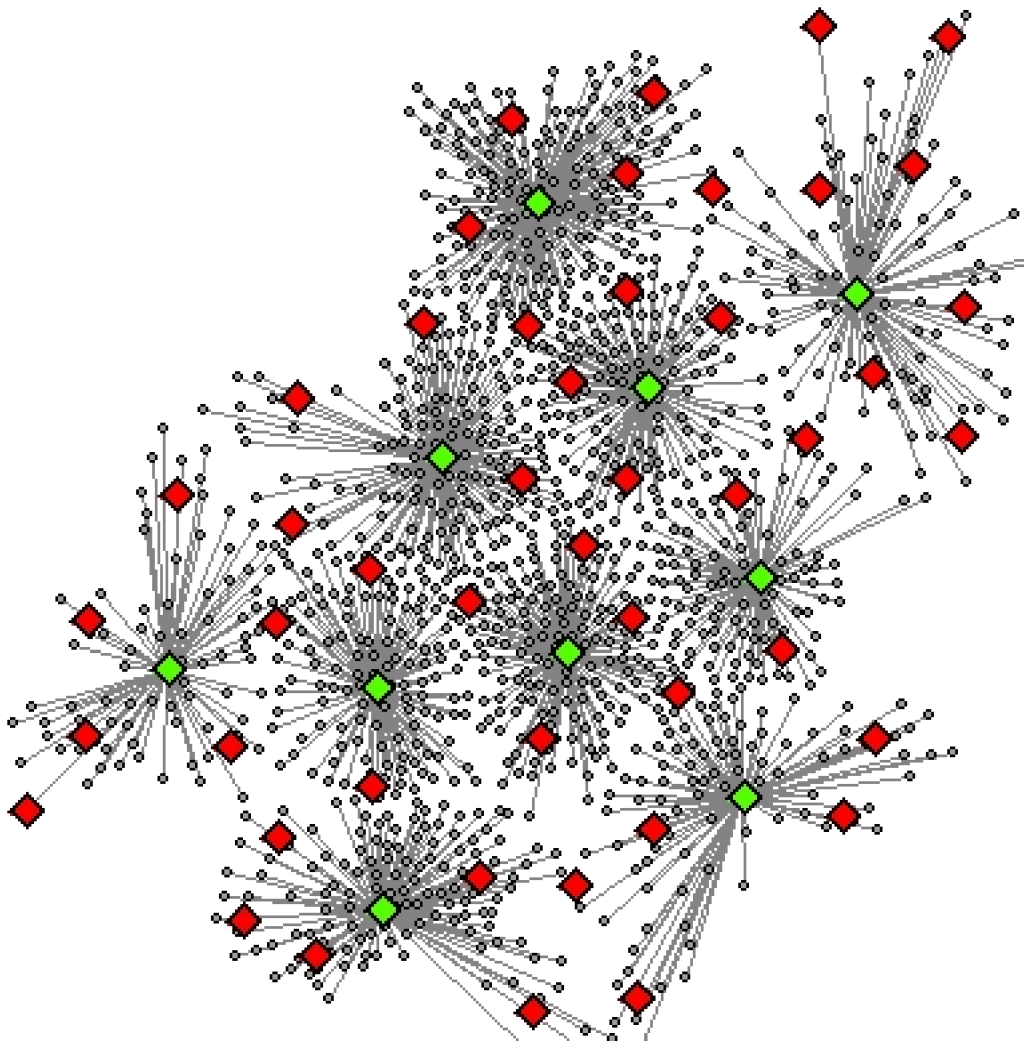
- Facility - The value in the user specified Name field of the selected facility
 - Type - the type of the facility - Selected or Fixed (if the facility was indicated as fixed in the input facilities dataset.
 - Num_Alloc - Number of customers allocated to this facility
 - Max_Dist - The distance to the farthest customer from this facility.
 - Total_Dist - The sum of the distances to all allocated cutomers.
- New Polyline feature class with lineslinking selected facilities to allocated to them customers. The attribute table of the resulting feature class will have the following fields
 - FacilityID - The original FID of the selected facility
 - CustomerID - The original FID of the customer
 - Facility - The value in the user specified Name field of the selected facility
 - Customer - The value in the user specified Name field of the customer
 - ET_Dist - The distance between the selected facility and the allocated customer

Illustration:

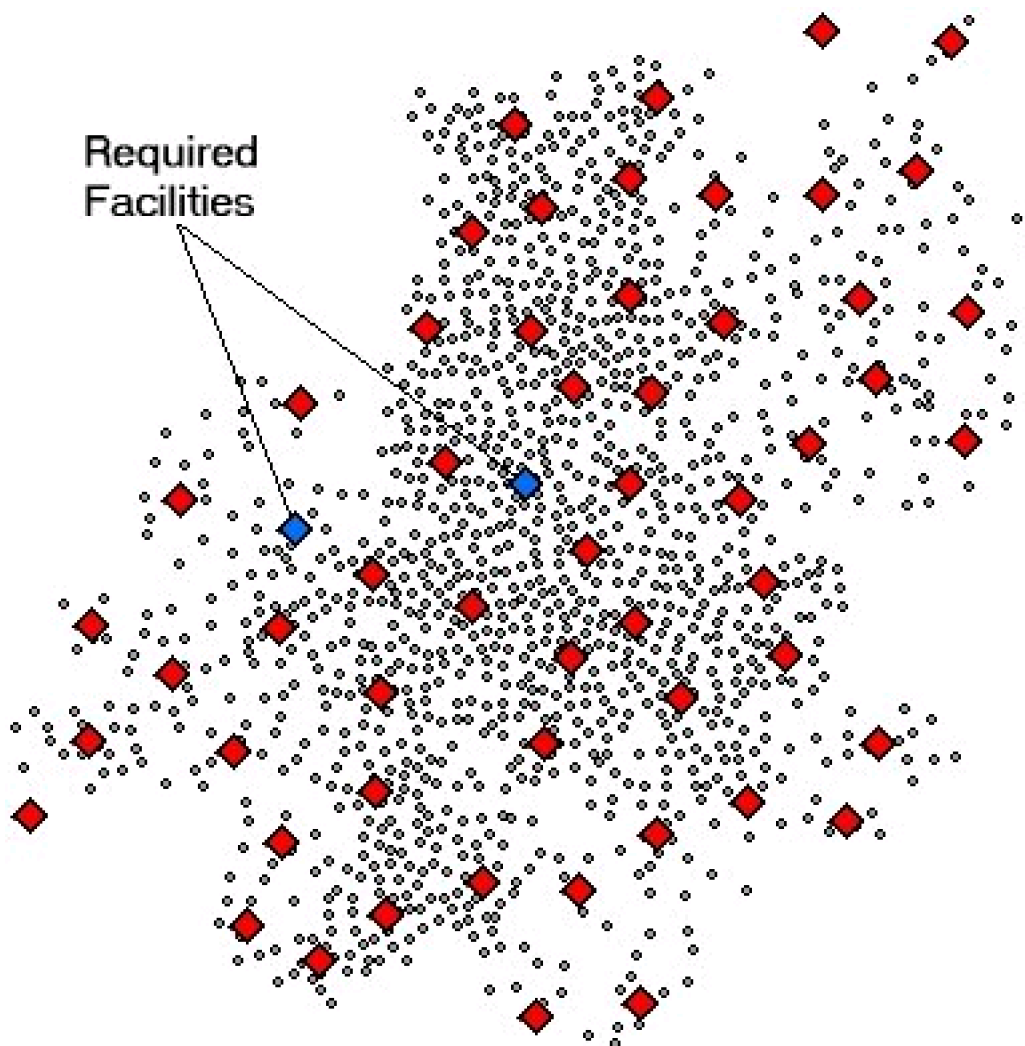
- Input Facilities and Customers - No required facilities



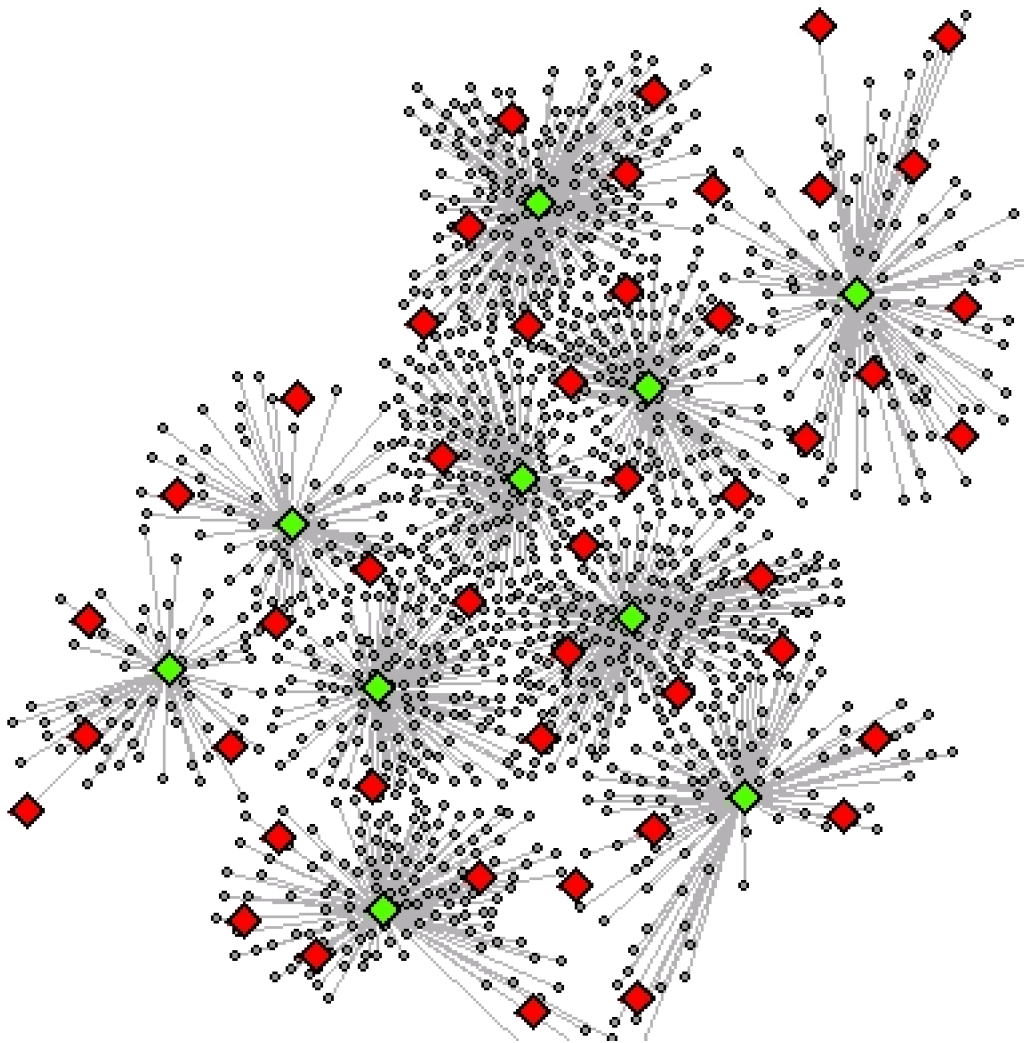
- Result (Selected facilities in green)



- Input Facilities and Customers - Two required facilities



- Result (Selected facilities in green)

**Notes:**

- The output spatial reference will be the one of the Facilities dataset
- The function has a restrictions and should not be applied if Number of Customers * Number Facilities > 5 Million

References:

- M.B. Teitz and P. Bart, Heuristic methods for estimating the generalized vertex median of a weighted graph. Gpns. Res. 16, 955-961 (1968).

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	Allocate
<facilities dataset>	A Point layer - the candidate facilities.
<customers dataset>	A Point feature class or feature layer - the Customers (demand points).
<out link layer>	A String - the full name of the output link layer
<out facilities layer>	A String - the full name of the output selected facilities feature class
<number facilities>	An integer - the number of facilities to be selected
{facility name field}	A String representing a field name - the values in this field are used to identify the facilities.
{facility type field}	A String representing a field name - the values in this field are used to identify the type of the facilities.
{customer name field}	A String representing a field name - the values in this field are used to identify the customers.
{Cutoff distance}	A number - the maximum distance between a Facility and a Customer to be used. The units of the tolerance are the units of spatial reference of the facilities dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the customers dataset.
{Keep Customers Sref}	A Boolean indicating whether the output to have the spatial reference of the customers layer. If False or 0, the spatial reference of the facilities layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<pre>subprocess.call([ETGWPath, "Allocate", "facilities dataset", "customers dataset", "out link layer", "out facilities layer", "number facilities", "facility name field", "facility type field", "customer name field", "Cutoff distance", "KeepSourceSref"])</pre>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "Allocate" "facilities dataset" "customers dataset" "out link layer" "out facilities layer" "number facilities" "facility name field" "facility type field" "customer name field" "Cutoff distance" "KeepSourceSref"</pre>
.NET using ETGWOutX.dll	<pre>Allocate(facilities dataset,customers dataset, out link layer,out facilities layer, number facilities, facility name field, facility type field, customer name field, Cutoff distance KeepSourceSref)</pre>
ArcPy	<pre>arcpy.Allocate(facilities dataset,customers dataset, out link layer,out facilities layer, number facilities, facility name field, facility type field, customer name field, Cutoff distance KeepSourceSref)</pre>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Build Thiessen Polygons

[Running programmatically](#)

Builds Thiessen Polygons from a feature layer

Thiessen (Voronoi) polygons define individual areas of influence around each of a set of points. Thiessen polygons are polygons whose boundaries define the area that is closest to each point relative to all other points. They are mathematically defined by the perpendicular bisectors of the lines between all points

Inputs:

- A feature layer (Point, Multipoint, Polyline, Polygon)

Outputs:

- New polygon layer.
 - If the Attach attributes option is selected, the attributes of the source features are transferred to the new attribute table.

Notes :

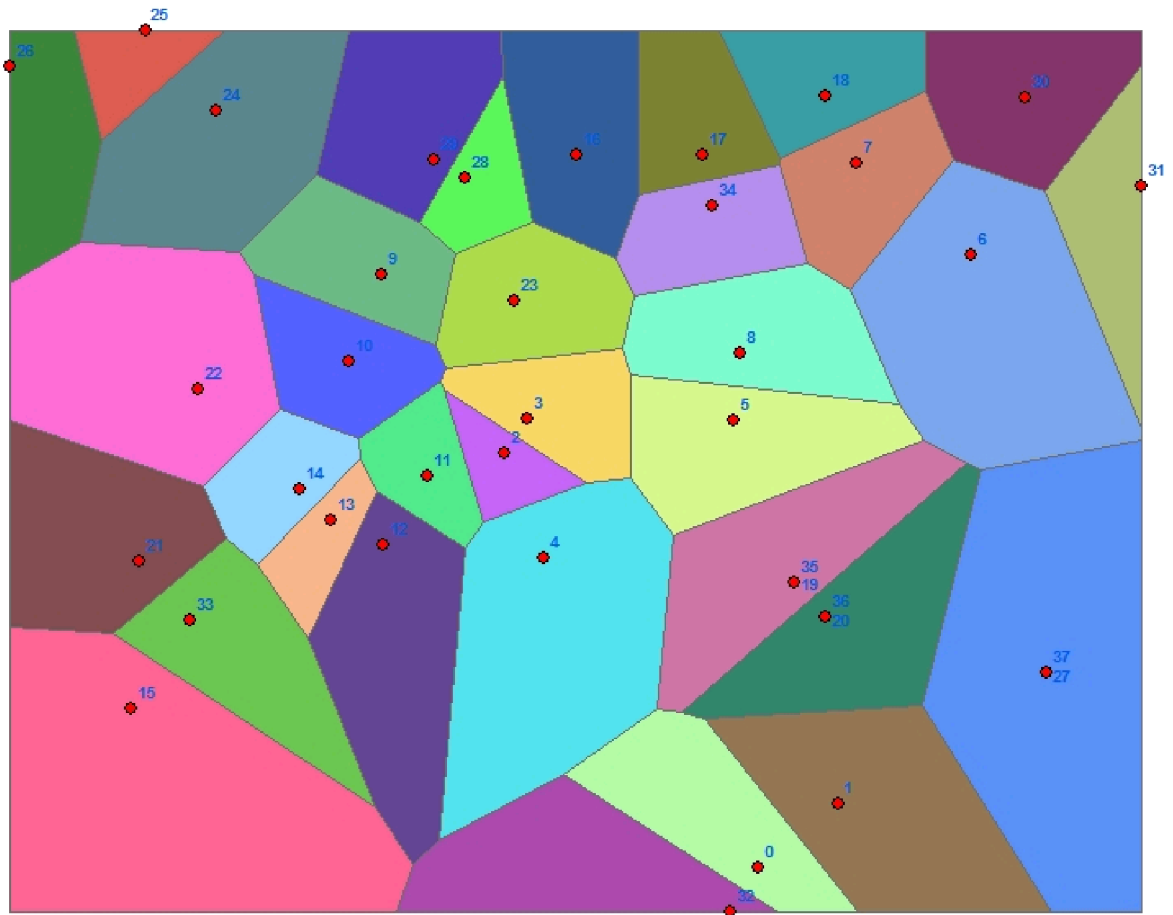
- By default the Thiessen polygons are clipped to the extents of the input features. There is an option to buffer the extents rectangle before clipping with it.
- The resulting feature class can be clipped (Clip Layer Wizard) with any polygon layer to match the shape of this layer.
- The function should work with no problems on datasets with up to 6 million points.

Examples of use:

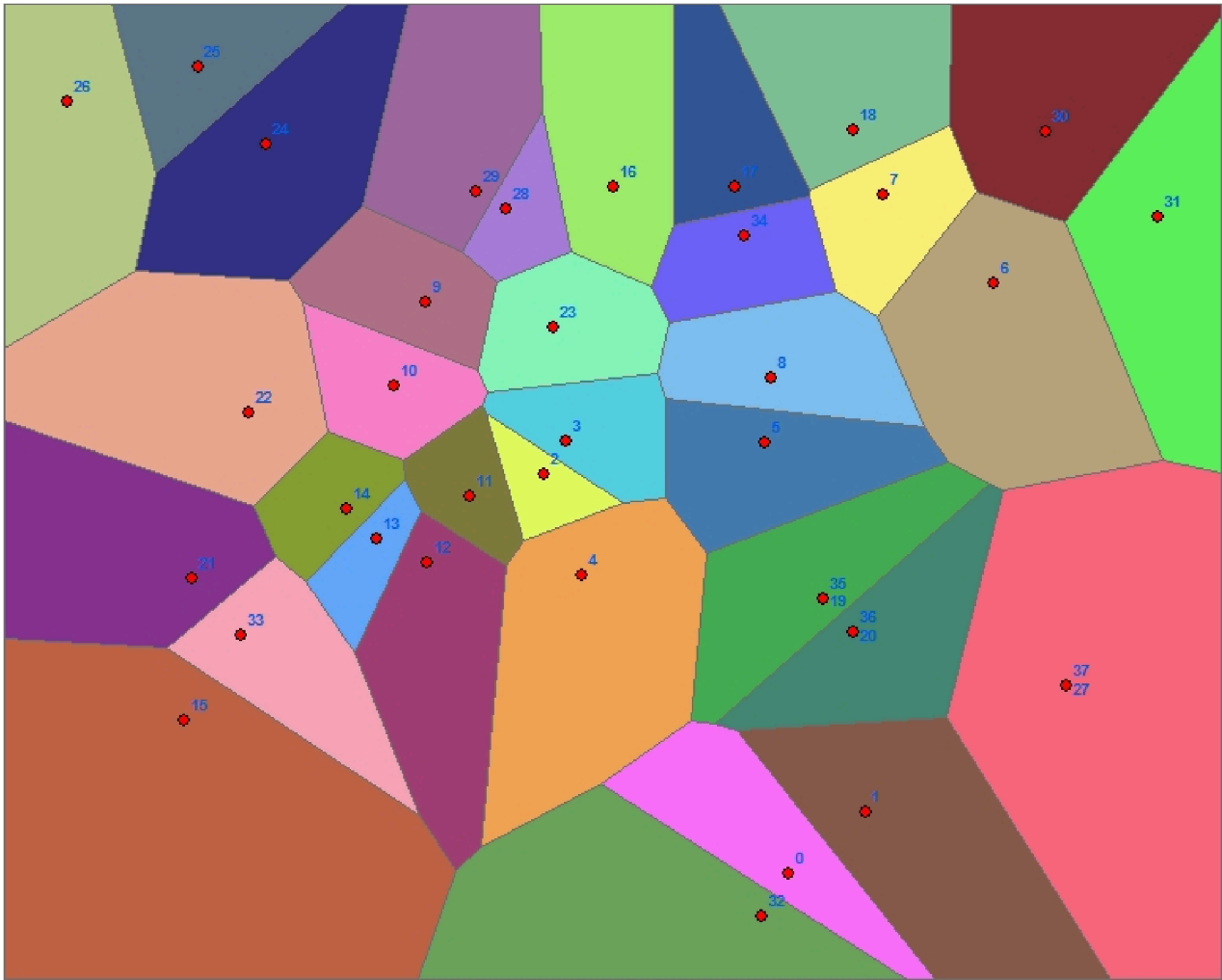
- Defining trade areas
- From a set of soil sampling points to define non overlapping polygons for each soil type

Example:

Thiessen - Buffer Distance = 0



Thiessen - Buffer Distance > 0



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	BuildThiessen
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Buffer Distance}	A Double representing the distance to buffer the input extent (in the units of the spatial reference of the input dataset).

{Add Attributes}	A Boolean that indicates whether the attributes of the input features are to be transfered to the thiessen polygons.
------------------	--

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "BuildThiessen", "input dataset", "output dataset", "Buffer Distance", "Add Attributes"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "BuildThiessen" "input dataset" "output dataset" "Buffer Distance" "Add Attributes"</pre>
.NET using ETGWOutX.dll	<code>BuildThiessen(input dataset, output dataset, Buffer Distance, Add Attributes)</code>
ArcPy	<code>arcpy.BuildThiessen(input dataset, output dataset, "Buffer Distance", "Add Attributes")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

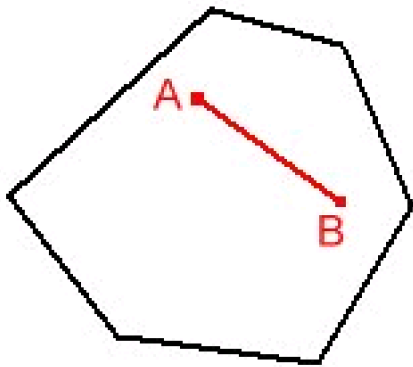
Create Concave Hull

[Running programmatically](#)

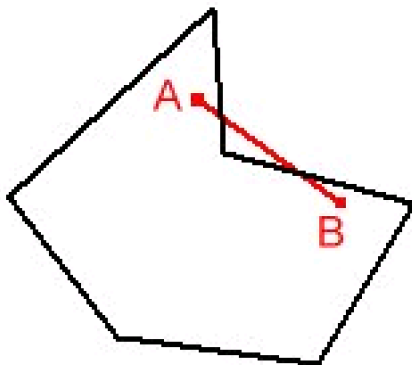
The Concave Hull function creates a polygon that represents the area occupied by a set of data points.

- The resulting polygon might be concave or convex

Convex

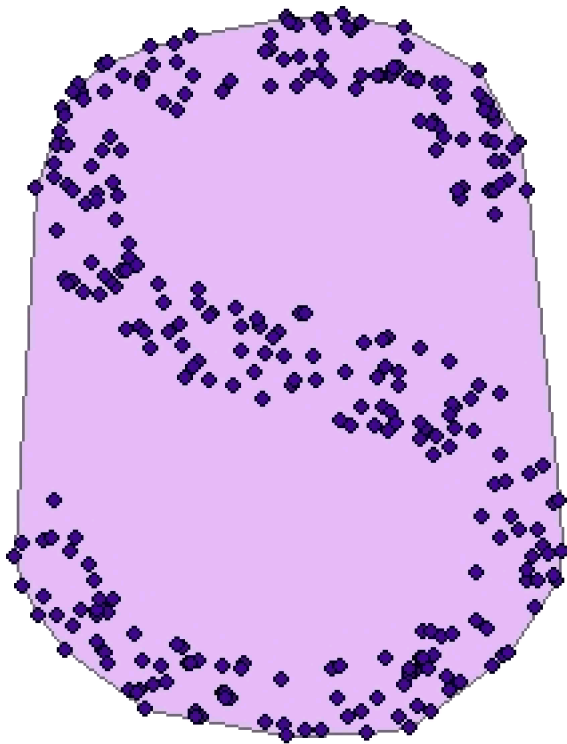


Non Convex (Concave)

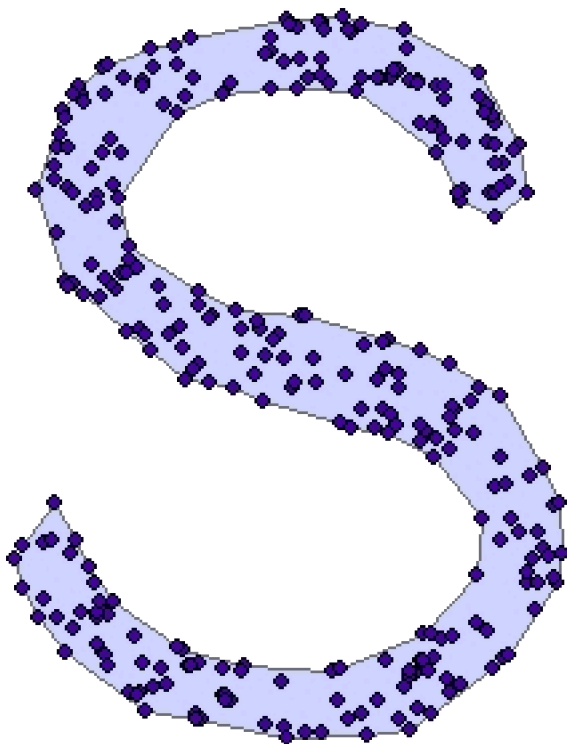


- A Concave hull describes better the shape of the point cloud than the convex hull

Convex Hul



Concave Hull



- Many solutions are possible for the same input data. The result depends on the user defined distance threshold. The larger the threshold, the closer the resulting polygon will be to the Convex Hull.

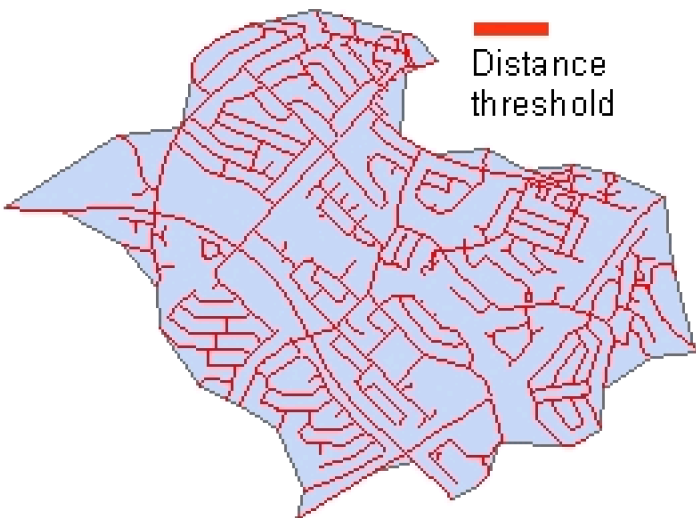
Source Data



Convex Hull



Concave Hull 1



Concave Hull 2



Inputs:

- A feature layer (Point, Multipoint, Polyline, Polygon)
- Distance threshold - in the units of the spatial reference of the input dataset

Outputs:

- New polygon layer.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CreateConcaveHull
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Distance Threshold>	A Double representing the threshold for creating a concave hull - in the units of the spatial reference of the input dataset.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CreateConcaveHull", "input dataset", "output dataset", "Distance Treshold"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateConcaveHull" "input dataset" "output dataset" "Distance Treshold"</code>
.NET using ETGWOuX.dll	<code>CreateConcaveHull(input dataset, output dataset, Distance Treshold)</code>
ArcPy	<code>arcpy.CreateConcaveHull(input dataset, output dataset, "Distance Treshold")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

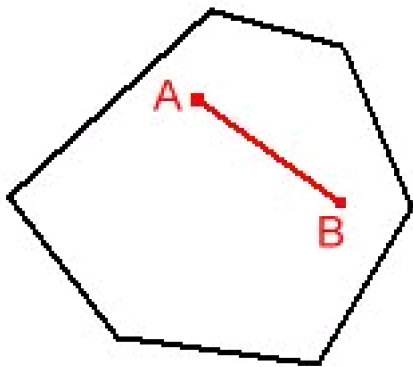
Build Convex Hull

[Running programmatically](#)

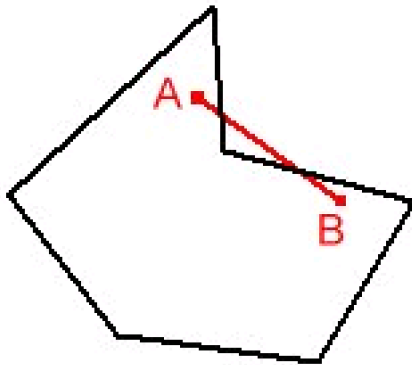
Builds the Convex Hull of the features of a layer

Convex hull is a polygonal area that is of smallest length and so that any pair of points within the area have the line segment between them contained entirely inside the area.

Convex



Non Convex (Concave)



Defining the convex Hull of a set of points is useful, for example in the case of enclosing the points, using a fence of shortest total length.

Source Data

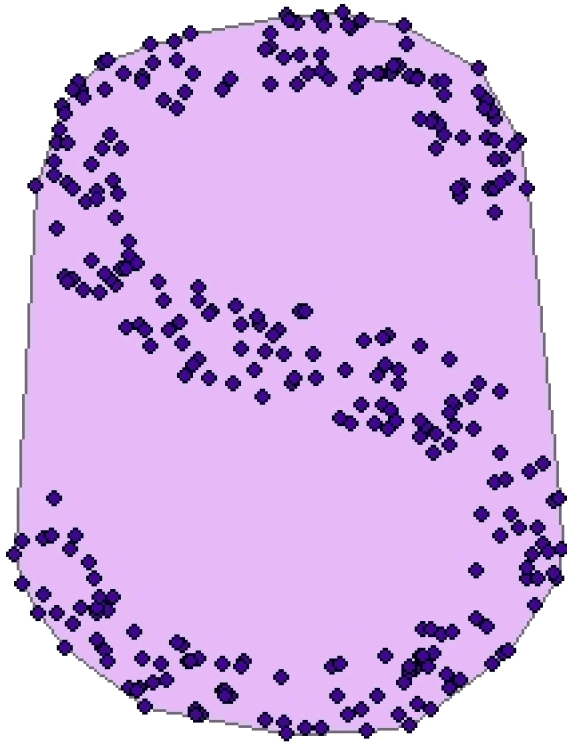


Convex Hull

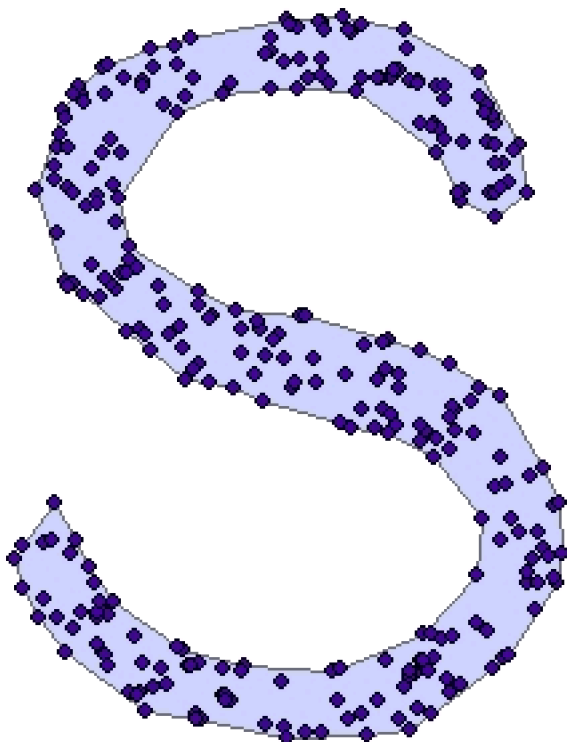


While in general the Convex Hull is good to describe the shape of the input data points, in many cases a polygon that describes better the region occupied by the point cloud is needed. See the [Create Concave Hull](#) function

Convex Hull



Concave Hull



Inputs:

- A feature layer (Point, Multipoint, Polyline, Polygon)

Outputs:

- New polygon layer.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ConvexHull
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ConvexHull", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "ConvexHull" "input dataset" "output dataset"</code>
.NET using ETGWOuX.dll	<code>ConvexHull(input dataset, output dataset)</code>
ArcPy	<code>arcpy.ConvexHull(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Transfer Polyline Attributes

[Running programmatically](#)

Transfers the attributes from one polyline layer to another based on the spatial relations of the polylines. Suitable for transferring the attributes between two polyline datasets that represent the same phenomenon - for example between existing street dataset with complete attributes and newly captured (spatially better) data for the same location but with no attributes.

Inputs:

- Target polyline layer.
- Polyline layer source for the attribute data.
- Search tolerance.
- Fields to be transferred from the source to the target dataset.

Outputs:

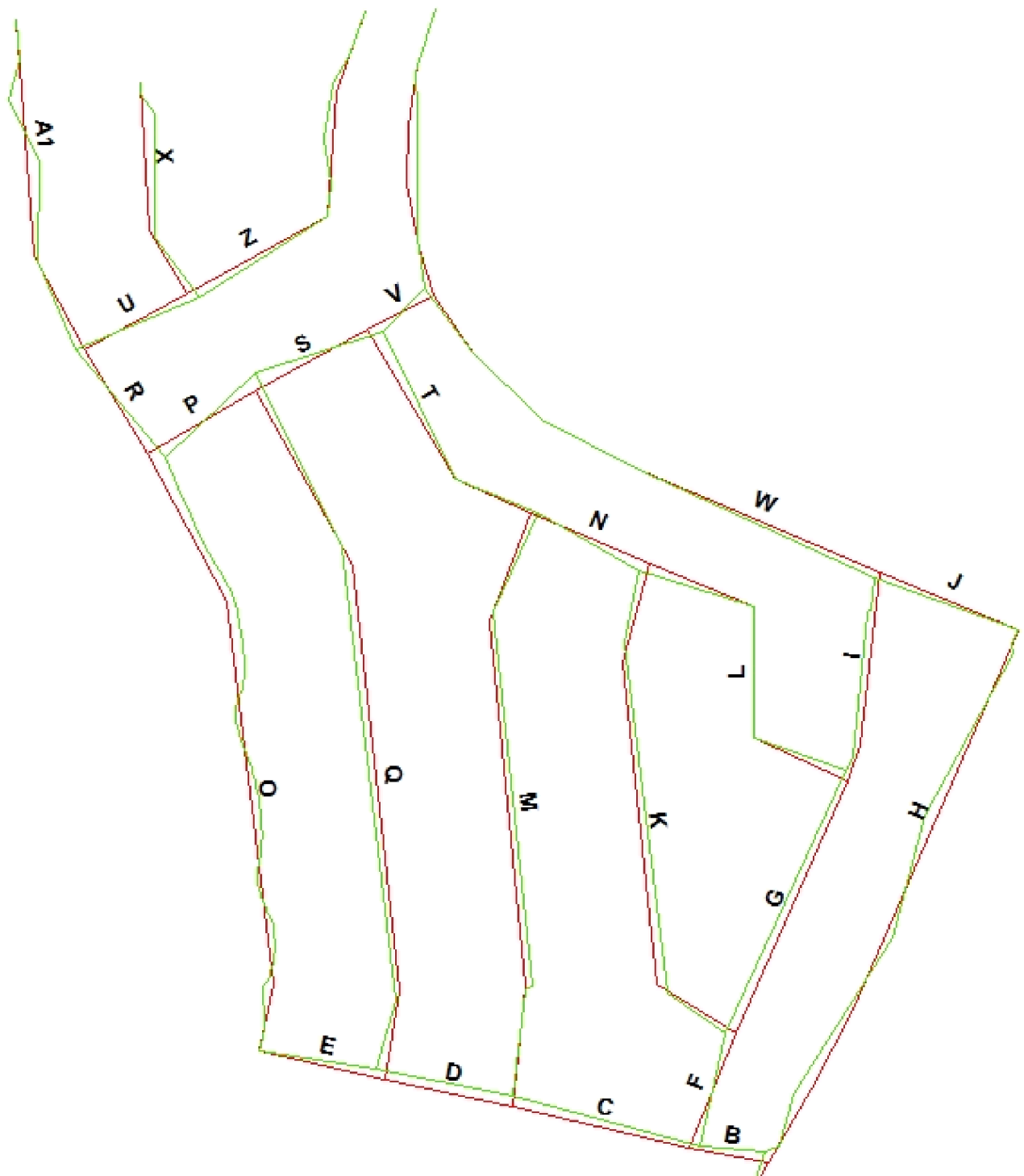
- New Polyline feature class

Notes:

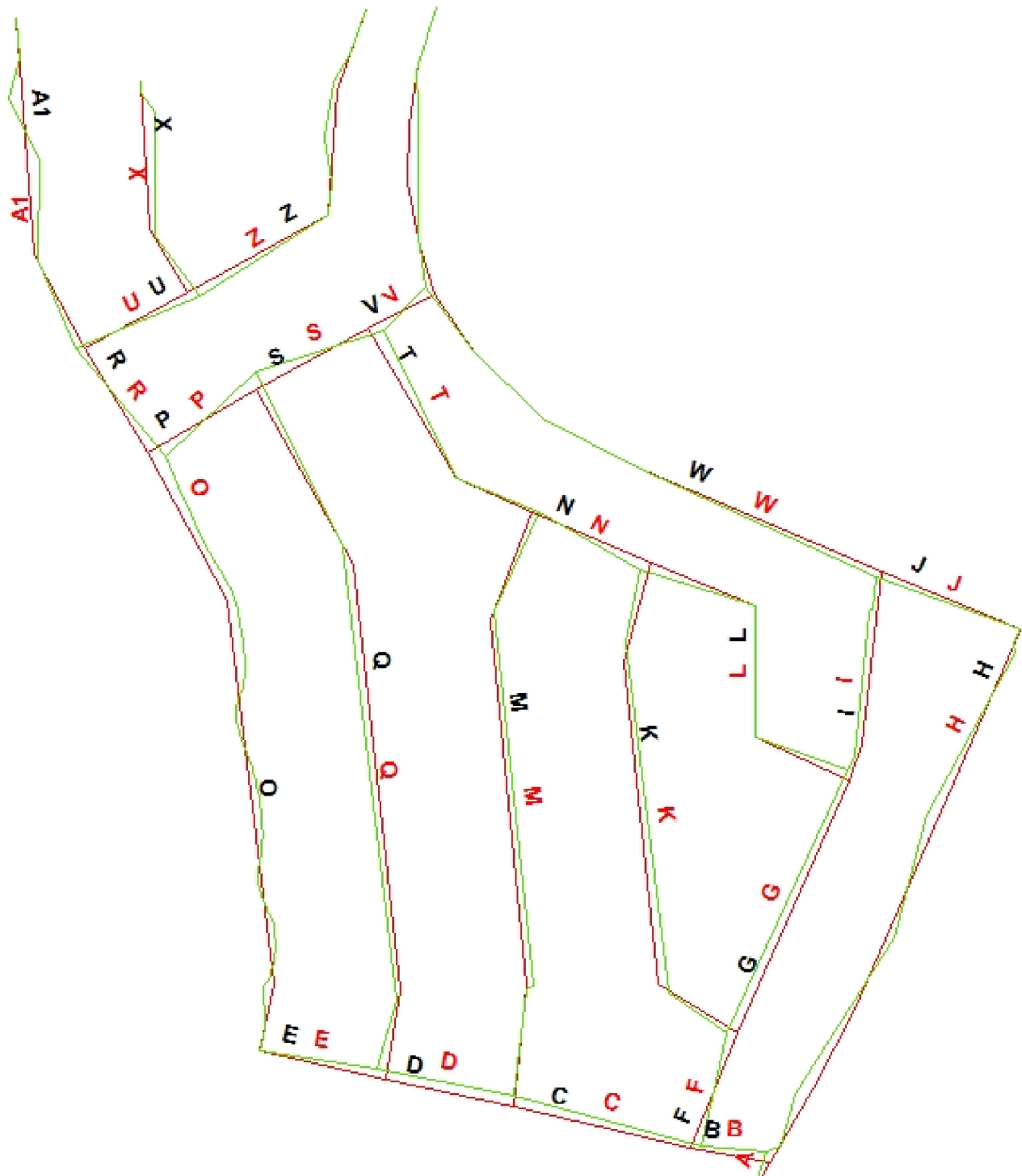
- The spatial references of both input dataset must have the same geographic coordinate system.
- The output spatial reference is the one of the Target dataset

Example:

Source Dataset -red and Target Dataset - green



Source Dataset -red - black labels and Output Dataset - green - red labels



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function	TransferPolylineAttributes

Name	
<Target Dataset>	A String representing the target layer. Must be of Point type.
<Source Datasett>	A String representing the source layer. Must be of Polyline type
<output dataset>	A String - the full name of the output layer.
<Search Tolerance>	A Double representing the serch tolerance in the units of the spatial reference of the target dataset.
{Transfe Fields}	A String - representing a list (separator - ";") of field names to be transfered. If missing - all attributes of the source dataset will be transfered to the target.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "TransferPolylineAttributes", "Target Dataset", "Source Datasett", "output dataset", "Search Tolerance" "Transfe Fields"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "TransferPolylineAttributes" "Target Dataset" "Source Datasett" "output dataset" "Search Tolerance" "Transfe Fields"</pre>
.NET using ETGWOuX.dll	<code>TransferPolylineAttributes(Target Dataset,Source Datasett, output dataset, Search Tolerance ,Transfe Fields)</code>
ArcPy	<code>arcpy.TransferPolylineAttributes(Target Dataset, Source Datasett, output dataset, "Search Tolerance", "Transfe Fields")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Find Closest Point

[Running programmatically](#)

Calculates the distance for each point of a point dataset to the closest point from the same dataset. The function produces similar results as the [Closest Feature Distance](#), but uses a robust algorithm and can be applied on datasets containing up to 8 million points.

Inputs:

- A Point feature layer
- Cutoff distance - the maximum distance to search for neighbor points.

Outputs:

- A new Point feature class. The attribute table of the resulting feature class will have three new fields
 - [ET_ID] - the ID of the feature
 - [ET_Dist] - the distance from the point to the closest point.
 - [ET_Closest] - the ID of the closest point.

Notes:

- If the distance from a point to the closest point is larger than the Cutoff distance then the [ET_Dist] will have a value of 0 and [ET_Closest] will have a value of -1
- If there are coincident points in the input dataset, only one of the coincident point will be assigned a closest neighbor. The other points in the same location will have values ET_Dist = 1 and ET_Closest = -1
- The bigger the search tolerance is, the slower the process will be
- The distance is calculated in the units of the Spatial Reference of the input dataset

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FindClosestPoint
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<CutOff Distance>	A Double representing the maximum distance between two points to be considered neighbors - in the units of the spatial reference of the input dataset.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FindClosestPoint", "input dataset", "output dataset", "CutOff Distance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "FindClosestPoint" "input dataset" "output dataset" "CutOff Distance"</code>
.NET using ETGWOuX.dll	<code>FindClosestPoint(input dataset, output dataset,CutOff Distance)</code>
ArcPy	<code>arcpy.FindClosestPoint(input dataset, output dataset,CutOff Distance)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)

- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Connect To Closest Point

[Running programmatically](#)

Creates an output polyline layer with single segmented polylines that connect each point of the input Point layer to it's closest neighbor. The function uses a robust algorithm and can be applied on datasets containing up to 2 million points.

Inputs:

- A Point feature layer
- Cutoff distance - the maximum distance to search for neighbor points.
- Add Duplicate Links option (see notes below)

Outputs:

- A new Polyline layer. The attribute table of the resulting layer will have three new fields
 - [ET_From] - the ID of the FROM point
 - [ET_To] - the ID of the TO point
 - [ET_Dist] - the distance from the point to the closest point

Notes:

- If the distance from a point to the closest point is larger than the Cutoff distance then no link will be created between the two points
- If there are coincident points in the input dataset, the duplicates will be ignored.
- The direction of the resulting polyline is always from the evaluated point to the closest point found.
- If there are 2 points "A" and "B" where point "B" is the closest neighbor of point "A" and point "A" is the closest neighbor of point "B"
 - If the Add Duplicate Links option is selected, the resulting feature class will have 2

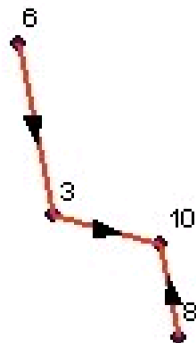
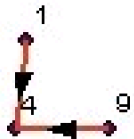
duplicate links - one from "A" to "B" and one from "B" to "A"

- If the Add Duplicate Links option is NOT selected, then only one of the two links will be stored in the output.

Examples:

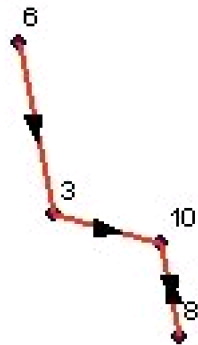
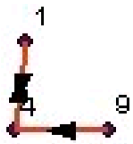
Result Dataset - Add Duplicate Links option NOT selected

- The closest point to point 8 is point 10.
- The closest point to point 10 is point 8
- Only one link is added - from point 8 to point 10



Result Dataset - Add Duplicate Links option NOT selected

- The closest point to point 8 is point 10.
- The closest point to point 10 is point 8
- Two coincident links are added - from point 8 to point 10 and from point 10 to point 8



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ConnectToClosestPoint
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<CutOff Distance>	A Double representing the CutOff Distance(in the units of the spatial reference of the input dataset).
{Add Duplicates}	A Boolean indicating whether duplicate links will be added to the result.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ConnectToClosestPoint", "input dataset", "output dataset", "CutOff Distance", "Add Duplicates"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ConnectToClosestPoint" "input dataset" "output dataset" "CutOff Distance" "Add Duplicates"</pre>
.NET using ETGWOuX.dll	<code>ConnectToClosestPoint(input dataset, output dataset, CutOff Distance, Add Duplicates)</code>
ArcPy	<code>arcpy.ConnectToClosestPoint(input dataset, output dataset, "CutOff Distance", "Add Duplicates")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Connect Unstructured Points

[Running programmatically](#)

Connects each point of a point dataset to its closest neighbors to create polylines. The function does not require attributes that define which points should pertain to a single polyline or order of the points within the polylines (if your point data has such attributes use the [Point To Polyline](#) function instead). The function uses a robust algorithm and can be applied on datasets containing up to 2 million points.

Inputs:

- A Point feature layer
- Cutoff distance - the maximum distance to search for neighbor points.
- Avoid Loops option (see notes and examples below)

Outputs:

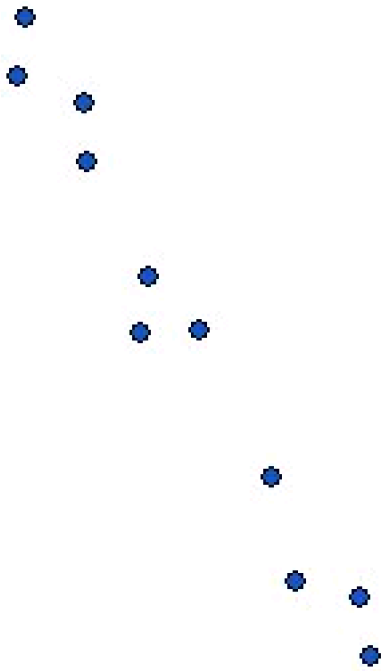
- A new Polyline feature class.

Notes:

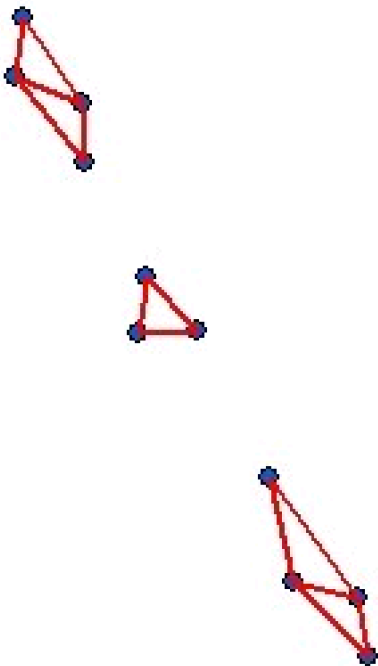
- If the distance from a point to the closest point is larger than the Cutoff distance then no link will be created between the two points
- If there are coincident points in the input dataset, the duplicates will be ignored.
- If the Avoid Loops option is not selected each point will be connected to its 2 closest neighbors (provided that the distance between the point and the neighbors is less than the Cutoff distance)
- If the Avoid Loops option is selected, the function will try to create longest non intersecting polyline possible.

Examples:

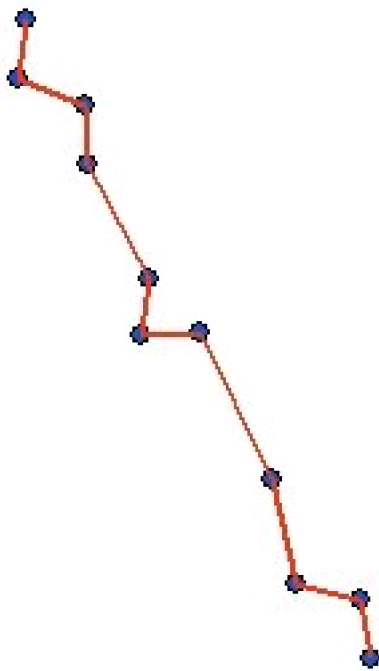
Source Points



Result - Avoid Loops = False

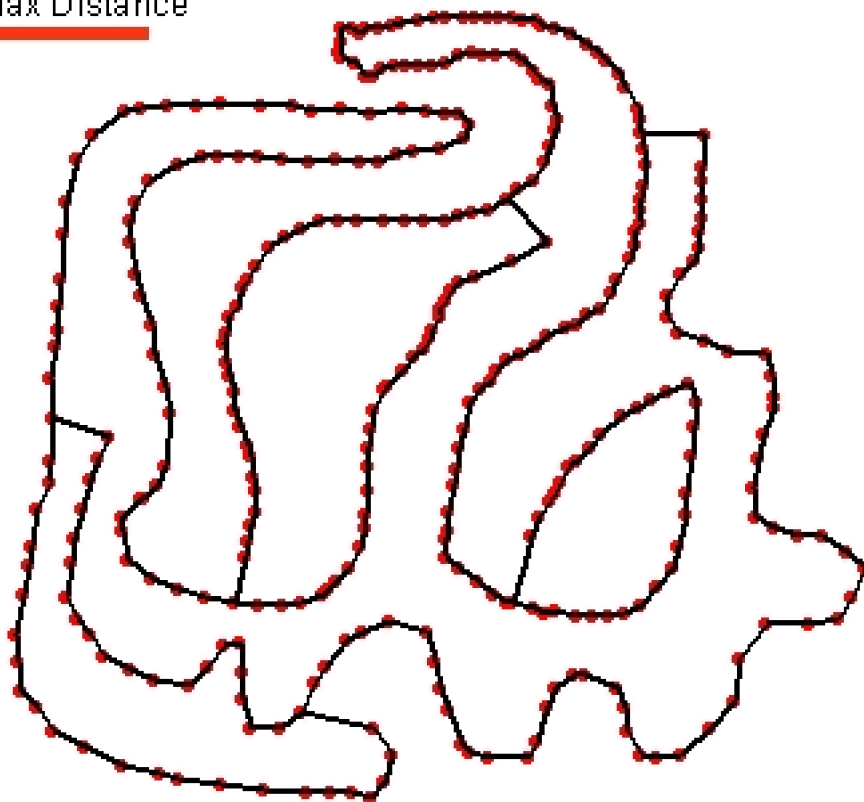


Result - Avoid Loops = True



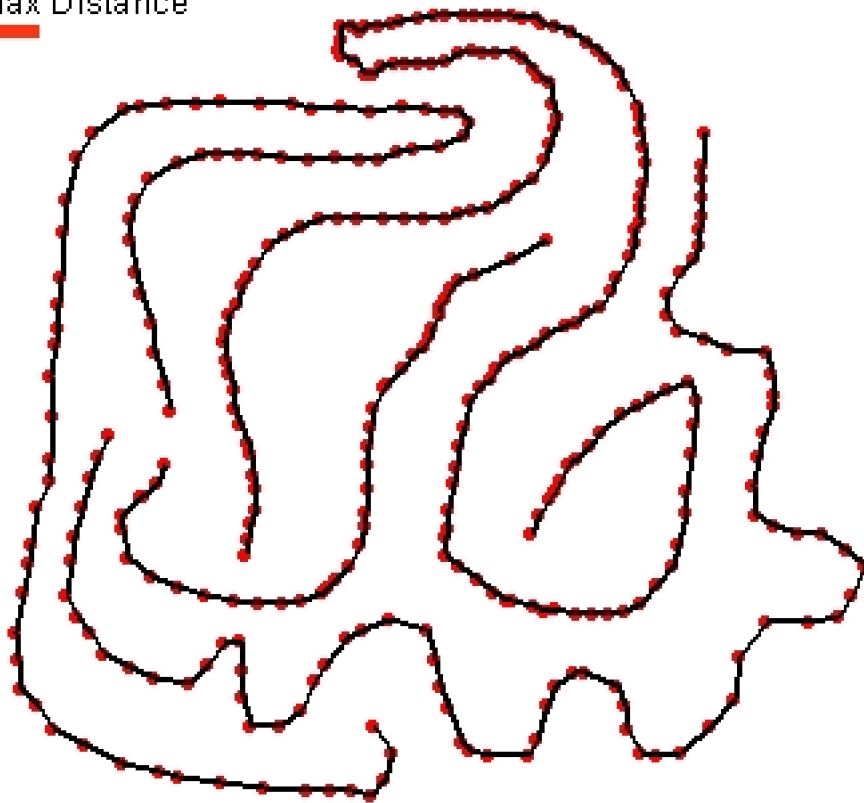
Tolerance 1

Max Distance



Tolerance 2

Max Distance



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ConnectUnstructuredPoints
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<CutOff Distance>	A Double representing the CutOff Distance(in the units of the spatial reference of the input dataset).
{Avoid Loops}	A Boolean. If TRUE the algorithm will try to remove the loops created by connecting to closest neighbor

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ConnectUnstructuredPoints", "input dataset", "output dataset", "CutOff Distance", "Avoid Loops"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ConnectUnstructuredPoints" "input dataset" "output dataset" "CutOff Distance" "Avoid Loops"</code>
.NET using ETGWOuX.dll	<code>ConnectUnstructuredPoints(input dataset, output dataset, CutOff Distance, Avoid Loops)</code>
ArcPy	<code>arcpy.ConnectUnstructuredPoints(input dataset, output dataset, "CutOff Distance", "Avoid Loops")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Cluster Polygons S

[Running programmatically](#)

Delineates cluster polygon for the input points based on user specified cluster distance.

Inputs:

- A Point feature layer.
- Cluster Tolerance - in the units of the spatial reference of the input dataset.
- Holes/No Holes option

Outputs:

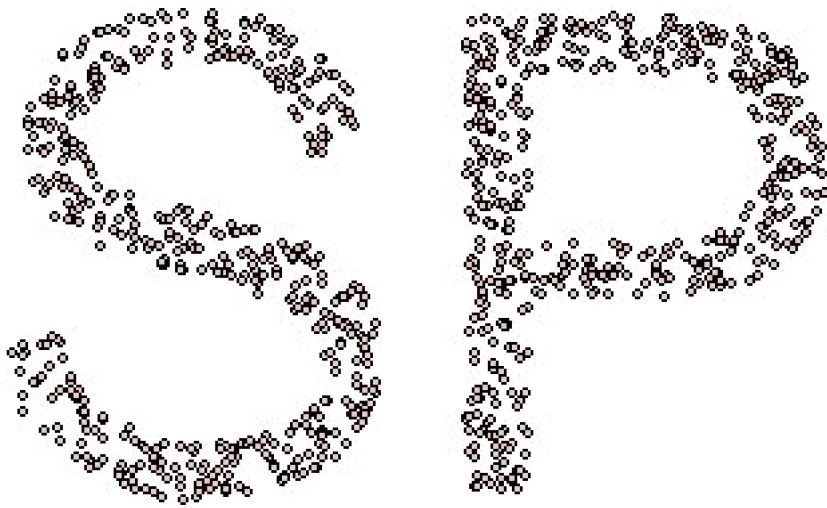
- New polygon layer.

Notes:

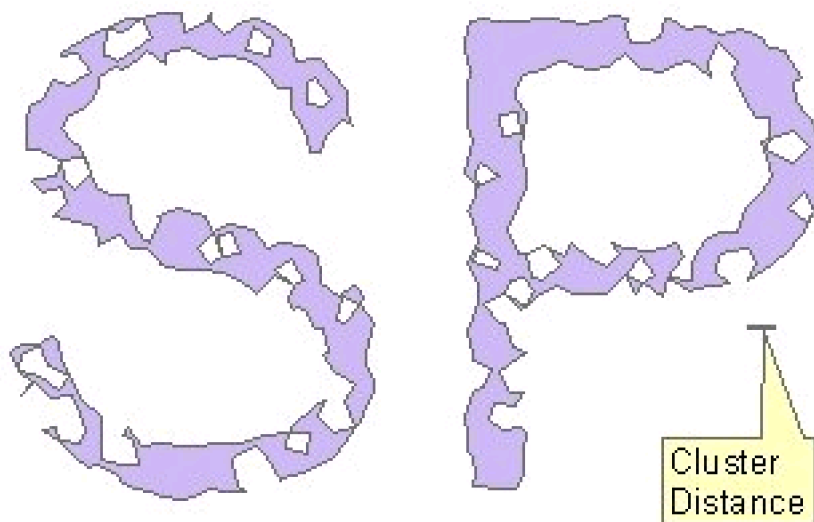
- The algorithm starts from a random point and if finds 2 points closer than the Cluster Tolerance to the start point creates an initial cluster polygon. As long as there is another point closer to the initial cluster polygon, it is joined to it. The function is suitable for creating elongated clusters for points representing linear spatial phenomenon.
- See also the [ClusterPolygonsC](#) function which uses a different algorithm and creates Center based clusters.

Examples:

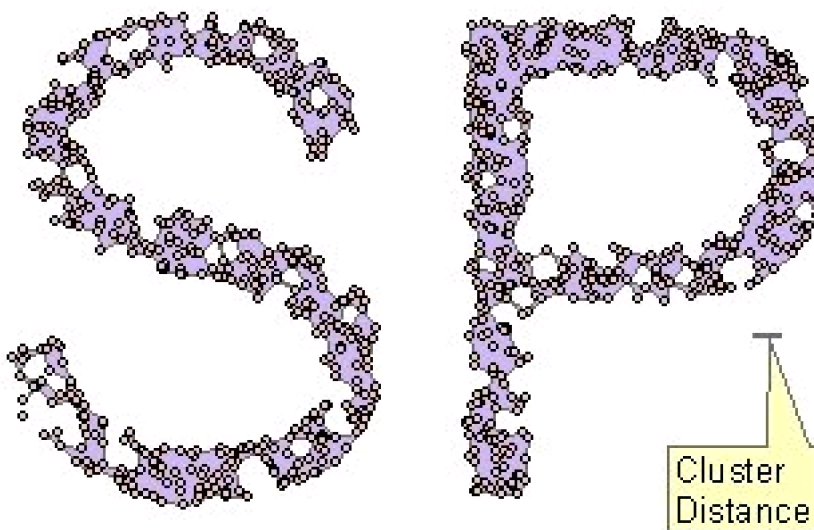
Source Points



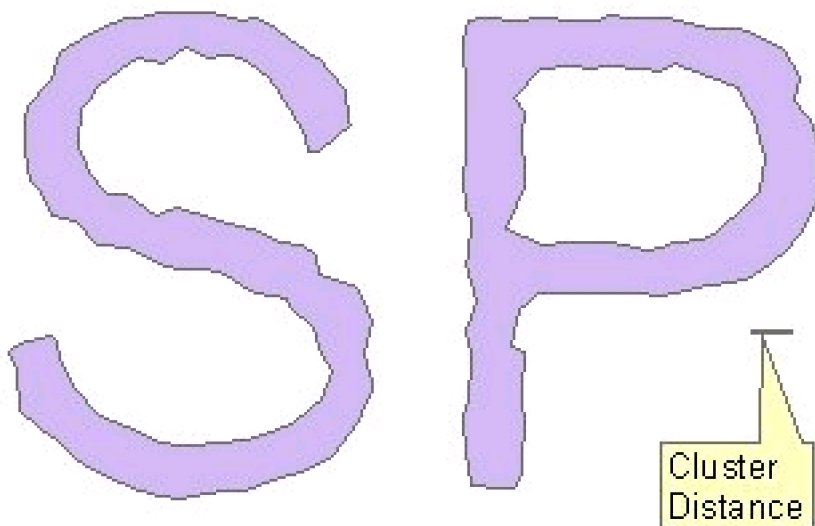
Cluster Polygons 1



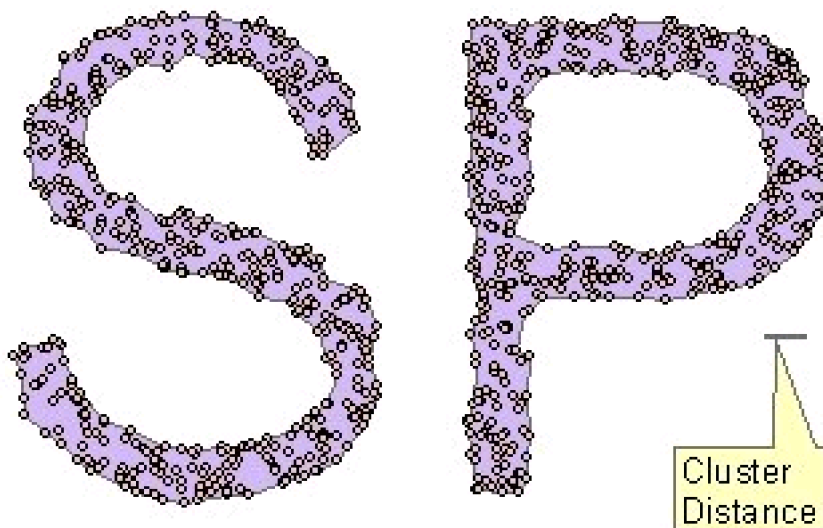
Cluster Polygons 1 overlaid with the source points



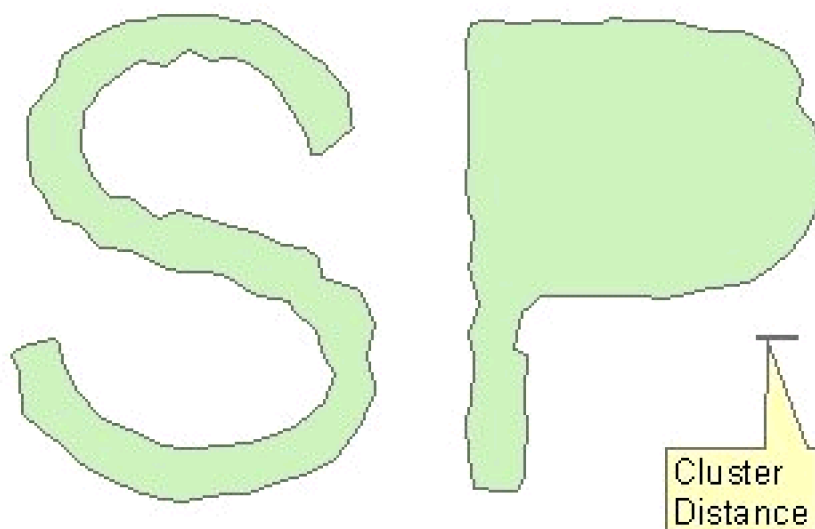
Cluster Polygons 2



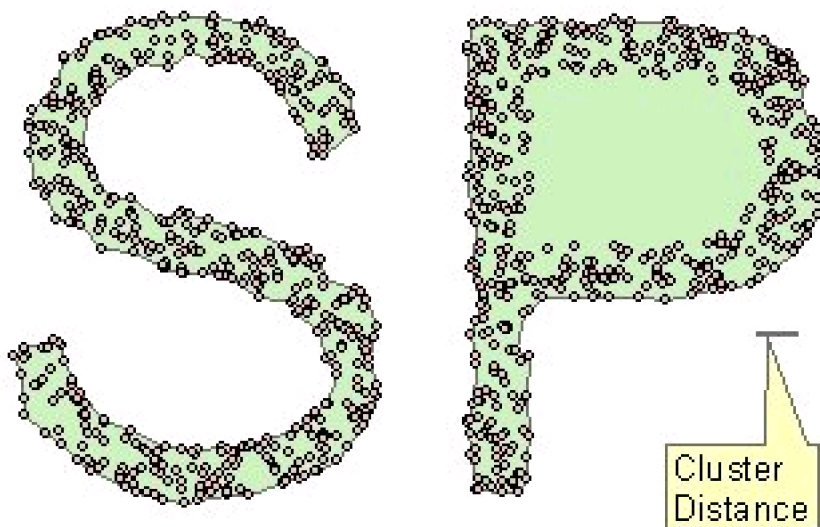
Cluster Polygons 2 overlaid with the source points



Cluster Polygons 3 (No Holes option selected)



Cluster Polygons 3 overlaid with the source point



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToClusterPolygonsS
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<Cluster Tolerance>	A Double representing the Cluster Tolerance(in the units of the spatial reference of the input dataset).
{Remove Holes}	A Boolean indicating whether the function to remove the holes created during the interpolation.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPath, "PointsToClusterPolygonsS", "input dataset", "output dataset", "Cluster Tolearance", "Remove Holes"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToClusterPolygonsS" "input dataset" "output dataset" "Cluster Tolearance" "Remove Holes"</code>
.NET using ETGWOutX.dll	<code>PointsToClusterPolygonsS(input dataset, output dataset, Cluster Tolearance, Remove Holes)</code>
ArcPy	<code>arcpy.PointsToClusterPolygonsS(input dataset, output dataset, "Cluster Tolearance", "Remove Holes")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Cluster Polygons C

[Running programmatically](#)

Delineates cluster polygon for the input points based on user specified cluster distance.

Inputs:

- A Point feature layer.
- Cluster Tolerance - in the units of the spatial reference of the input dataset.
- Cluster Polygon Type

Outputs:

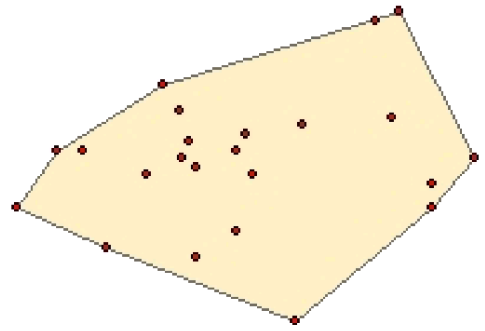
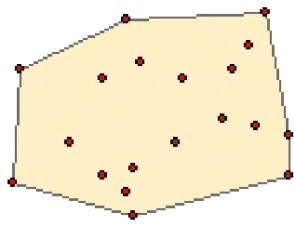
- New polygon layer.

Notes:

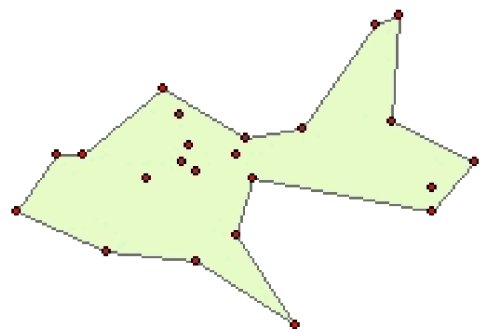
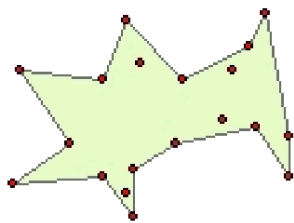
- The algorithm tries first to find the centers for the clusters of points and then assigns to these centers the points located within the cluster tolerance.
- See also the [ClusterPolygonsS](#) function which uses a different algorithm and is able to create elongated clusters.

Examples:

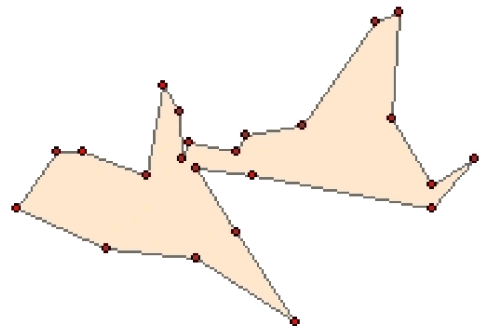
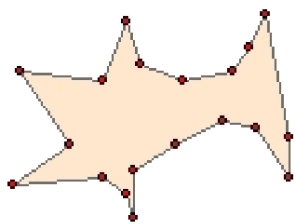
Convex Clusters



Concave Conservative Clusters

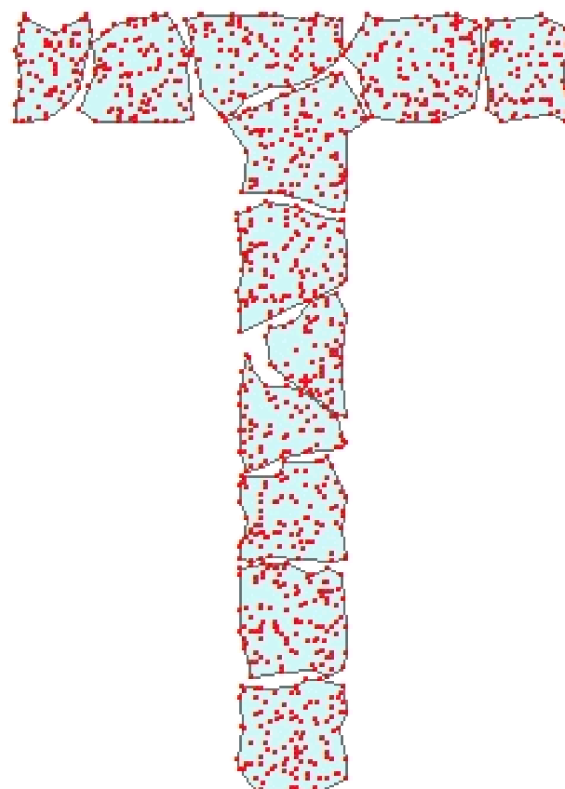
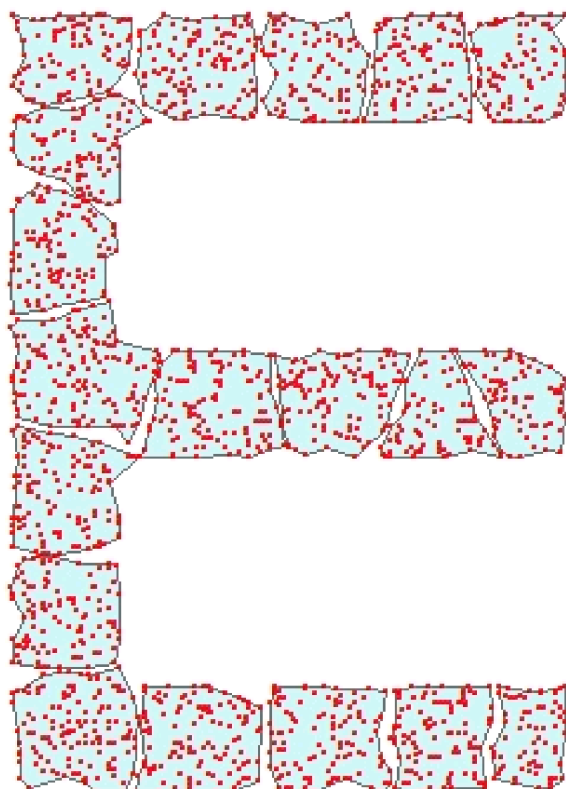


Concave Aggresive Clusters

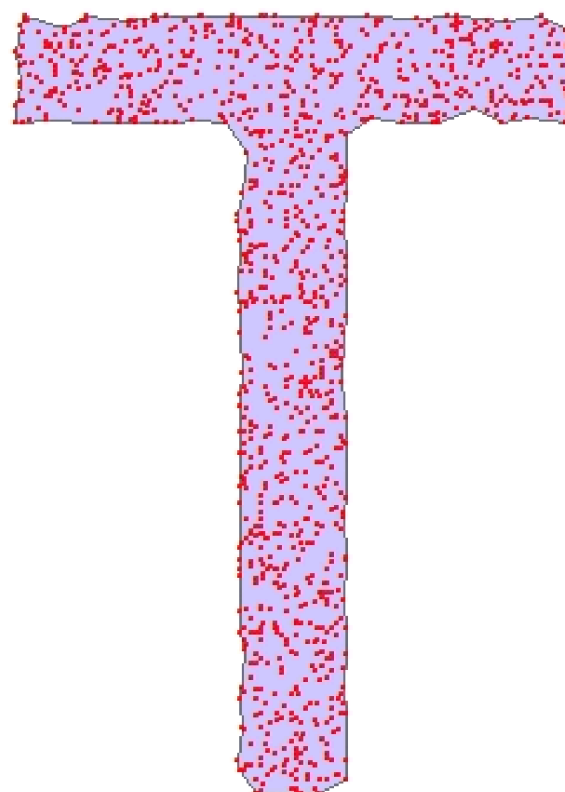
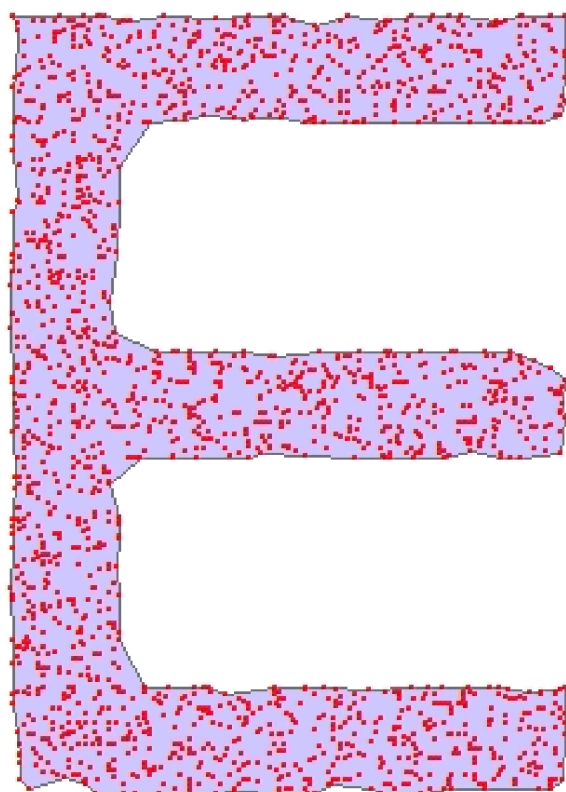


Examples ClusterPolygonsC vs. ClusterPolygonsS:

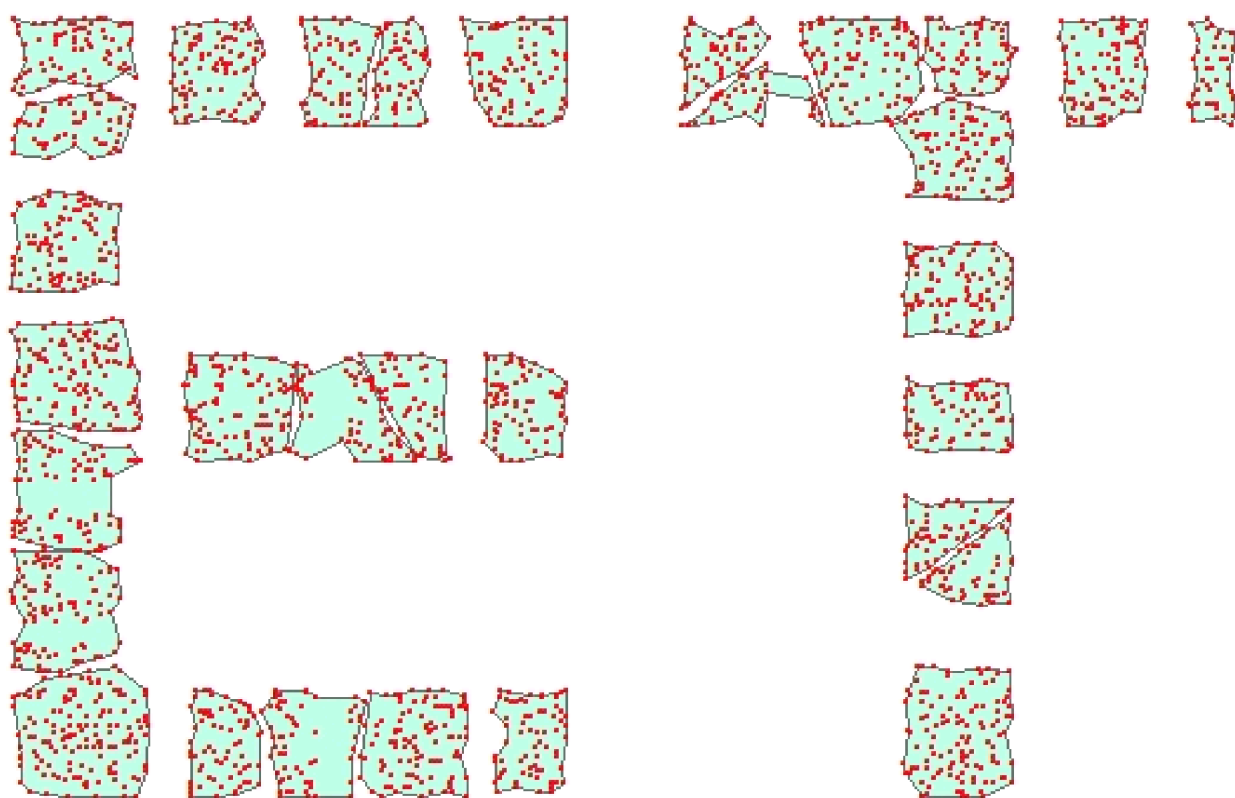
Cluster C 1



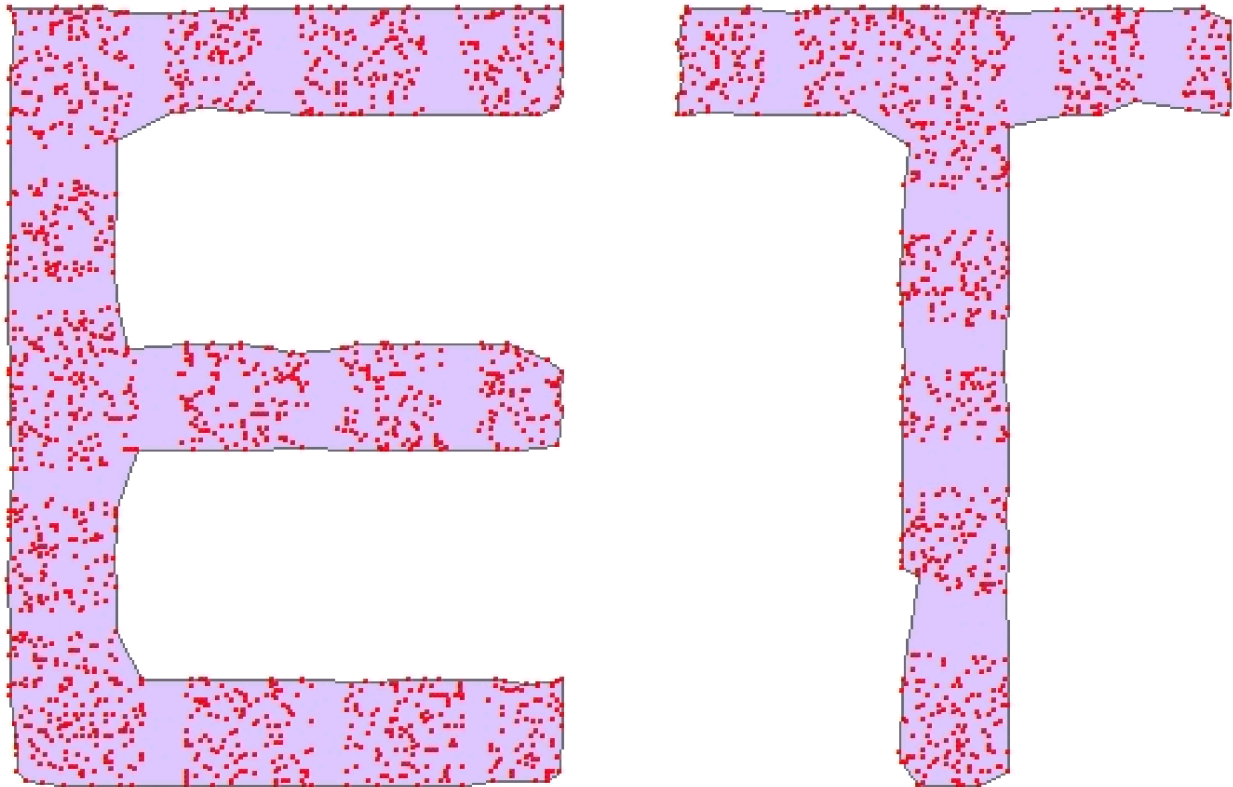
Cluster S 1



Cluster C 2



Cluster S 2



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToClusterPolygonsC
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<Cluster Tolerance>	A Double representing the Cluster Tolerance(in the units of the spatial reference of the input dataset).
<Polygon Type>	A String indicating the type of cluster polygons to be created. Valid options - "Convex", "ConcaveC" - conservative, "ConcaveA" - aggressive.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToClusterPolygonsC", "input dataset", "output dataset", "Cluster Tolearance", "Polygon Type"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToClusterPolygonsC" "input dataset" "output dataset" "Cluster Tolearance" "Polygon Type"</code>
.NET using ETGWOtX.dll	<code>PointsToClusterPolygonsC(input dataset, output dataset, Cluster Tolearance, Polygon Type)</code>
ArcPy	<code>arcpy.PointsToClusterPolygonsC(input dataset, output dataset, "Cluster Tolearance", "Polygon Type")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Google Earth General

Google Earth is a powerful tool for viewing, creating and sharing GIS data. The latest improvements in the KML format allow storing attributes as structured data, which makes possible exchange and even editing of GIS data using Google Earth. Google Earth comes in four different versions (from Free to Enterprise). Make sure to read the Google Earth [license agreement](#) before using it.

What is KML

Keyhole Markup Language (KML) is an XML - based language for managing the display of geo spatial data in Google Maps and Google Earth. Since a KML file is a text file, its size might become quite large. Google Earth also takes a lot of RAM when large KML files are loaded. If possible split your datasets to subsets before converting them to KML.

What is KMZ

The compressed version of the KML with the extension KMZ. Actually this is a zipped archive and the contents can be extracted with any zip program. A KMZ file can contain one or more KML files together with images etc. The export function of ET GeoWizards expect a full file name (with the extension). The extension of the output file defines whether the file will be compressed (KMZ) or not (KML)

Google Earth version

ET GeoWizards exports KML version 2.2 files (this is the KML version which introduced support for attributes called in KML "ExtendedData"). Since it is impossible to find out which exactly version of Google Earth starts supporting KML 2.2, we recommend using Google Earth 4.2 or above.

Google Earth projection

For its reference system, KML uses Geographic Coordinate System (GCS) with WGS84 datum. In ArcGIS this projection is called GCS_WGS_1984. The export to Google Earth functions of ET GeoWizards project the data on the fly to GCS_WGS_1984. If the source data is in a projection that have different datum, the functions of ET GeoWizards do on the fly geographic transformations on the data.

- If the input data does not have a projection associated with it or have so called "Unknown" coordinate system, the data cannot be exported to KML.
- If the export functions cannot find an appropriate geographic transformation to project the input data to GCS_WGS_1984, they will not export the data. This might happen if the input data is in a very specific or outdated projection.

Consideration when exporting to Google Earth

1. The export function of ET GeoWizards supports multipart features, which are exported as MultiGeometry in the KML file. These features will consist of several not connected geometries

and will have a single label point. If the same features are imported back, each geometry will be created as a single part feature with the same attributes.

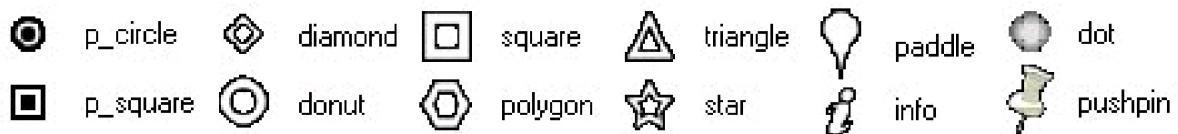
2. The export functions of ET GeoWizards allow creating Labels for each Polyline and Polygon features. The Labels are created as follows:

- For polygons - the label points of the polygons.
- For polylines - the middle point of the polyline

Using Labels is convenient way to display the name of a polyline or polygon feature. Since Labels are part of the feature they are also linked to the feature attributes, which can be displayed by clicking on the Label as well as by clicking on the feature.

Point features are always labelled and the user can not turn the label option off.

3. Point Symbols. ET GeoWizards uses a set of the standard Google Earth marker symbols to display point features and Labels for polyline and polygon features. The user can select the marker to be used for each feature class. The symbols that can be used are:



The size and the color of the symbols are taken from:

- Size - assigned by the user
- Color - randomly assigned

4. Exporting elevations: The export functions of ET GeoWizards allow three ways of exporting Z values for the features.

- Z values from geometry - Only available if the exported dataset to be exported has Z values (PointZ, PolylineZ, PolygonZ).
- Z values from a field - A numeric field is required
- Constant Z values for all features

Note that Google Earth uses elevation values in Meters. If the Z values of the dataset are in Feet, the user needs to indicate this in the export procedure.

Representation of the elevations in Google Earth:

- Z Type - how the Z values will be interpreted by Google Earth
 - Absolute - Sets the altitude of the coordinate relative to sea level, regardless of the elevation of the Google Earth terrain beneath the feature.
 - Relative - Sets the altitude of the feature relative to the Google Earth terrain in a particular location.
 - NONE - the Z values are ignored - the feature will be displayed on the Google Earth surface
- Extrusion - Specifies whether to connect the geometry to the ground.

5. Attributes. All attributes of the features are exported and can be displayed in Google Earth. To display the attributes select the feature or its Label point.

Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Export To Google Earth

[Running programmatically](#)

Converts a feature class to a Google Earth KML or KMZ file. Available only in the ToolBox implementation

Inputs:

- A dataset - Point, Polyline or Polygon
- The appearance of the data in Google Earth can be customized using the parameters described below

Outputs:

- A KML or KMZ file redy to be loaded in Google Earth.
 - [See important information here](#)
 - The attributes of the input data set are added to the output KML.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ExportToGoogle
<input dataset>	A feature layer
<out file>	A String - the full name of the output Google Earth file. The extension (KML or KMZ) is required and will define whether the result will be compressed (KMZ) or not (KML).
{KML	A String that will be used for a general description of the KML file.

Description}	
{label field}	A String representing a field name. The values in this field will be used for naming the Google Earth features.
{transparency}	A Double indicating the transparency to be used. 0 = Opaque, 100 = invisible
{Z Source}	<p>A String indicating what will be the source for the elevation values. Valid strings:</p> <ul style="list-style-type: none"> • "Z" - Z values from geometry - Only if the exported dataset to be exported has Z values (PointZ, PolylineZ, PolygonZ). • "Field " - Z values from a field - A numeric field is required • "Constant" - constant Z values for all features
{z field}	A String representing a field name (numeric field). If the Z Source = "Field ", the values in this field will be used to get the Z values.
{z constant}	A Double representing the Z values for all features if Z Source = "Constant"
{z units}	A String indicating the units of the Z values of the input dataset. Valid strings - "Meters" and "Feet".
{z type}	<p>A String indicating how the Z values will be interpreted. Valid strings:</p> <ul style="list-style-type: none"> • "Absolute" - Sets the altitude of the coordinate relative to sea level, regardless of the elevation of the Google Earth terrain beneath the feature. • "Relative" - Sets the altitude of the feature relative to the Google Earth terrain in a particular location. • "None" - the Z values are ignored - the feature will be displayed on the Google Earth surface.
{extrude geometries}	A Boolean indicating whether to connect the geometry to the ground.

{fill color}	A String indicating the fill color for the polygons and line color for polylines. Valid values: white, red, gree, blue, cyan, magenta, yellow, black, grey
{line color}	A String indicating the outline color for the polygons. Valid values: white, red, gree, blue, cyan, magenta, yellow, black, grey
{line width}	A Double representing the width of the Polyline features and outline width for Polygon features
{export labels}	A Boolean indicating whether labels will be exported as Info-Points.
{info symbol}	A String indicating which of the available Google Earth symbols will be used for displaying Info-Points. The available symbols are above.
{info size}	A Double indicating the size of the Icon for the Info-Points. Actually this is a scale factor for the Google Earth markers - values of 0.5 to 1.5 will give good results.
{coordinate precision}	An Integer representing the number of digits after the decimal point for exported coordinates.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "ExportToGoogle", "input dataset", "out file", "KML Description", "label field", "transparency", "Z Source", "z field", "z constant", "z units", "z type", "extrude geometries", "fill color", "line color", "line width", "export labels", "info symbol", "info size", "coordinate precision"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ExportToGoogle" "input dataset" "out file" "KML Description" "label field" "transparency" "Z Source" "z field" "z constant" "z units" "z type" "extrude geometries" "fill color" "line color" "line width" "export labels" "info symbol" "info size" "coordinate precision"
.NET using ETGWOuX.dll	ExportToGoogle(input dataset, out file, KML Description, label field, transparency, Z Source, z field, z constant, z units, z type, extrude geometries, fill color, line color, line width, export labels, info symbol, info size, coordinate precision)

ArcPy

```
arcpy.ExportToGoogle(input dataset, out file, "KML Description", "label field",  
"transparency", "Z Source", "z field", "z constant", "z units", "z type", "extrude  
geometries", "fill color", "line color", "line width", "export labels", "info symbol",  
"info size", "coordinate precision")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Import from Google Earth

[Running programmatically](#)

Converts the feature data contained in a KML or KMZ file to shapefile or File GDB layers

Inputs:

- A Google Earth KML or KMZ file
- Output workspace - a folder or FileGDB

Notes:

- The KML format allows a lot of freedom in the data structure and not all applications that create KML files structure the data in the same way. This in many cases makes it impossible to import correctly the data. This is frequently the case with attribute data exported from GIS formats. Often the attribute data is exported as part of the description field for a feature. This is usually done in HTML format, which is not structured. KML Version 2.2 supports structured attribute data through the "ExtendedData" element. The import function of ET GeoWizards creates attributes based on this element.
- KML Version 2.2 supports models, which are 3D objects in their own coordinate space. Such models are not imported by ET GeoWizards. This includes Google SketchUp models.
- ET GeoWizards does not support the "Link" element, which references data in external KML or KMZ files.
- The KML structure does have a definition of the Field Types, but not for the fields width, precision and scale. General rules are used to create the fields. You can use the Redefine fields to fix some problems in the field definitions.
- See [Google Earth general](#) for important additional information.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	ImportFromGoogle
<Input KML>	A String representing the full path to the KML(KMZ) file
<Output Workspace>	A String - the full path to the output folder or File GDB.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ImportFromGoogle", "Input KML", "Output Workspace"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportFromGoogle" "Input KML" "Output Workspace"</code>
.NET using ETGWOutX.dll	<code>ImportFromGoogle(Input KML, Output Workspace)</code>
ArcPy	<code>arcpy.ImportFromGoogle(Input KML, Output Workspace)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Generate

[Running programmatically](#)

Inputs:

- A text file - the format is described below
- If the text file contains attributes, the field names are extracted from the first line in the text file. The user has to specify the type of the fields (String, Integer, Long, Double) , the length and the scale (for double type fields)
- The user has to specify what will be the output feature class type

Outputs:

- A new feature class
 - Point, PointZ, PointM
 - Polyline, PolylineZ, PolylineM
 - Polygon, PolygonZ, PolygonM
 - Polygon (from a box type input file)

Notes:

- All the non valid records will be ignored
 - Characters in the coordinate lines or positions
 - Polylines with less than two coordinate lines
 - Polygons with less than three coordinate lines
 - Coordinate entries with less than two coordinates for normal shapes and less than three coordinates for Z or M shapes
- The Polygon options will use only the closed shapes described. If "Force closure" option is used all the shapes that can be closed will be added to the feature class
- If "Attribute" option is used the field names will be extracted from the first non empty line in the text file
- Avoid using reserved field names "Shape", "ObjectID" etc.
- All the field names longer than 10 characters will be converted to 10 character strings

File formats:

The shapes that have Z or M values will have an additional coordinate

Shape Type	Standard Format	Example	Extended Format	Example
Point PointZ PointM	id,x,y id,x,y id,x,y END	1,34.5,-14.3 2,12.8,-19.6 3,13.4,-25.6 END	ID,X,Y,FIELD,FIELD id,x,y,value,value id,x,y,value,value id,x,y,value,value END	ID,X,Y,Town,Population 1,34.5,-14.3,London,44 2,12.8,-19.6,Paris,34 3,13.4,-25.6,Madrid,56 END
Polyline PolylineZ PolylineM	id x,y x,y END id x,y x,y x,y END END	1 34.5,-14.3 12.8,-19.6 END 2 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END	ID,FIELD,FIELD id,value,value x,y x,y END id,value,value x,y x,y x,y END END END	ID,Street,StreetsType 1,Church,Street 34.5,-14.3 12.8,-19.6 END 2,Second,Avenue 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END END
Polygon PolygonZ PolygonM	id,xLabel,yLabel x,y x,y x,y END id x,y x,y x,y END END	1, 12.5,-18,6 34.5,-14.3 12.8,-19.6 12.43,-19.88 END 2,14.3,24.5 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END	ID,FIELD,FIELD id,value,value x,y x,y x,y END id,value,value x,y x,y x,y END END END	ID,X,Y, Dam,Volume 1,12.5,-18,6,Vaal,5346 34.5,-14.3 12.8,-19.6 12.43,-19.88 END 2,14.3,24.5,Gariep,6578 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END END
Box	id,xmin,ymin,xmax,ymax id,xmin,ymin,xmax,ymax id,xmin,ymin,xmax,ymax END	1,34.5,-14.3,34.8,-14.1 2,12.8,-19.6,12.9,-19.2 3,13.4,-25.6,13.6,-25.4 END	ID,XMIN,YMIN,XMAX,YMAX,FIELD id,xmin,ymin,xmax,ymax,value id,xmin,ymin,xmax,ymax,value id,xmin,ymin,xmax,ymax,value END	

Running Programmatically

[\(Go to TOP\)](#)

Two different functions are available for the Vector Grid creation

Vector Grid Extent

Parameters

Expression	Explanation
Function Name	PointGridExtent
<output dataset>	A String - the full name of the output layer.

<Shape Type>	Required. A String indicating the type of the grid to be created. Valid values: <ul style="list-style-type: none"> • "Triangle" • "Square" • "Rectangle"
<Has Attributes>	A Double - the cell size in X direction.
<Has Z>	A Double - the cell size in Y direction.
<Extents From Reference>	A Boolean indicating whether the extents of the grid will be taken from a reference dataset.
{Reference Dataset}	A String - the full name of the reference dataset.
{MinX}	A Double - minimum X of the extent.
{MinY}	A Double - minimum Y of the extent.
{MaxX}	A Double - maximum X of the extent.
{MinX}	A Double - maximum Y of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PointGridExtent", "output dataset", "Shape Type", "Has Attributes", "Has Z", "Extents From Reference", "Reference Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointGridExtent" "output dataset" "Shape Type" "Has Attributes" "Has Z" "Extents From Reference" "Reference Dataset"
.NET using ETGWOuX.dll	PointGridExtent(output dataset, Shape Type, Has Attributes, Has Z, Extents From Reference, Reference Dataset)
ArcPy	arcpy.PointGridExtent(output dataset, "Shape Type", "Has Attributes", "Has Z", "Extents From Reference", "Reference Dataset")

[\(Go to TOP\)](#)

Vector Grid Origin

Parameters

Expression	Explanation
------------	-------------

Function Name	Generate
<input file>	A String - the full name of the input text file.
<output dataset>	A String - the full name of the output layer.
<Shape Type>	Required. A String indicating the shape type of the data in the text file. Valid values: <ul style="list-style-type: none"> • "Point" • "Polyline" • "Polygon" • "Box" • "Line"
<Has Attributes>	A Boolean indicating whether the input data contains attributes.
{Has Z}	A Boolean indicating whether the input data has Z values.
{Has M}	A Boolean indicating whether the input data has M values.
{Reference Dataset}	A String - the full name of the reference dataset.
{Force Closure}	A Boolean indicating whether to close unclosed polygons (only if the Shape Type = "Polygon").

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "Generate", "input file", "output dataset", "Shape Type", "Has Attributes", "Has Z", "Has M", "Reference Dataset", "Force Closure"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "Generate" "input file" "output dataset" "Shape Type" "Has Attributes" "Has Z" "Has M" "Reference Dataset" "Force Closure"
.NET using ETGWOtX.dll	Generate(input file, output dataset, Shape Type, Has Attributes, Has Z, Has M, Reference Dataset, Force Closure)
ArcPy	arcpy.Generate(input file, output dataset, "Shape Type", "Has Attributes", "Has Z", "Has M", "Reference Dataset", Force Closure)

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Ungenerate

[Running programmatically](#)

Exports a feature class to ArcInfo generate format text file. The user can specify optionally to export the attributes. In this case the format of the result text file will be in an extended version of ArcInfo generate format - described below

Inputs:

- A feature layer
 - Point, PointZ, PointM
 - Polyline, PolylineZ, PolylineM
 - Polygon, PolygonZ, PolygonM

Outputs:

- New text file

Notes:

- If the "Export attributes" option is selected the output text file might not be readable from the ArcInfo Generate command
- If the "Export bounding rectEnvelope Onlys only" is selected (available for Polyline and Polygon layers), the resulting text file will contain the coordinates of the extents for each shape.

File formats:

The shapes that have Z or M values will have an additional coordinate

Shape Type	Standard Format	Example	Extended Format	Example
Point PointZ PointM	id,x,y id,x,y id,x,y END	1,34.5,-14.3 2,12.8,-19.6 3,13.4,-25.6 END	ID,X,Y,FIELD,FIELD id,x,y,value,value id,x,y,value,value id,x,y,value,value END	ID,X,Y,Town,Population 1,34.5,-14.3,London,44 2,12.8,-19.6,Paris,34 3,13.4,-25.6,Madrid,56 END
Polyline PolylineZ PolylineM	id x,y x,y END id x,y x,y x,y END END	1 34.5,-14.3 12.8,-19.6 END 2 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END	ID,FIELD,FIELD id,value,value x,y x,y END id,value,value x,y x,y x,y END END	ID,Street,Streettype 1,Church,Street 34.5,-14.3 12.8,-19.6 END 2,Second,Avenue 13.4,-25.6 16.4,-27.6 13.8,-22.1 END END

Polygon	id,xLabel,yLabel	1, 12.5,-18,6	ID,FIELD,FIELD	ID,X,Y, Dam,Volume
PolygonZ	x,y	34.5,-14.3	id,value,value	1,12.5,-18,6,Vaal,5346
PolygonM	x,y	12.8,-19.6	x,y	34.5,-14.3
	x,y	12.43,-19.88	x,y	12.8,-19.6
	END	END	x,y	12.43,-19.88
	id	2,14.3,24.5	END	END
	x,y	13.4,-25.6	id,value,value	2,14.3,24.5,Gariep,6578
	x,y	16.4,-27.6	x,y	13.4,-25.6
	x,y	13.8,-22.1	x,y	16.4,-27.6
	END	END	x,y	13.8,-22.1
	END	END	END	END
	END	END	END	END
Box	id,xmin,ymin,xmax,ymax	1,34.5,-14.3,34.8,-14.1	ID,XMIN,YMIN,XMAX,YMAX,FIELD	
	id,xmin,ymin,xmax,ymax	2,12.8,-19.6,12.9,-19.2	id,xmin,ymin,xmax,ymax,value	
	id,xmin,ymin,xmax,ymax	3,13.4,-25.6,13.6,-25.4	id,xmin,ymin,xmax,ymax,value	
	END	END	id,xmin,ymin,xmax,ymax,value	
			END	

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	Ungenerate
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output file>	A String - the full name of the output text file.
<Delimiter>	A String indicating the delimiter to be used. Valid values: "Comma", "Tab", "Space"
{Delimiter Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the distance between the grid points.
{Export Attributes}	A Boolean indicating whether the attributes to be exported to the text file
{Envelope Only}	A Boolean indicating whether the envelope of the geometry to be exported instead of the entire geometry.
{Precision}	A Integer representing the number of digits after the decimal point to be used for storing coordinates.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "Ungenerate", "input dataset", "output file", "Delimiter", "Export Attributes", "Envelope Only", "Precision"])

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "Ungenerate" "input dataset" "output file" "Delimiter" "Export Attributes" "Envelope Only" "Precision"
.NET using ETGWOutX.dll	Ungenerate(input dataset, output file, Delimiter, Export Attributes, Envelope Only, "Precision")
ArcPy	arcpy.Ungenerate(input dataset, output file , "Delimiter", "Export Attributes", "Envelope Only", "Precision")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import from MapInfo

[Running programmatically](#)

Converts MapInfo native (TAB) or interchange (MIF) file to shapefile or File GDB layer/s.

Inputs

- A TAB or MIF file.

Outputs

- A shapefile/s or File GDB layer/s. One MapInfo file might contain different type geometries. If this is the case a multiple layers will be produced - one for each geometry type. The names of the output layers will be:
 - Point - OtputNamePnt
 - Plyline - OtputNamePI
 - Polygon - OtputNamePg

Notes

- A MIF refers to a set of two MapInfo interchange files with extensions .MID and .MIF.
- A TAB refers to a set of MapInfo binary files (usually with extensions .TAB, .DAT, .MAP, .ID, .IND)
- There might be some problems with the spatial reference of the converted layers.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function	ImportFromMapInfo

Name	
<Input MapInfo>	A String - the full name of the TAB or MIF file to be converted.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "ImportFromMapInfo", "Input MapInfo", "output dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportFromMapInfo" "Input MapInfo" "output dataset"
.NET using ETGWOtX.dll	ImportFromMapInfo(Input MapInfo, output dataset)
ArcPy	arcpy.ImportFromMapInfo(Input MapInfo, output dataset)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import DXF

[Running programmatically](#)

Converts an AutoDesk interchange DXF file to shapefile or File GDB layer/s.

Inputs

- A DXF file.

Outputs

- A shapefile/s or File GDB layer/s. One DXF file might contain different type geometries. If this is the case a multiple layers will be produced - one for each geometry type. The names of the output layers will be:
 - Point - OtputNamePnt
 - Plyline - OtputNamePI
 - Polygon - OtputNamePg

Notes

- The resulting layer/s will not associated spatial reference.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportFromDXF
<Input DXF>	A String - the full name of the DXF file to be converted.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ImportFromDXF", "Input DXF", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "ImportFromDXF" "Input DXF" "output dataset"</code>
.NET using ETGWOuX.dll	<code>ImportFromDXF(Input DXF, output dataset)</code>
ArcPy	<code>arcpy.ImportFromDXF(Input DXF, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import from GeoJSON

[Running programmatically](#)

Converts spatial data encoded in GeoJSON format plain text files (.JSON or .GEOJSON file extensions) to shapefile or File GDB layer/s.

Inputs

- A JSON or GEOJSON file.
- The geometry type (Point, Polyline, Polygon or All) to be imported.

Outputs

- A shapefile/s or File GDB layer/s. One GeoJSON file might contain different type geometries. The user can specify which geometry type (or All) to be extracted. If the type specified is "All" and the GeoJSON file contains multiple geometry types, then multiple layers will be produced - one for each geometry type present in the input. The names of the output layers will be:
 - Point - OutputNamePnt
 - Polyline - OutputNamePI
 - Polygon - OutputNamePg

Notes:

- The output will preserve the spatial reference of the input GeoJSON file.
- If the input file does not have spatial reference defined, then
 - If the extents of the input data are between -180 and 180, a geographic coordinate system (WGS 84) will be set for spatial reference of the output layer/s.
 - If the extents of the input data are out of the -180 and 180 range and Unknown spatial reference will be set for the output layer/s.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportGeoJson
<Input GeoJSON>	A String - the full name of the .json or .geojson file to be converted.
<output dataset>	A String - the full name of the output layer.
<Option>	<div>A String - the conversion option. Valid values.<ul style="list-style-type: none">• "Point"• "Polygon"• "Polyline"• "All"</div>

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ImportGeoJson", "Input GeoJSON", "output dataset", "Option"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportGeoJson" "Input GeoJSON" "output dataset" "Option"</code>
.NET using	<code>ImportGeoJson(Input GeoJSON, output dataset, Option)</code>

ETGWOtX.dll	
ArcPy	<code>arcpy.ImportGeoJson(Input GeoJSON, output dataset, Option)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Export To GeoJson

[Running programmatically](#)

Converts a feature layer to a text file encoded in GeoJSON format (.JSON or .GEOJSON file extensions).

Inputs

- A feature layer (Point, Polyline or Polygon).

Outputs

- New text file encoded in GeoJSON format (.JSON or .GEOJSON file extensions).

Notes:

- The input will be projected (if needed) and a geographic coordinate system (WGS 84) will be set for spatial reference of the GeoJSON dataset.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ExportToGeoJson
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output GeoJSON>	A String - the full name of the output .json or .geojson file

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPath, "ExportToGeoJson", "input dataset", "output GeoJSON"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "ExportToGeoJson" "input dataset" "output GeoJSON"</code>
.NET using ETGWOuX.dll	<code>ExportToGeoJson(input dataset, output GeoJSON)</code>
ArcPy	<code>arcpy.ExportToGeoJson(input dataset, output GeoJSON)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import from Open Street Map

[Running programmatically](#)

Converts spatial data encoded in Open Street Map - .osm (XML based) and .pbk (optimized binary) formats to shapefile or File GDB layer/s.

Inputs

- A .OSM or .PBF file.
- The geometry type (Point, Polyline, Polygon or All) to be imported.

Outputs

- A shapefile/s or File GDB layer/s. One Open Street Map file might contain different type geometries. The user can specify which geometry type (or All) to be extracted. If the type specified is "All" and the Open Street Map file contains multiple geometry types, then multiple layers will be produced - one for each geometry type present in the input. The names of the output layers will be:
 - Point - OutputNamePnt
 - Polyline - OutputNamePI
 - Polygon - OutputNamePg

Notes:

- In some specific cases the processing of large XML based (.osm) files might fail. If you can obtain the data in an optimized binary (.pbk) file it is much better to use this format.
- The output will preserve the spatial reference of the input Open Street Map file.
- If the input file does not have spatial reference defined, then
 - If the extents of the input data are between -180 and 180, a geographic coordinate system (WGS 84) will be set for spatial reference of the output layer/s.
 - If the extents of the input data are out of the -180 and 180 range and Unknown

spatial reference will be set for the output layer/s.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportFromOSM
<Input OSM>	A String - the full name of the .osm or .pbk file to be converted.
<output dataset>	A String - the full name of the output layer.
<Option>	<div>A String - the conversion option. Valid values.<ul style="list-style-type: none">• "Point"• "Polygon"• "Polyline"• "All"</div>

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<pre>subprocess.call([ETGWPath, "ImportFromOSM", "Input OSM", "output dataset", "Option"])</pre>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportFromOSM" "Input OSM" "output dataset" "Option"</pre>

.NET using ETGWOtX.dll	ImportFromOSM(Input OSM, output dataset, Option)
ArcPy	arcpy.ImportFromOSM(Input OSM, output dataset, Option)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import a feature layer from ArcGIS REST service

[Running programmatically](#)

Converts a single layer from an ArcGIS REST server to shapefile or File GDB layer. An attribute query or spatial extent can be used to convert only part of the published layer.

Inputs

- An ArcGIS Server end point - URL to a specific published Feature Service. The End Point should contain the full path to the feature service that you want to access (http://host/instances/services/folder/subfolder/service). For example:
"http://sampleserver3.arcgisonline.com/ArcGIS/rest/services/Hydrography/Watershed173811/FeatureServer".
- User name and Password if required.
- The name of the layer to be imported.
- Where clause if you want to filter to be imported by attributes (for example: **Country = 'USA' AND Population > 50000**).
- Extraction Extent if you want to import only the features in a specific geographic extent (rectangle).

Outputs

- A shapefile or File GDB layer.

Notes:

- The interface of ET GeoWizards allows to create a list of frequently used services and manage them.
- The output will preserve the spatial reference of the input layer.
- The function is not available in the Toolboxes for ArcGIS Desktop and ArcGIS Pro.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportRESTLayer
<Input EndPoint>	A String - see above.

<Input Layer Name>	A String - the name of the layer to be converted.
<output dataset>	A String - the full name of the output layer.
{Where Clause}	A String representing the Where Clause to be used as filter.
{Extraction Envelope}	A String representing the envelope to be used as spatial filter. The format of the string is "XMin;YMin;XMax;YMax"
{User Name}	A String representing the User Name.
{Pasword}	A String representing the Password.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "ImportRESTLayer", "Input EndPoint", "Input Layer Name", "output dataset", "Where Clause", "Extraction Envelope","User Name","Pasword"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportRESTLayer" "Input EndPoint" "Input Layer Name" "output dataset" "Where Clause" "Extraction Envelope" "User Name" "Pasword"
.NET using ETGWOutX.dll	ImportRESTLayer(Input EndPoint,Input Layer Name, output dataset, Where Clause, Extraction Envelope, User Name, Pasword)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import multiple layers from ArcGIS REST service

[Running programmatically](#)

Converts multiple layers from an ArcGIS REST server to shapefiles or File GDB layers. An user defined spatial extent can be used to convert only parts of the published layers.

Inputs

- An ArcGIS Server end point - URL to a specific published Feature Service. The End Point should contain the full path to the feature service that you want to access (http://host/instances/services/folder/subfolder/service). For example:
"http://sampleserver3.arcgisonline.com/ArcGIS/rest/services/Hydrography/Watershed173811/FeatureServer".
- User name and Password if required.
- The names of the layers to be converted.
- Extraction Extent if you want to import only the features in a specific geographic extent (rectangle).

Outputs

- Shapefile/s or File GDB layer/s.

Notes:

- The interface of ET GeoWizards allows to create a list of frequently used services and manage them.
- The outputs will preserve the spatial reference of the input layers.
- The function is not available in the Toolboxes for ArcGIS Desktop and ArcGIS Pro.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportMultipleRESTLayers
<Input EndPoint>	A String - see above.
<Input Layers Names>	A String representing a list of the names of the layers to be converted. The format of the list is "LayerName1;LayerName2;LayerName3". Note that is an empty string is passed, all

	layers available will be imported.
<output workspace>	A String - the full name to the folder or FileGDB where the outputs will be stored.
{Extraction Envelope}	A String representing the envelope to be used as spatial filter. The format of the string is "XMin;YMin;XMax;YMax"
{User Name}	A String representing the User Name.
{Pasword}	A String representing the Password.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "ImportMultipleRESTLayers", "Input EndPoint", "Input Layers Names", "output workspace", "Extraction Envelope", "User Name", "Pasword"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportMultipleRESTLayers" "Input EndPoint" "Input Layers Names" "output workspace" "Extraction Envelope" "User Name" "Pasword"
.NET using ETGWOuX.dll	ImportMultipleRESTLayers(Input EndPoint,Input Layers Names, output workspace, Extraction Envelope, User Name, Pasword)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Import multiple layers from ArcGIS REST service

[Running programmatically](#)

Converts multiple layers from an Open Geospatial Consortium (OGS) Web Feature Service (WFS) to shapefiles or File GDB layers.

Inputs

- An WFS end point - URL to a specific published Feature Service. The End Point should contain the full path to the feature service that you want to access (http://PathTo/WebFeatureService/service). For example:
"http://nsidc.org/cgi-bin/atlas_south".
- User name and Password if required.
- The names of the layers to be converted.

Outputs

- Shapefile/s or File GDB layer/s.

Notes:

- The interface of ET GeoWizards allows to create a list of frequently used services and manage them.
- The outputs will preserve the spatial reference of the input layers.
- The function is not available in the Toolboxes for ArcGIS Desktop and ArcGIS Pro.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ImportWFSLayers

<Input EndPoint>	A String - see above.
<Input Layer Names>	A String representing a list of the names of the layers to be converted. The format of the list is "LayerName1;LayerName2;LayerName3". Note that if an empty string is passed, all layers available will be imported.
<output workspace>	A String - the full name to the folder or FileGDB where the outputs will be stored.
{User Name}	A String representing the User Name.
{Pasword}	A String representing the Password.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ImportWFSLayers", "Input EndPoint", "Input Layers Names", "output workspace", "User Name", "Pasword"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ImportWFSLayers" "Input EndPoint" "Input Layers Names" "output workspace" "User Name" "Pasword"</pre>
.NET using ETGWOuX.dll	<code>ImportWFSLayers(Input EndPoint,Input Layers Names, output workspace, User Name, Pasword)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Copyright © Ianko Tchoukanski

Vector Grid

[Running programmatically](#)

Creates a polyline or polygon vector grid using:

- User defined extents and cell size
- User defined origin, cell size, number rows and columns.

Inputs:

- Grid creation options:
 - Initial extent of the grid and cell size
 - Origin point of the grid, cell size, number of rows and columns.
- Output spatial reference - can be assigned based on a reference layer used.
- Grid type
 - Polyline
 - Polygon

Outputs:

- New Polyline or Polygon feature class

Notes:

- The initial extent of the grid is defined by the extents of the selected reference layer or manually
- The cell size will be in the units of the reference layer or "Unknown" if no reference layer is used.

- In order to avoid incorrect inputs, the size of the grid is limited to 8,000,000 cells
- A ET_Index field will be added to the attribute table. The values will indicate:
 - Polygon grid - the index of each Grid cell. The cell in the bottom left corner of the Grid will have an index of "0-0"
 - Polyline grid - the X (Y) for each polyline in the input units.

Running Programmatically

[\(Go to TOP\)](#)

Two different functions are available for the Vector Grid creation

Vector Grid Extent

Parameters

Expression	Explanation
Function Name	VectorGridExtent
<output dataset>	A String - the full name of the output layer.
<Grid Type>	Required. A String indicating the type of the grid to be created. Valid values: <ul style="list-style-type: none"> • "Polygon" • "Polyline"
<Cell SizeX>	A Double - the cell size in X direction.
<Cell SizeY>	A Double - the cell size in Y direction.
<Extents From Reference>	A Boolean indicating whether the extents of the grid will be taken from a reference dataset.

{Reference Dataset}	A String - the full name of the reference dataset.
{MinX}	A Double - minimum X of the extent.
{MinY}	A Double - minimum Y of the extent.
{MaxX}	A Double - maximum X of the extent.
{MinX}	A Double - maximum Y of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "VectorGridExtent", "output dataset", "Grid Type", "Cell SizeX", "Cell SizeY", "Extents From Reference", "Reference Dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "VectorGridExtent" "output dataset" "Grid Type" "Cell SizeX" "Cell SizeY" "Extents From Reference" "Reference Dataset"</code>
.NET using ETGWOutX.dll	<code>VectorGridExtent(output dataset, Grid Type, Cell SizeX, Cell SizeY, Extents From Reference, Reference Dataset)</code>
ArcPy	<code>arcpy.VectorGridExtent(output dataset, "Grid Type", "Cell SizeX", "Cell SizeY", "Extents From Reference", "Reference Dataset")</code>

[\(Go to TOP\)](#)

Vector Grid Origin

Parameters

Expression	Explanation
Function Name	VectorGridOrigin

<output dataset>	A String - the full name of the output layer.
<Grid Type>	Required. A String indicating the type of the grid to be created. Valid values: <ul style="list-style-type: none"> • "Polygon" • "Polyline"
<Cell SizeX>	A Double - the cell size in X direction.
<Cell SizeY>	A Double - the cell size in Y direction.
<Number Columns>	An Integer - the number of columns to be created.
<Number Rows>	An Integer - the number of rows to be created.
<Origin From Reference>	A Boolean indicating whether the origin point of the grid will be taken from a reference dataset.
{Reference Dataset}	A String - the full name of the reference dataset.
{Lower LeftX}	A Double - X of the lower left corner of the extent.
{Lower LeftY}	A Double - Y of the lower left corner of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "VectorGridOrigin", "output dataset", "Grid Type", "Cell SizeX", "Cell SizeY", "Number Columns", "Number Rows", "Origin From Reference", "Reference Dataset"])
.NET using	StartInfo.FileName = ETGWPath

ETGWRun.exe	StartInfo.Arguments = "VectorGridOrigin" "output dataset" "Grid Type" "Cell SizeX" "Cell SizeY" "Number Columns" "Number Rows" "Origin From Reference" "Reference Dataset"
.NET using ETGWOutX.dll	VectorGridOrigin(output dataset, Grid Type, Cell SizeX, Cell SizeY, Number Columns, Number Rows, Origin From Reference, Reference Dataset)
ArcPy	arcpy.VectorGridOrigin(output dataset, "Grid Type" , "Cell SizeX", "Cell SizeY", "Number Columns", "Number Rows", "Origin From Reference", "Reference Dataset")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Point Grid

[Running programmatically](#)

Creates a grid of points using user defined grid type and distance between the points.

- User defined extents and cell size
- User defined origin, cell size, number rows and columns.

Inputs:

- Grid creation options:
 - Initial extent of the grid and cell size
 - Origin point of the grid, cell size, number of rows and columns.
- Output spatial reference - can be assigned based on a reference layer used.
- Grid type
 - Square
 - Rectangle
 - Triangle - equilateral
- Distance between the points - in the case of rectangle X and Y spacing are required

Outputs:

- New Point dataset

Notes:

- The initial extent of the grid is defined by the extents of the selected reference layer or manually

- The cell size will be in the units of the reference layer or "Unknown" if no reference layer is used.
- In order to avoid incorrect inputs, the size of the grid is limited to 8,000,000 cells
- A ET_Index field will be added to the point attribute table to indicate the index of each point in the Grid. The point in the bottom left corner of the Grid will have an index of "0-0"

Running Programmatically

[\(Go to TOP\)](#)

Two different functions are available for the Vector Grid creation

Vector Grid Extent

Parameters

Expression	Explanation
Function Name	PointGridExtent
<output dataset>	A String - the full name of the output layer.
<Grid Type>	Required. A String indicating the type of the grid to be created. Valid values: <ul style="list-style-type: none"> • "Triangle" • "Square" • "Rectangle"
<Cell SizeX>	A Double - the cell size in X direction.
<Cell SizeY>	A Double - the cell size in Y direction.
<Extents From Reference>	A Boolean indicating whether the extents of the grid will be taken from a reference dataset.

{Reference Dataset}	A String - the full name of the reference dataset.
{MinX}	A Double - minimum X of the extent.
{MinY}	A Double - minimum Y of the extent.
{MaxX}	A Double - maximum X of the extent.
{MinX}	A Double - maximum Y of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PointGridExtent", "output dataset", "Grid Type", "Cell SizeX", "Cell SizeY", "Extents From Reference", "Reference Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointGridExtent" "output dataset" "Grid Type" "Cell SizeX" "Cell SizeY" "Extents From Reference" "Reference Dataset"
.NET using ETGWOutX.dll	PointGridExtent(output dataset, Grid Type, Cell SizeX, Cell SizeY, Extents From Reference, Reference Dataset)
ArcPy	arcpy.PointGridExtent(output dataset, "Grid Type", "Cell SizeX", "Cell SizeY", "Extents From Reference", "Reference Dataset")

[\(Go to TOP\)](#)

Vector Grid Origin

Parameters

Expression	Explanation
Function Name	PointGridOrigin
<output	A String - the full name of the output layer.

dataset>	
<Grid Type>	<p>Required. A String indicating the type of the grid to be created. Valid values:</p> <ul style="list-style-type: none"> • "Triangle" • "Square" • "Rectangle"
<Cell SizeX>	A Double - the cell size in X direction.
<Cell SizeY>	A Double - the cell size in Y direction.
<Number Columns>	An Integer - the number of columns to be created.
<Number Rows>	An Integer - the number of rows to be created.
<Origin From Reference>	A Boolean indicating whether the origin point of the grid will be taken from a reference dataset.
{Reference Dataset}	A String - the full name of the reference dataset.
{Lower LeftX}	A Double - X of the lower left corner of the extent.
{Lower LeftY}	A Double - Y of the lower left corner of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<pre>subprocess.call([ETGWPath, "PointGridOrigin", "output dataset", "Grid Type", "Cell SizeX", "Cell SizeY", "Number Columns", "Number Rows", "Origin From Reference", "Reference Dataset"])</pre>

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointGridOrigin" "output dataset" "Grid Type" "Cell SizeX" "Cell SizeY" "Number Columns" "Number Rows" "Origin From Reference" "Reference Dataset"
.NET using ETGWOutX.dll	PointGridOrigin(output dataset, Grid Type, Cell SizeX, Cell SizeY, Number Columns, Number Rows, Origin From Reference, Reference Dataset)
ArcPy	arcpy.PointGridOrigin(output dataset, "Grid Type", "Cell SizeX", "Cell SizeY", "Number Columns", "Number Rows", "Origin From Reference", "Reference Dataset")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Random Points On Polylines

[Running programmatically](#)

Generates random points located on the polylines of the input polyline dataset. The number of points per polyline can be constant or different for each polyline - based on the values in a numeric field of the input polyline feature class."

Inputs:

- A polyline feature class
- The number of points per polyline can be input in one of the following ways
 - A numeric field which values will be used to get the number of points to be generated per polyline.
 - A constant number
- Optional: Minimum Distance from boundary - no point will be generated that is closer than this tolerance to the ends of the polyline.

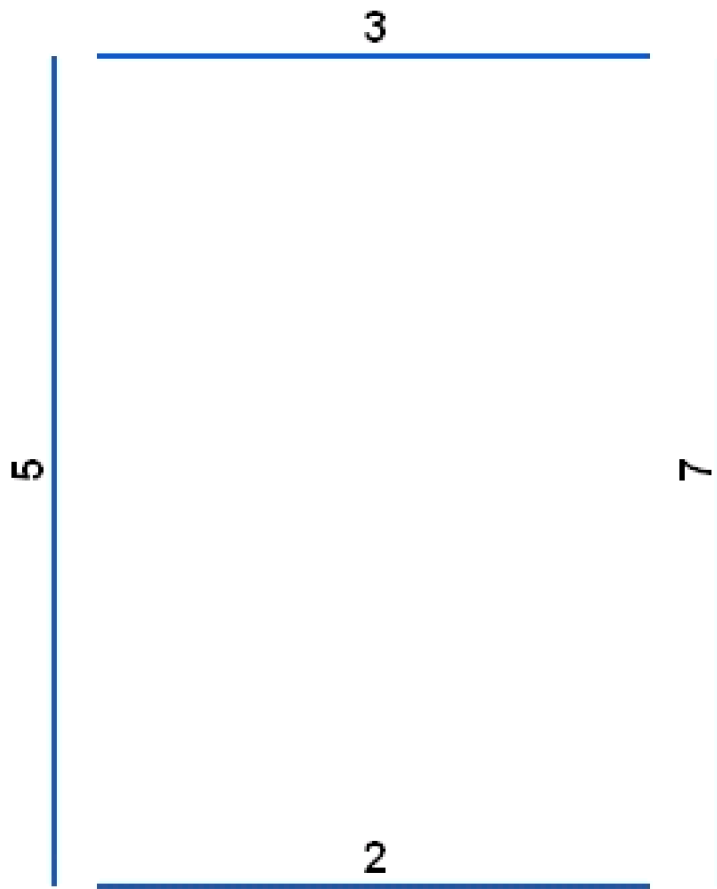
Outputs:

- New Point feature class.
- The following fields are added to the attribute table of the resulting feature class.
 - [ET_ID] - the ID of the original polyline
 - [ET_X] - the X coordinate of the point
 - [ET_Y] - the Y coordinate of the point
 - [ET_Station] - the absolute position of the point along the polyline (in the units of the spatial reference of the input point feature class)

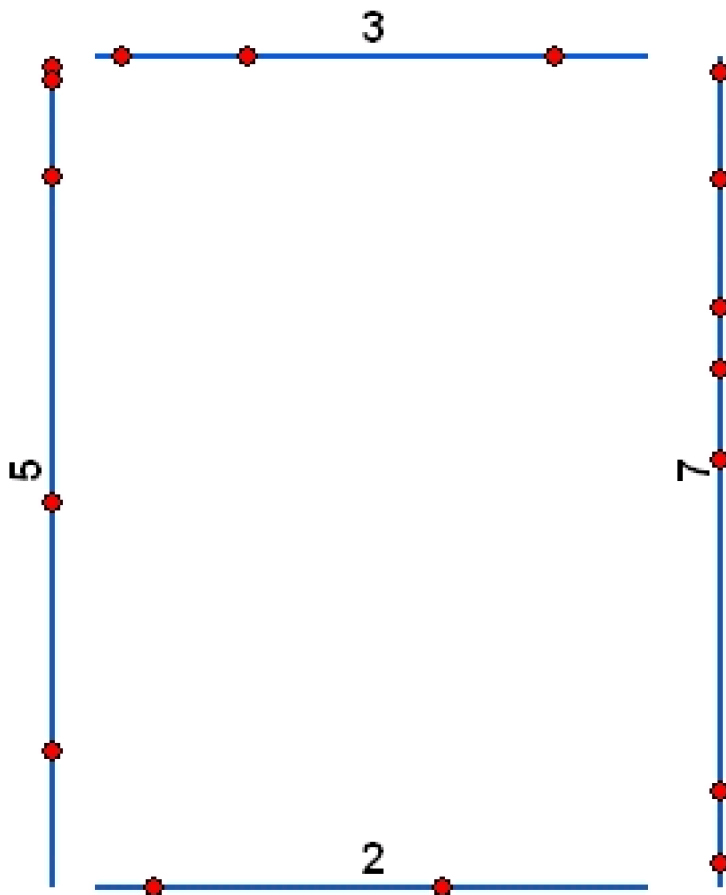
Note: Be careful with assigning the number of points per polyline. The function will try to create N unique points on the polyline and if the number of points allocated is too large, might be very slow or even fall into an indefinite loop.

Illustration:

Original polylines labeled with the values in field to be used as a source for getting the number of points to be generated



The resulting point dataset



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RandomPointsOnPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Number Points}	An Integer representing the number of points per polygon to be created.
{Number	A String representing the name of a field in the in the attribute table of the input

Points Field}	dataset. The field has the values for the number points per polygon to be created.
{Distance to Ends}	A Double representing the minimum distance to the ends of the polyline

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "RandomPointsOnPolylines", "input dataset", "output dataset", "Number Points", "Number Points Field", "Distance to Ends"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "RandomPointsOnPolylines" "input dataset" "output dataset" "Number Points" "Number Points Field" "Distance to Ends"</code>
.NET using ETGWOuX.dll	<code>RandomPointsOnPolylines(input dataset, output dataset, Number Points, Number Points Field, Distance to Ends)</code>
ArcPy	<code>arcpy.RandomPointsOnPolylines(input dataset, output dataset, "Number Points", "Number Points Field", "Distance to Ends")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Random Points In Polygons

[Running programmatically](#)

Generates random points located in the polygons of the input polygon dataset. The number of points per polygon can be constant or different for each polygon - based on the values in a numeric field of the input polygon feature class.

Inputs:

- A polygon feature class
- The number of points per polygon can be input in one of the following ways
 - A numeric field which values will be used to get the number of points to be generated per polygon.
 - A constant number
- Optional: Minimum Distance from boundary - no point will be generated that is closer than this tolerance to the boundary of the polygon.
- Optional: Minimum distance between the points

Outputs:

- New Point feature class.
- The following fields are added to the attribute table of the resulting feature class.
 - [ET_ID] - the ID of the original polygon
 - [ET_X] - the X coordinate of the point
 - [ET_Y] - the Y coordinate of the point

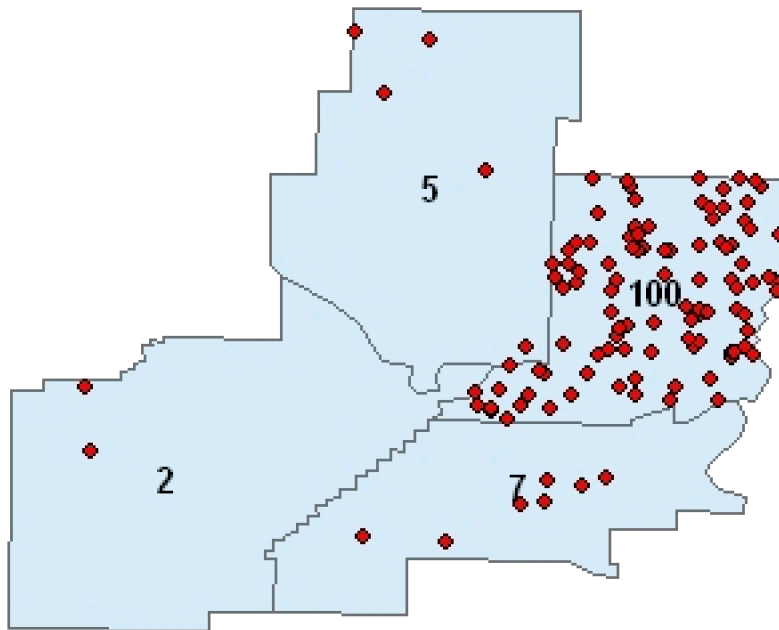
Notes:

- If the number of points specified cannot be placed in specific polygon, a message will be stored in the log file.

- The larger minimum distance between the points specified, the more uniform the points created will be.

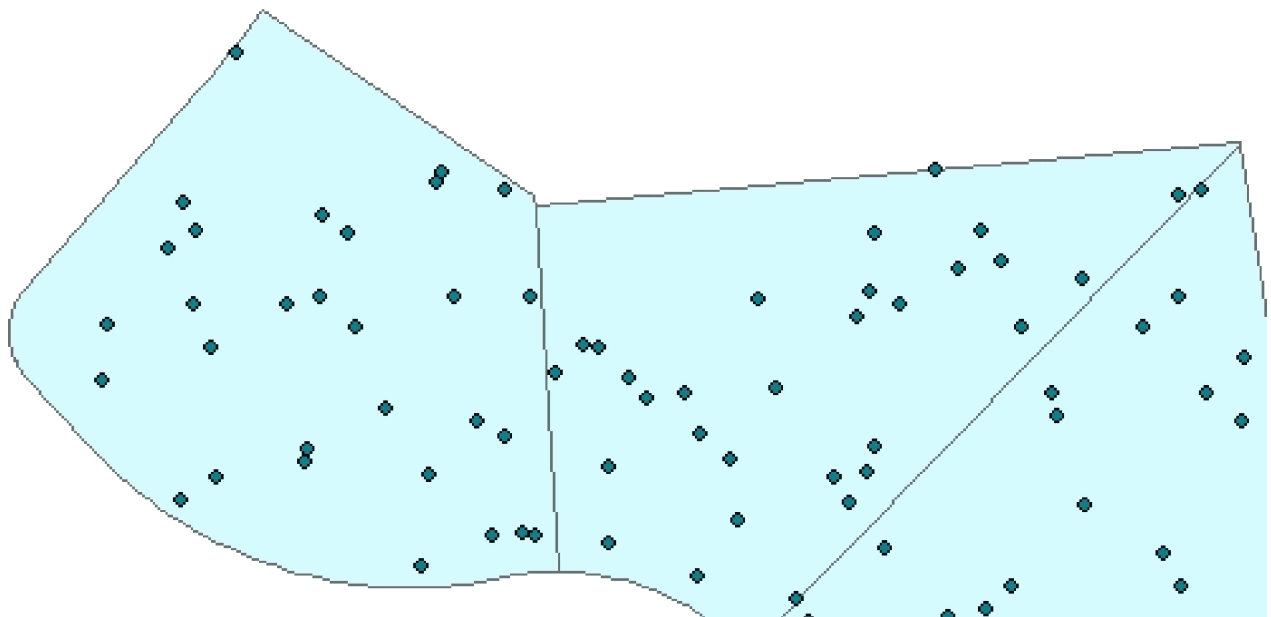
Illustration:

Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points. Distance between points = 0, Minimum distance to boundary = 0

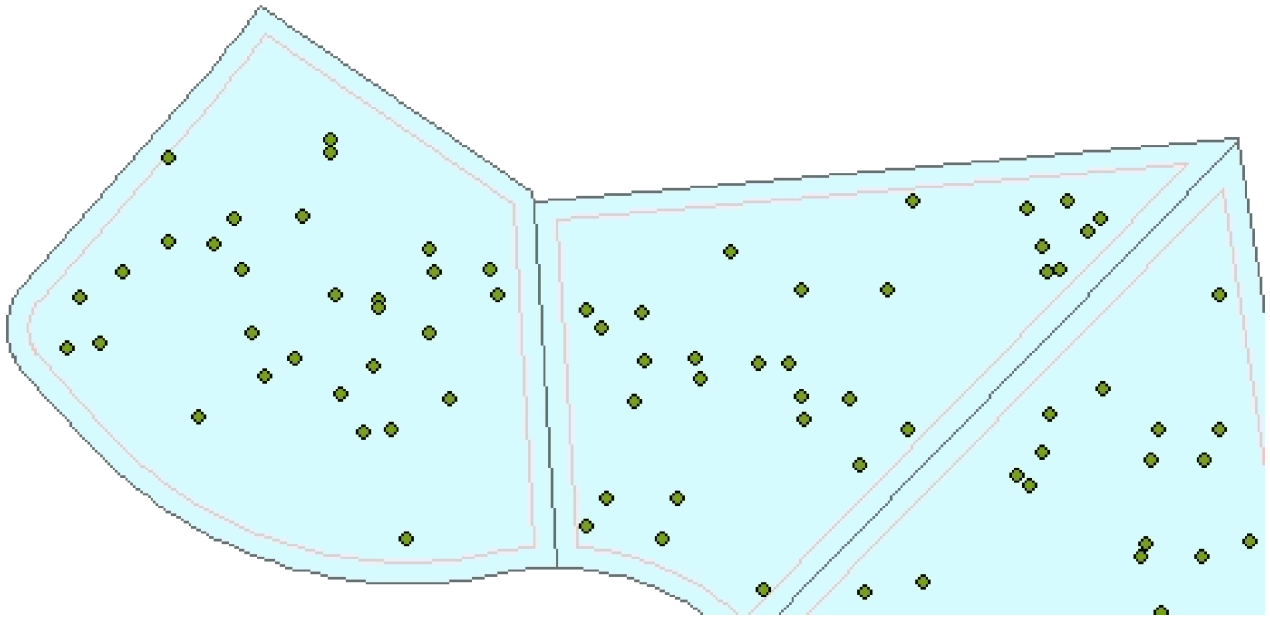


30 points per polygon.

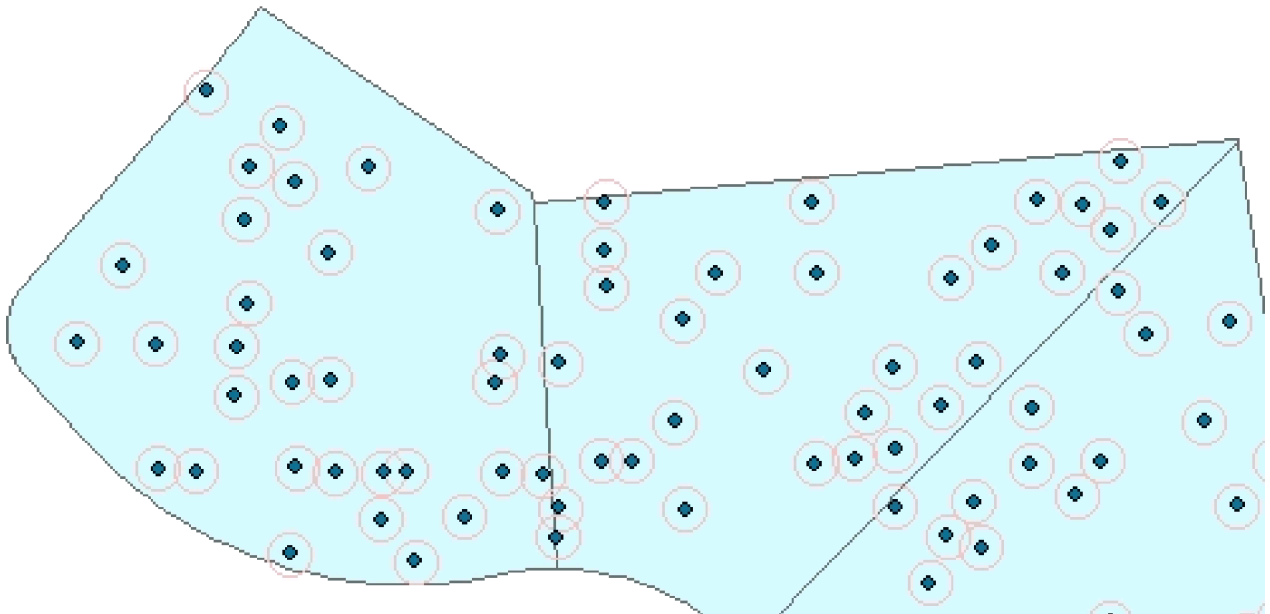
Distance between points = 0, Minimum distance to boundary = 0



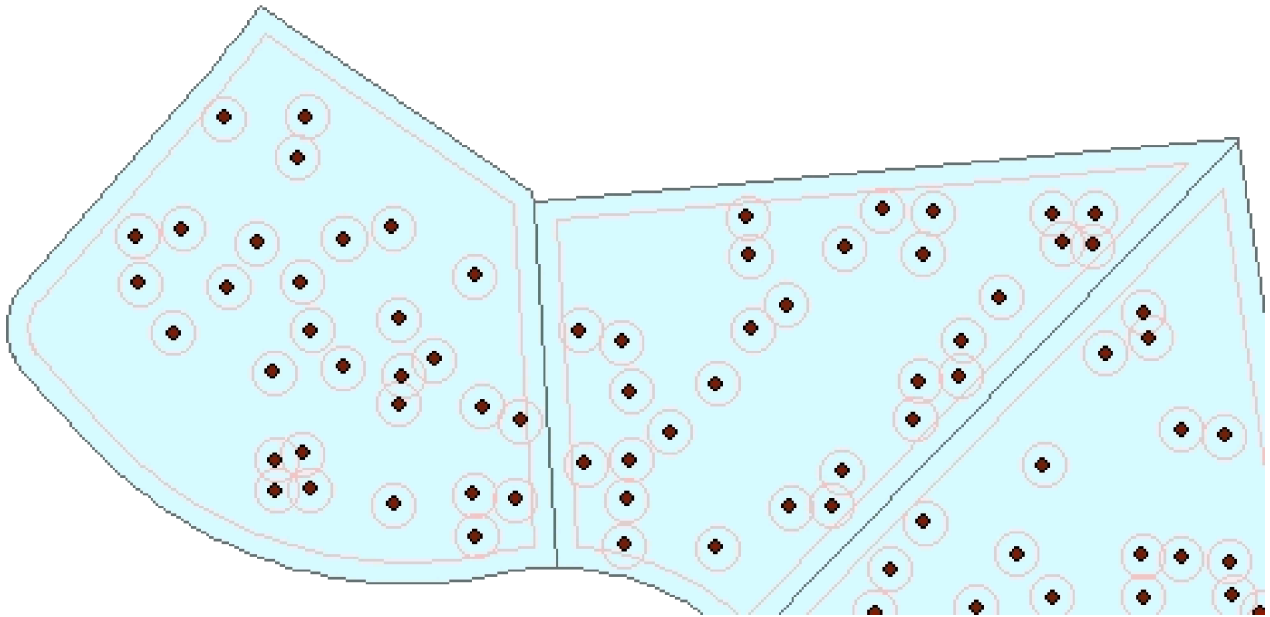
Distance between points = 0, Minimum distance to boundary = 5



30 points per polygon. Distance between points = 5, Minimum distance to boundary = 0



30 points per polygon. Distance between points = 5, Minimum distance to boundary = 5



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RandomPointsInPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Number Points}	An Integer representing the number of points per polygon to be created.
{Number Points Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the number points per polygon to be created.
{Distance Between}	A Double representing the minimum distance between the points to be generated.
{Distance to Boundary}	A Double representing the minimum distance to the polygon boundary

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "RandomPointsInPolygons", "input dataset", "output dataset", "Number Points", "Number Points Field", "Distance Between", "Distance to Boundary"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "RandomPointsInPolygons" "input dataset" "output dataset" "Number Points" "Number Points Field" "Distance Between" "Distance to Boundary"</code>
.NET using ETGWOutX.dll	<code>RandomPointsInPolygons(input dataset, output dataset, Number Points, Number Points Field, Distance Between, Distance to Boundary)</code>
ArcPy	<code>arcpy.RandomPointsInPolygons(input dataset, output dataset, "Number Points", "Number Points Field", "Distance Between", "Distance to Boundary")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Point Grids In Polygons

[Running programmatically](#)

Generates uniform (regularly spaced) points located in the polygons of the input polygon dataset. The distance between the points for each polygon can be the same or different - based on the values in a numeric field of the input polygon feature class. The user can specify the rotation angle for the resulting point grid.

Inputs:

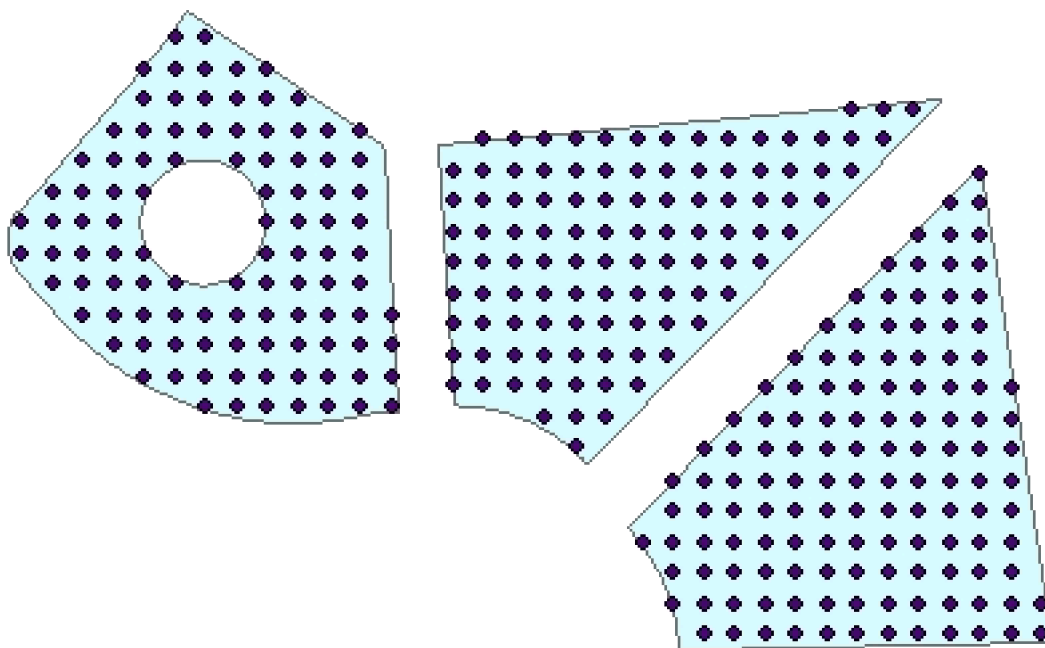
- A polygon feature class
- The distance between the points for each polygon can be input in one of the following ways
 - A numeric field which values will be used to get the distance between points for each polygon.
 - A constant number defining the distance between points for all polygons.
- Rotation angle
 - Constant for all polygons - (in degrees starting from North clockwise)
 - From field - different rotation angle for each field (in degrees starting from North clockwise)
 - Along the longest axis of each polygon
 - Along the longest side of each polygon

Outputs:

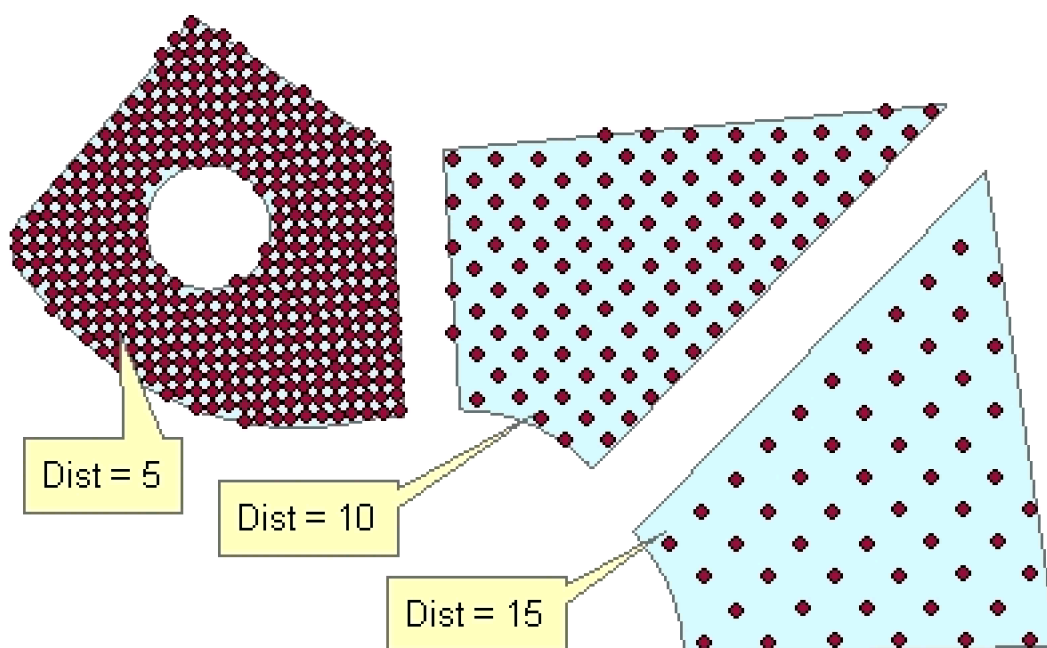
- New Point feature class.
- The following fields are added to the attribute table of the resulting feature class.
 - [ET_ID] - the ID of the original polygon

Illustration:

Distance between points = 10, Rotational angle = 0



Original polygons labeled with the values in field to be used as a source for getting the distance between the points. Rotational angle = "Along the longest Side"



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointGridsInPolygons

<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Size}	A Double representing distance between the grid points.
{Size Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the distance between the grid points.
<Angle From>	<p>Required. A String -This parameter defines what will be the source rotation of the grids:</p> <ul style="list-style-type: none"> • "Constant" - user defined value for all grids. • "FromField" - different for each grid based on a value in the attribute table. • "LongestAxis" - the grids will be oriented along the longest axis of each polygon. • "LongestSide" - the grids will be oriented along the longest side of each polygon.
{Angle}	A Double representing the rotation angle of the grids
{Angle Field}	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the angle of the grids to be created.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PointGridsInPolygons", "input dataset", "output dataset", "Size", "", "Angle From", "Angle", ""])

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointGridsInPolygons" "input dataset" "output dataset" "Size" "" "Angle From" "Angle" ""
.NET using ETGWOutX.dll	PointGridsInPolygons(input dataset, output dataset, Size, "", Angle From, Angle, "")
ArcPy	arcpy.PointGridsInPolygons(input dataset, output dataset , "Size", "", "Angle From", "Angle", "")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Square Grids In Polygons

[Running programmatically](#)

Generates square grids located in the polygons of the input polygon dataset. The cell size for each polygon can be the same or different - based on the values in a numeric field of the input polygon feature class. The user can specify the rotation angle for the resulting square grids.

Inputs:

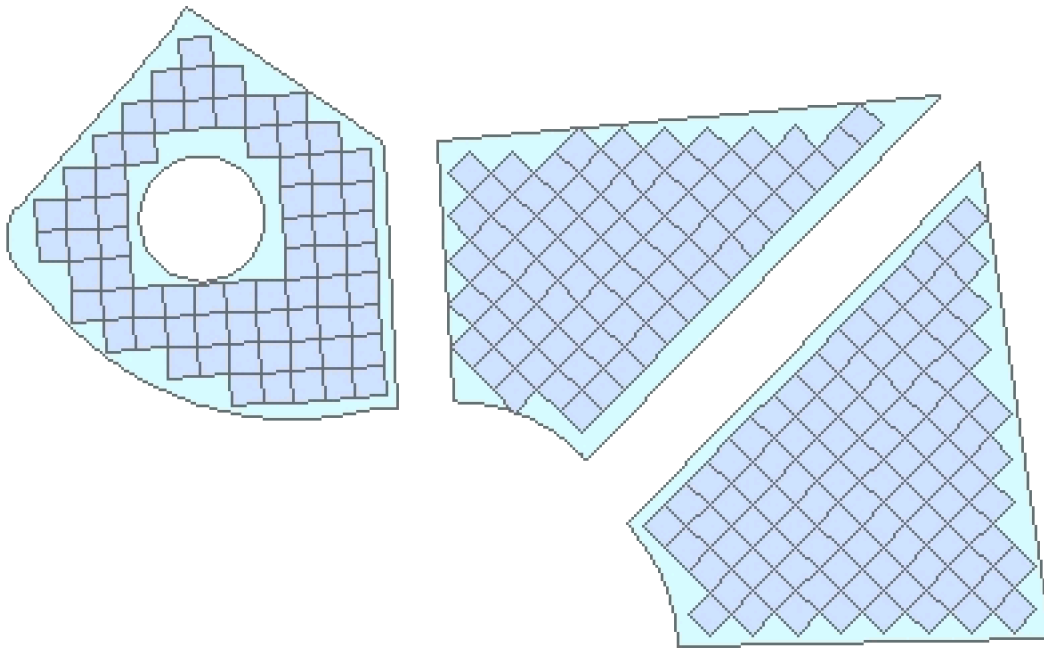
- A polygon feature class
- The cell size of the grid for each polygon can be input in one of the following ways
 - A numeric field which values will define the cell size for the grid for each polygon.
 - A constant number that defines the cell size (the same for all polygons)
- Rotation angle
 - Constant for all polygons - (in degrees starting from North clockwise)
 - From field - different rotation angle for each field (in degrees starting from North clockwise)
 - Along the longest axis of each polygon
 - Along the longest side of each polygon
- Squares completely inside the polygons
 - if TRUE - no square will intersect the polygon boundary.
 - If FALSE - the centers of the squares will be inside the polygons, but the squares might intersect the polygon boundary

Outputs:

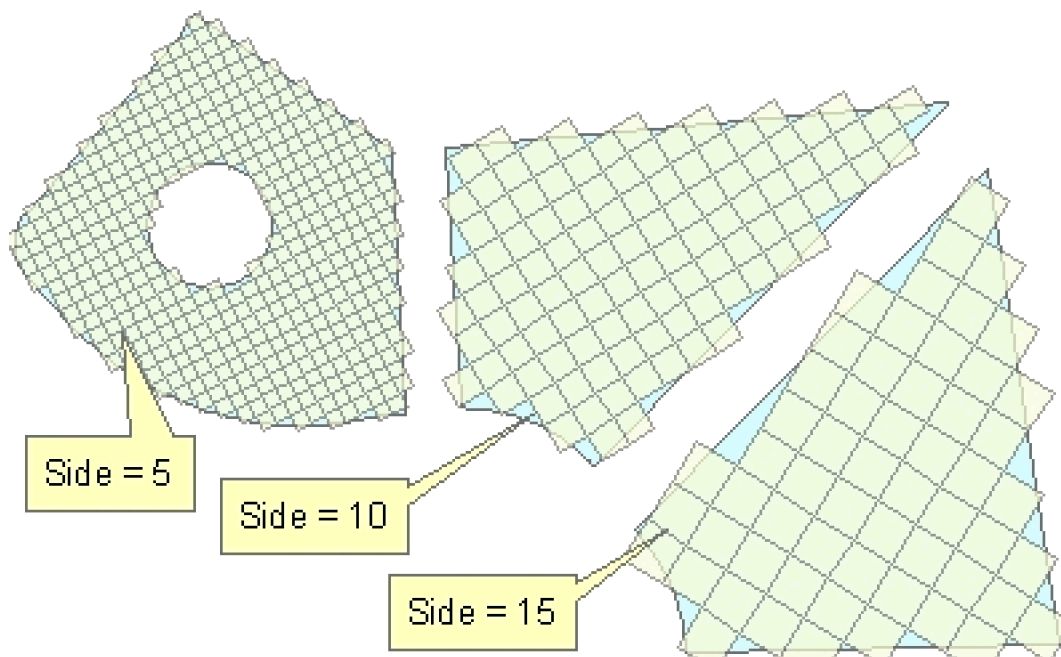
- New Polygon feature class.
- The following fields are added to the attribute table of the resulting feature class.
 - [ET_ID] - the ID of the original polygon

Illustration:

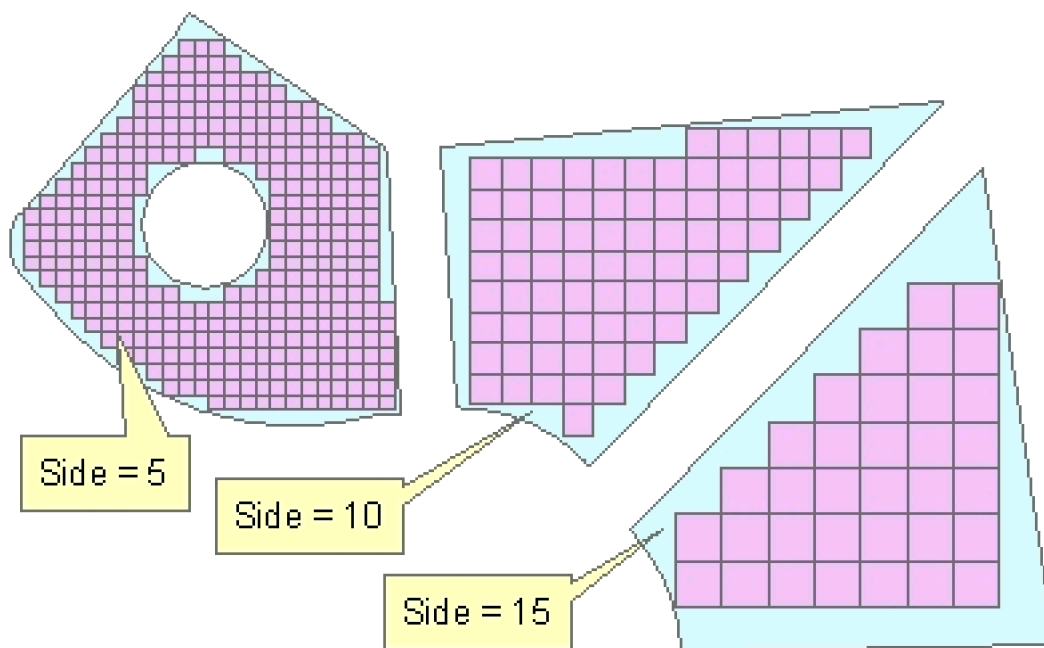
Cell size = 10, Rotational angle = "Along the longest side", Completely inside = TRUE



Original polygons labeled with the values in field to be used as a source for the Cell Size. Rotational angle = "Along the longest Axis", Completely inside = FALSE



Original polygons labeled with the values in field to be used as a source for the Cell Size. Rotational angle = 0, Completely inside = TRUE



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SquareGridsInPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Size}	A Double representing grid cell size
{Size Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the cell size of the grid polygons to be created.
<Angle From>	Required. A String -This parameter defines what will be the source rotation of the grids: <ul style="list-style-type: none"> "Constant" - user defined value for all grids.

	<ul style="list-style-type: none"> • "FromField" - different for each grid based on a value in the attribute table. • "LongestAxis" - the grids will be oriented along the longest axis of each polygon. • "LongestSide" - the grids will be oriented along the longest side of each polygon.
{Angle}	A Double representing the rotation angle of the grids
{Angle Field}	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the angle of the grids to be created.
{Inside Only}	A Boolean indicating whether all cells to be only inside the polygons.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SquareGridsInPolygons", "input dataset", "output dataset", "Size", "", "Angle From", "Angle", "", "Inside Only"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SquareGridsInPolygons" "input dataset" "output dataset" "Size" "" "Angle From" "Angle" "" "Inside Only"</pre>
.NET using ETGWOuX.dll	<code>SquareGridsInPolygons(input dataset, output dataset, Size, "", Angle From, Angle, "", Inside Only)</code>
ArcPy	<code>arcpy.SquareGridsInPolygons(input dataset, output dataset , "Size", "", "Angle From", "Angle", "", "Inside Only")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Uniform Points In Polygons

[Running programmatically](#)

Generates uniform (regularly spaced) points located in the polygons of the input polygon dataset. The number of points per polygon can be constant or different for each polygon - based on the values in a numeric field of the input polygon feature class. The distance between the points is interpolated for each polygon. The user can specify the rotation angle for the resulting point grid.

Inputs:

- A polygon feature class
- The number of points per polygon can be input in one of the following ways
 - A numeric field which values will be used to get the number of points to be generated per polygon.
 - A constant number
- Optional: Minimum Distance from boundary - no point will be generated that is closer than this tolerance to the boundary of the polygon.
- Rotation angle
 - Constant for all polygons - (in degrees starting from East anti-clockwise)
 - From field - different rotation angle for each field (in degrees starting from East anti-clockwise)
 - Along the longest axis of each polygon
 - Along the longest side of each polygon

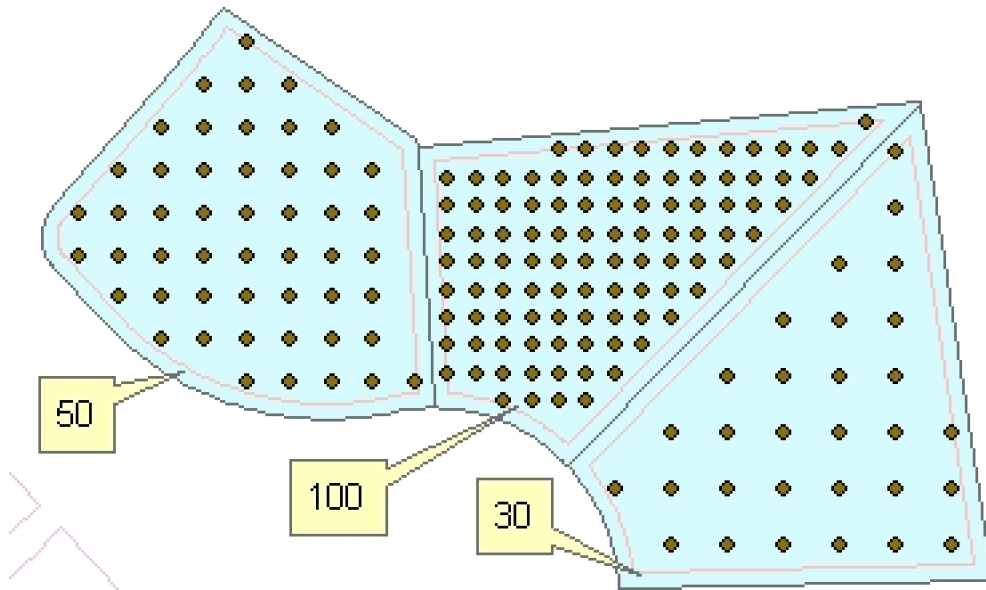
Outputs:

- New Point feature class.
- The following fields are added to the attribute table of the resulting feature class.
 - [ET_ID] - the ID of the original polygon
 - [ET_X] - the X coordinate of the point
 - [ET_Y] - the Y coordinate of the point
 - [ET_Dist] - the distance between the generated points (constant for the points in

each polygon)

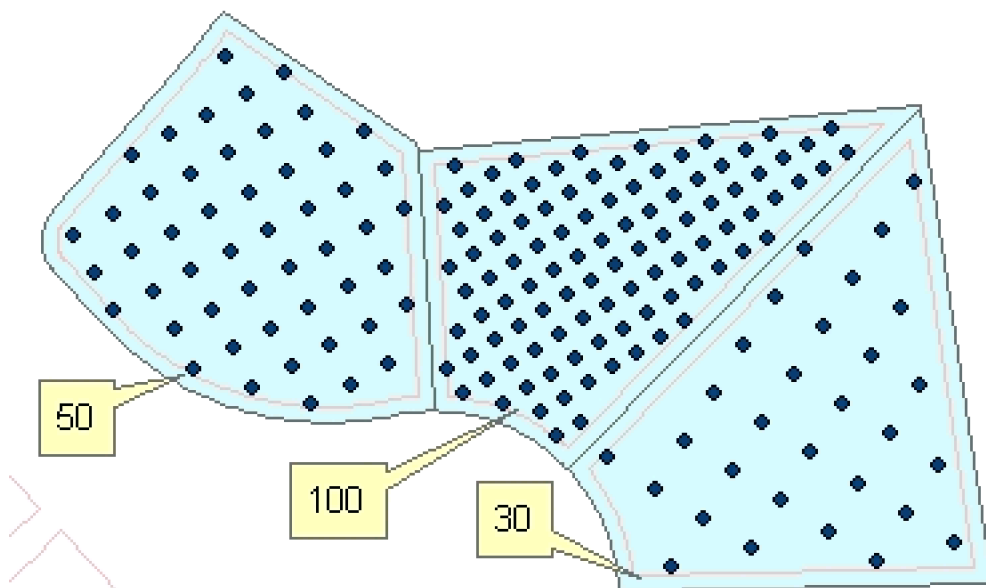
Illustration:

Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points. Rotational angle = 0, Minimum distance to boundary = 5

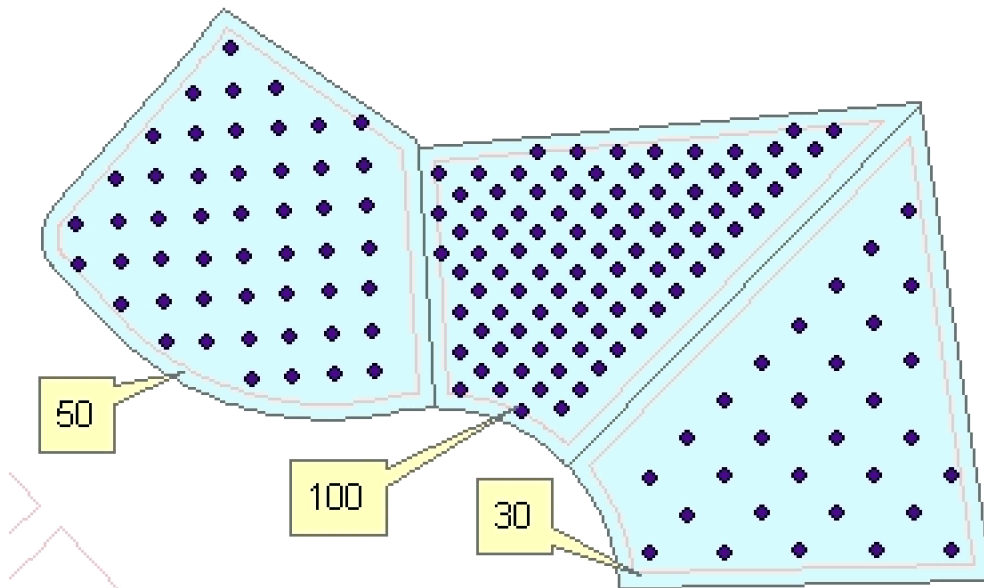


Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points.

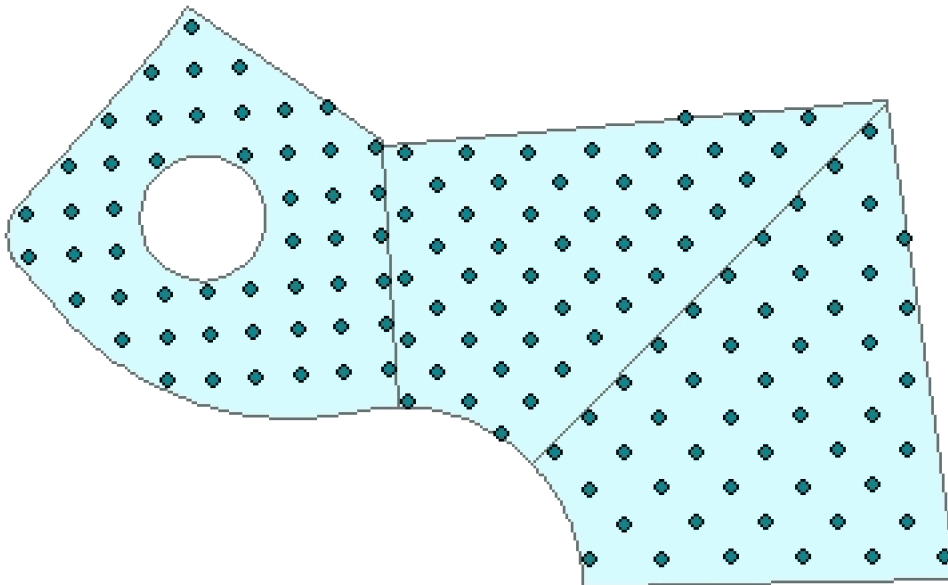
Rotational angle = "Along the longest Axis", Minimum distance to boundary = 5



Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points. Rotational angle = "Along the longest side", Minimum distance to boundary = 5



50 points per polygon. Rotational angle = "Along the longest side", Minimum distance to boundary = 0



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	UniformPointsInPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Angle From>	<p>Required. A String -This parameter defines what will be the source rotation of the grids:</p> <ul style="list-style-type: none"> • "Constant" - user defined value for all grids. • "FromField" - different for each grid based on a value in the attribute table. • "LongestAxis" - the grids will be oriented along the longest axis of each polygon. • "LongestSide" - the grids will be oriented along the longest side of each polygon.
{Number Points}	An Integer representing the number of points per polygon to be created.
{Number Points Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the number points per polygon to be created.
{Angle}	A Double representing the rotation angle of the grids
{Angle Field}	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the angle of the grids to be created.
{Distance to Boundary}	A Double representing the minimum distance to the polygon boundary

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program

Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "UniformPointsInPolygons", "input dataset", "output dataset", "Angle From", "Number Points", "Number Points Field", "Angle", "", "Distance to Boundary"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "UniformPointsInPolygons" "input dataset" "output dataset" "Angle From" "Number Points" "Number Points Field" "Angle" "" "Distance to Boundary"</code>
.NET using ETGWOuX.dll	<code>UniformPointsInPolygons(input dataset, output dataset, Angle From, Number Points, Number Points Field, Angle, "", Distance to Boundary)</code>
ArcPy	<code>arcpy.UniformPointsInPolygons(input dataset, output dataset , "Angle From", "Number Points", "Number Points Field", "Angle", "", "Distance to Boundary")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Tiles

[Running programmatically](#)

Creates a regular polygons grid with user defined.

- extents, tile shape and size
- origin, number rows and columns, tile shape and size .

Inputs:

- Tiles creation options:
 - Extents
 - Origin, number rows and columns.
- Output spatial reference - can be assigned based on a reference layer used.
- Tiles shape
 - Triangle
 - Square
 - Hexagon
- Cell size.
- Tile Type option - depending on the user input the size parameter can represent
 - The side of the polygon
 - The radius of the circle inscribed in the polygon
 - The radius of the circle circumscribed around the polygon.

- Shape orientation (for square and hexagon tiles only) - see examples below
 - Flat
 - Pointy

Outputs:

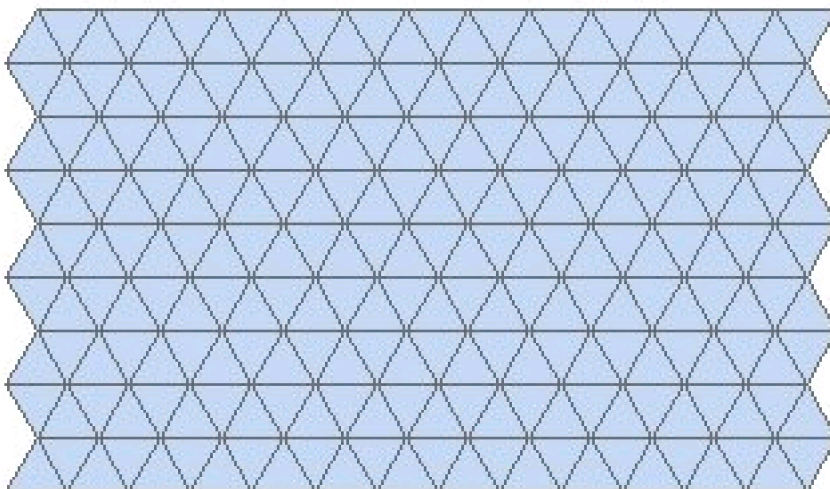
- New Polygon layer

Notes:

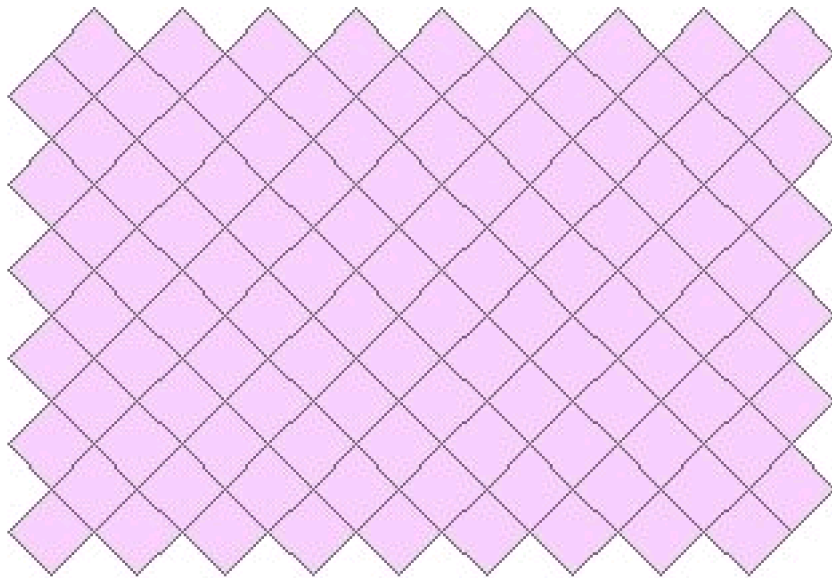
- The initial extent of the tiles is defined by the extents of the selected reference layer or manually
- The cell size will be in the units of the reference layer or "Unknown" if no reference layer is used.
- In order to avoid incorrect inputs, the size of the grid is limited to 8,000,000 cells
- A ET_Index field will be added to the attribute table. The values will indicate the index of each Grid cell. The tile in the bottom left corner will have an index of "0-0"

Examples:

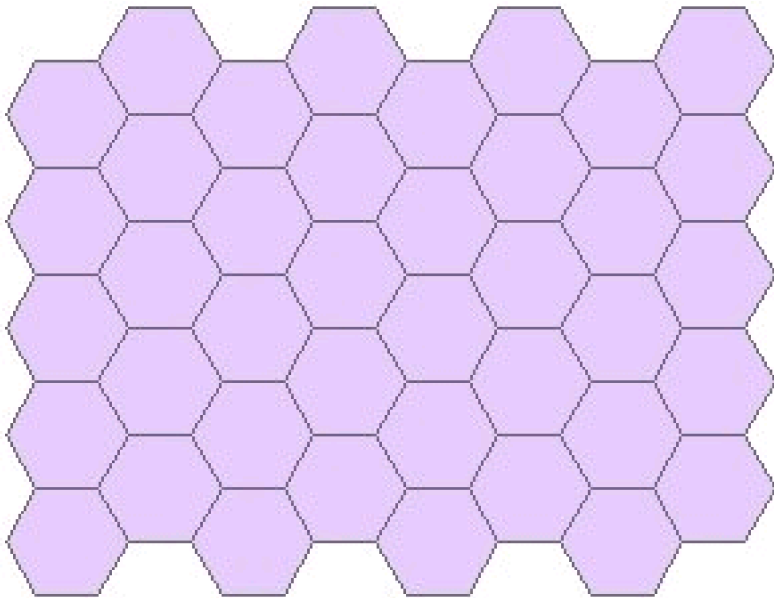
Shape = Triangle



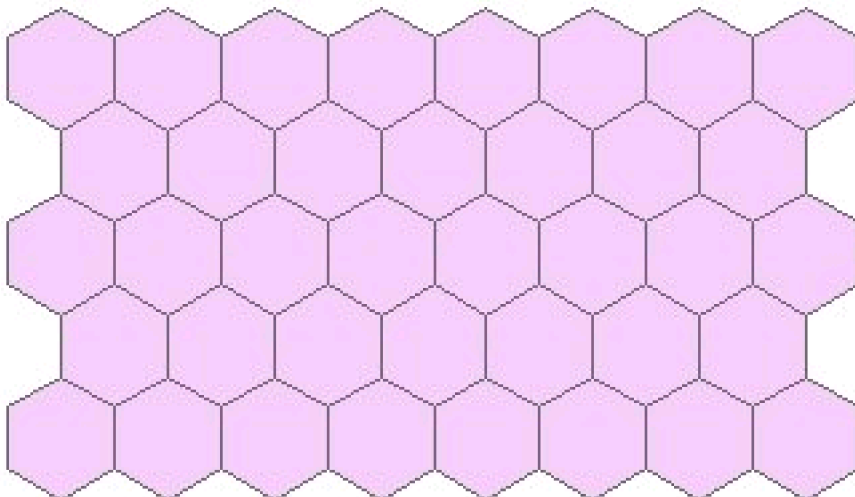
Shape = Square, Orientation = "Pointy"



Shape = Hexagon, Orientation = "Flat"



Shape = Hexagon, Orientation = "Pointy"



Running Programmatically

[\(Go to TOP\)](#)

Two different functions are available for the Vector Grid creation

Create Tiles Extent

Parameters

Expression	Explanation
Function Name	CreateTilesExtent
<output dataset>	A String - the full name of the output layer.
<Tile Type>	Required. A String indicating the type of the tiles to be created. Valid values: <ul style="list-style-type: none">• "Triangle"• "Square"• "Hexagon"
<Size Represents>	A String indicating the meaning of the SIZE parameter. Valid values: <ul style="list-style-type: none">• "Side"• "RadiusIn"• "RadiusOut"
<Tile Size>	A Double - the size of the tile.
<Orientation>	A String indicating the orientation of the tiles (see examples above). Valid values: <ul style="list-style-type: none">• "Flat"

	<ul style="list-style-type: none"> • "Pointy"
<Extents From Reference>	A Boolean indicating whether the extents of the grid will be taken from a reference dataset.
{Reference Dataset}	A String - the full name of the reference dataset.
{MinX}	A Double - minimum X of the extent.
{MinY}	A Double - minimum Y of the extent.
{MaxX}	A Double - maximum X of the extent.
{MinX}	A Double - maximum Y of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CreateTilesExtent", "output dataset", "Tile Type", "Size Represents", "Tile Size", "Orientation", "Extents From Reference", "Reference Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateTilesExtent" "output dataset" "Tile Type" "Size Represents" "Tile Size" "Orientation" "Extents From Reference" "Reference Dataset"
.NET using ETGWOutX.dll	CreateTilesExtent(output dataset, Tile Type, Size Represents, Tile Size, Orientation, Extents From Reference, Reference Dataset)
ArcPy	arcpy.CreateTilesExtent(output dataset, "Tile Type", "Size Represents", "Tile Size", "Orientation" "Extents From Reference", "Reference Dataset")

[\(Go to TOP\)](#)

Create Tiles Origin

Parameters

Expression	Explanation
Function Name	CreateTilesOrigin
<output dataset>	A String - the full name of the output layer.
<Tile Type>	Required. A String indicating the type of the tiles to be created. Valid values: <ul style="list-style-type: none">• "Triangle"• "Square"• "Hexagon"
<Number Columns>	An Integer - the number of columns to be created.
<Number Rows>	An Integer - the number of rows to be created.
<Size Represents>	A String indicating the meaning of the SIZE parameter. Valid values: <ul style="list-style-type: none">• "Side"• "RadiusIn"• "RadiusOut"
<Tile Size>	A Double - the size of the tile.
<Orientation>	A String indicating the orientation of the tiles (see examples above). Valid values: <ul style="list-style-type: none">• "Flat"

	<ul style="list-style-type: none"> • "Pointy"
<Origin From Reference>	A Boolean indicating whether the origin point of the grid will be taken from a reference dataset.
{Reference Dataset}	A String - the full name of the reference dataset.
{Lower LeftX}	A Double - X of the lower left corner of the extent.
{Lower LeftY}	A Double - Y of the lower left corner of the extent.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CreateTilesOrigin", "output dataset", "Tile Type", "Number Columns", "Number Rows", "Size Represents", "Tile Size", "Origin From Reference", "Reference Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateTilesOrigin" "output dataset" "Tile Type" "Number Columns" "Number Rows" "Size Represents" "Tile Size" "Origin From Reference" "Reference Dataset"
.NET using ETGWOutX.dll	CreateTilesOrigin(output dataset, Tile Type, Number Columns, Number Rows, Size Represents, Tile Size, Origin From Reference, Reference Dataset)
ArcPy	arcpy.CreateTilesOrigin(output dataset, "Tile Type", "Number Columns", "Number Rows", "Size Represents", "Tile Size", "Origin From Reference", "Reference Dataset")

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Points

[Running programmatically](#)

Removes the duplicate points from a point feature data set.

Inputs:

- A point feature layer

Outputs:

- New Point dataset
 - Each set of duplicate points (that have exactly the same location) will be replaced by a single point. This point will carry the attributes of one of the original points.
- Optional Point dataset that identifies the duplicates in the input data set.
 - The attribute table of the duplicates feature class has all the fields from the input data set.
 - The attributes of the points are these that have not been preserved in the clean feature class. Example:. If two points with attributes "A" and "B" have exactly the same location. The clean feature class will contain only one of them e.g. "A". The duplicates feature class will contain the other one -"B"

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanPoints
<input dataset>	A String representing the input layer. Must be of Polygon type.

<output dataset>	A String - the full name of the output layer.
{Duplicates Dataset}	A String - the full name of the output dataset that identifies the points removed as duplicates. (A dataset with the same full name should not exist) .

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CleanPoints", "input dataset", "output dataset", "Duplicates Dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanPoints" "input dataset" "output dataset" "Duplicates Dataset"</code>
.NET using ETGWOutX.dll	<code>CleanPoints(input dataset, output dataset, Duplicates Dataset)</code>
ArcPy	<code>arcpy.CleanPoints(input dataset, "output dataset" , "Duplicates Dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Connect Points

[Running programmatically](#)

Connects with lines each point from a point dataset to every other point from the same dataset that is closer to the point than the user defined cut off distance.

- Assigns the IDs of the From and To Points to each line.
- Calculates the length and angle of the connector lines.

Inputs:

- A Point layer
- Cutoff distance - in the units of the spatial reference of the input dataset. Points that are farther to each other than this tolerance will not be connected.
- Max number of points to connect.
- Add duplicate lines. If false only one line will be created between two points closer to each other than the Cutoff distance.

Outputs:

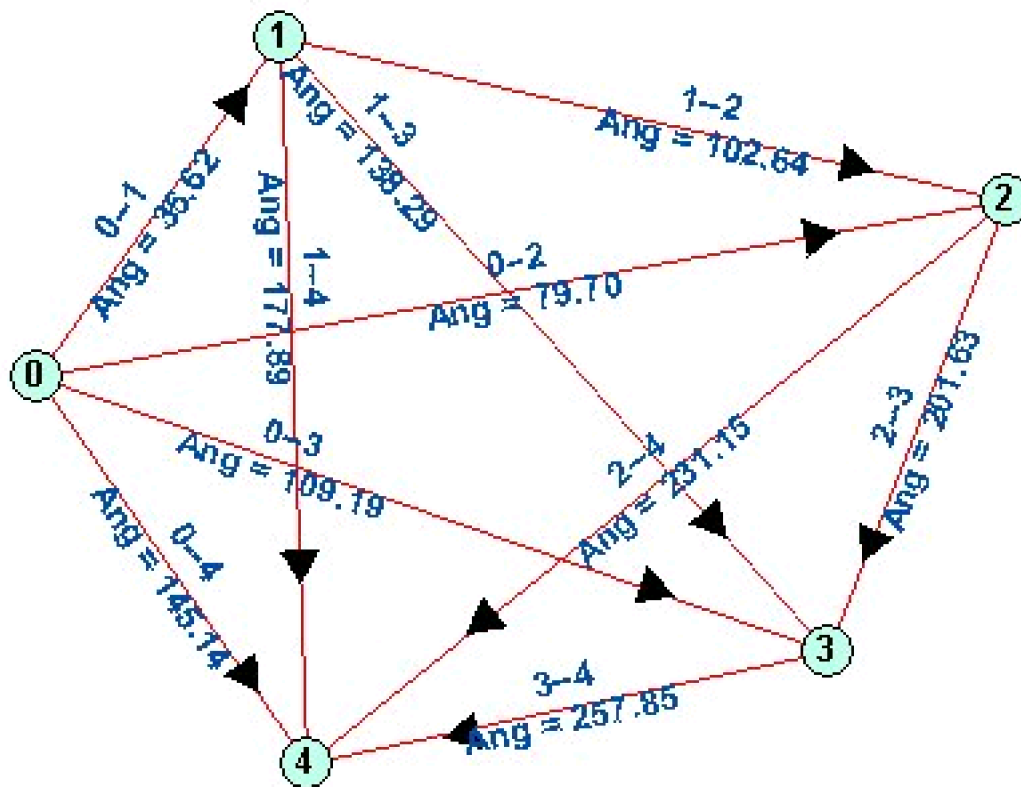
- New point layer. Fields:
 - [ET_From] - the ID of the start point of the line
 - [ET_To] - the ID of the end point of the line
 - [ET_Length] - the length of the line
 - [ET_Angle] - the angle of the line

Notes:

- If no Cutoff distance tolerance is specified each point will be connected to all other points

- Important: The number of connector lines created if no cutoff distance is used can be calculated using the formulae $N = n \times (n-1)/2$ (N - number of lines, n - number of input points). For example 1,000 points will create 499,500 lines, 10,000 points will create 49,995,000 lines, which is not a dataset you want to handle.

Example:



- Point 0 connects to Points 1, 2, 3, 4
- Point 1 connects to Points 2, 3, 4 (it already has been connected to Point 0)
- Point 2 connects to Points 3, 4 (it already has been connected to Points 0 and 1)
-

The angle is calculated in North Azimuth direction



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ConnectPoints
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<Number Neighbours>	An Integer representing the maximum number of points to connect to each point.
{CutOff Distance}	A Double representing the maximum distance to a neighbor to be connected. The units of the tolerance are the units of spatial reference of the input dataset.
{Add Duplicates}	A Boolean indicating whether duplicate lines will be added to the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ConnectPoints", "input dataset", "output dataset", "Number Neighbours", "CutOff Distance", "Add Duplicates"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ConnectPoints" "input dataset" "output dataset" "Number Neighbours" "CutOff Distance" "Add Duplicates"</code>
.NET using ETGWOutX.dll	<code>ConnectPoints(input dataset, output dataset, Number Neighbours, CutOff Distance, Add Duplicates)</code>
ArcPy	<code>arcpy.ConnectPoints(input dataset, output dataset, Number Neighbours, CutOff Distance, Add Duplicates)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Disperse Points

[Running programmatically](#)

Disperses (separates) the coincident points. The first point in a location preserves its coordinates. Every next point found in the same location is moved randomly within user defined maximum offset distance from its original location.

Inputs:

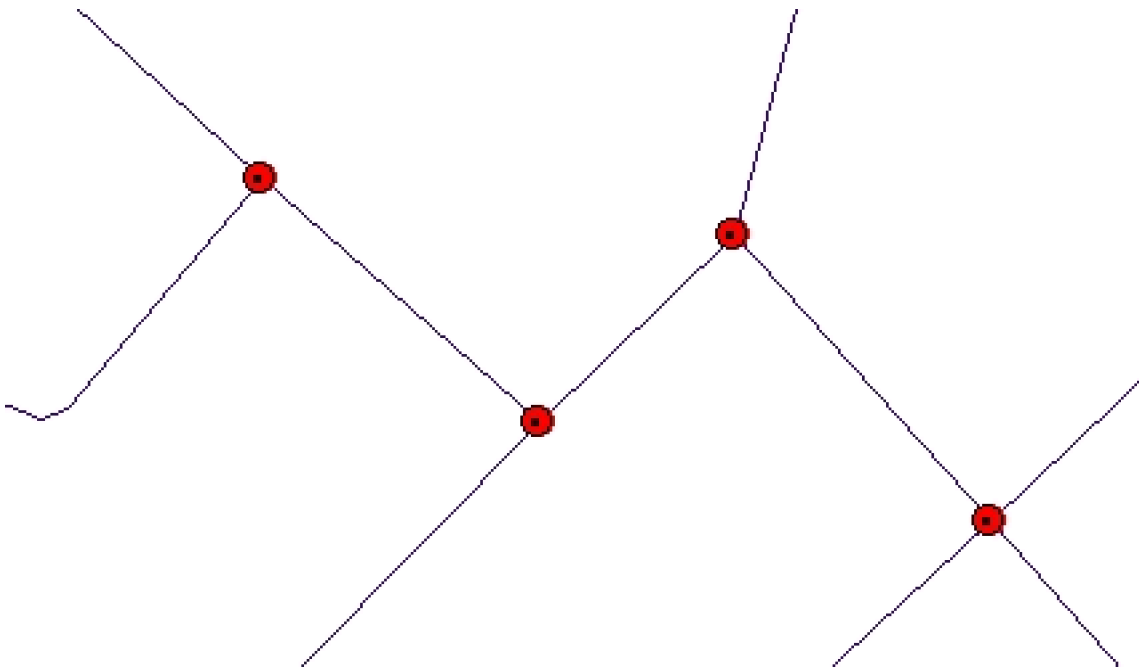
- A Point feature class
- Maximum allowed offset - the duplicate points will move no further than this tolerance from their original location.
- Dispersion method:
 - Random - the points will be relocated randomly within the specified allowed distance
 - Regular - the resulting points will be placed on a circle with radius the allowed offset distance.

Outputs:

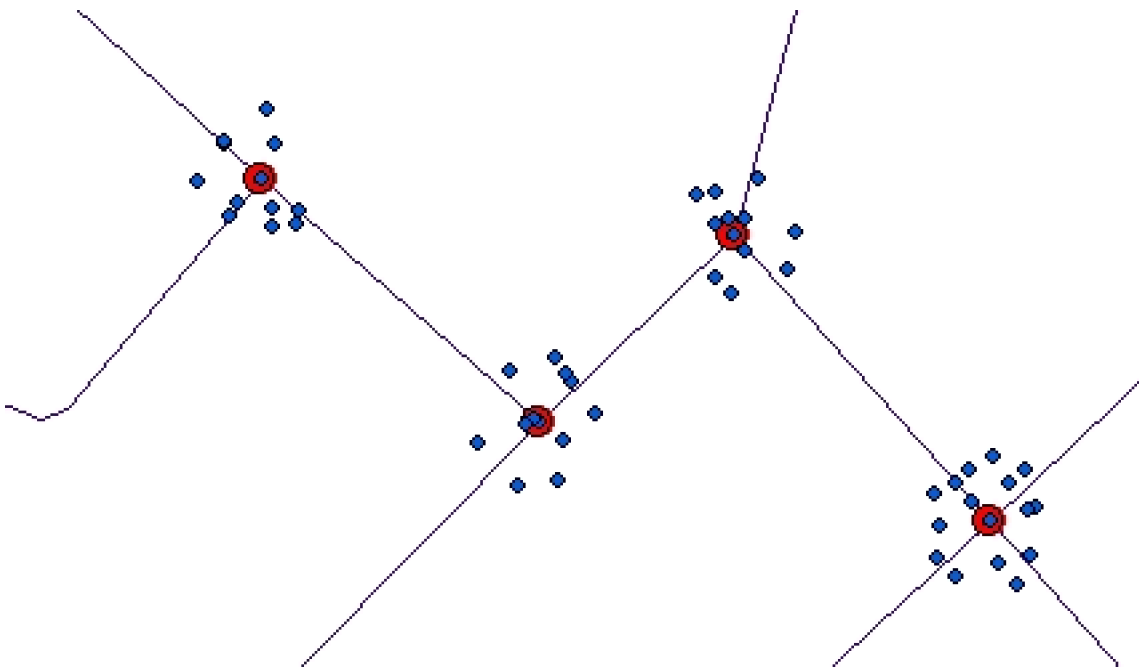
- New point feature class. Fields:
 - The attributes of the original features will be preserved.
 - [ET_Status] field will be added. The values in this field will indicate whether the resulting point is in its original position or has been relocated.

Examples:

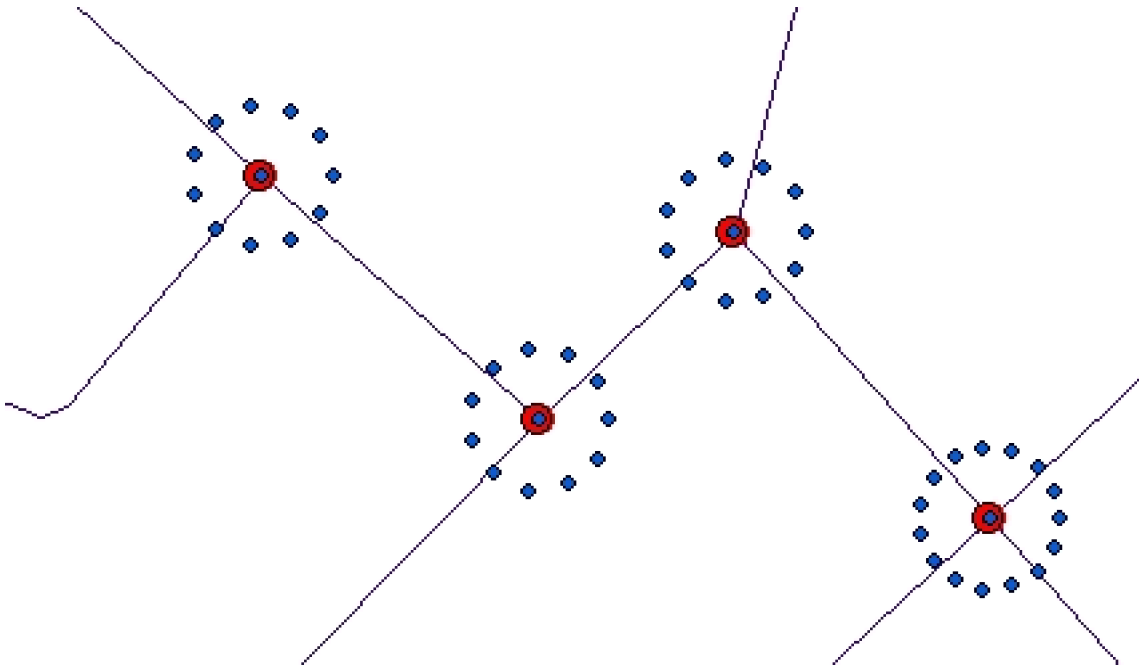
Input Dataset



Dispersed Randomly



Dispersed Regularly



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	DispersePoints
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Tolerance>	A Double representing the Maximum allowed offset - the duplicate points will move no further than this tolerance from their original location.
<Disperse Method>	A string defining how the points will be dispersed. Valid values - "Random" or "Regular".

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "DispersePoints", "input dataset", "output dataset", "Tolerance", "Disperse Method"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "DispersePoints" "input dataset" "output dataset" "Tolerance" "Disperse Method"</pre>
.NET using ETGWOuX.dll	<code>DispersePoints(input dataset, output dataset, Tolerance, Disperse Method)</code>
ArcPy	<code>arcpy.DispersePoints(input dataset, output dataset, "Tolerance", "Disperse Method")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Perpendiculars from Points to Polylines

[Running programmatically](#)

Draws a perpendicular polyline from each point to the closest polyline from the reference layer and calculates several attributes for each perpendicular line.

Inputs:

- A point layer
- A reference polyline layer
- Search tolerance.
- Keep Input Spatial Reference - If selected the output will have the spatial reference of the input dataset, else the spatial reference of the reference dataset will be used

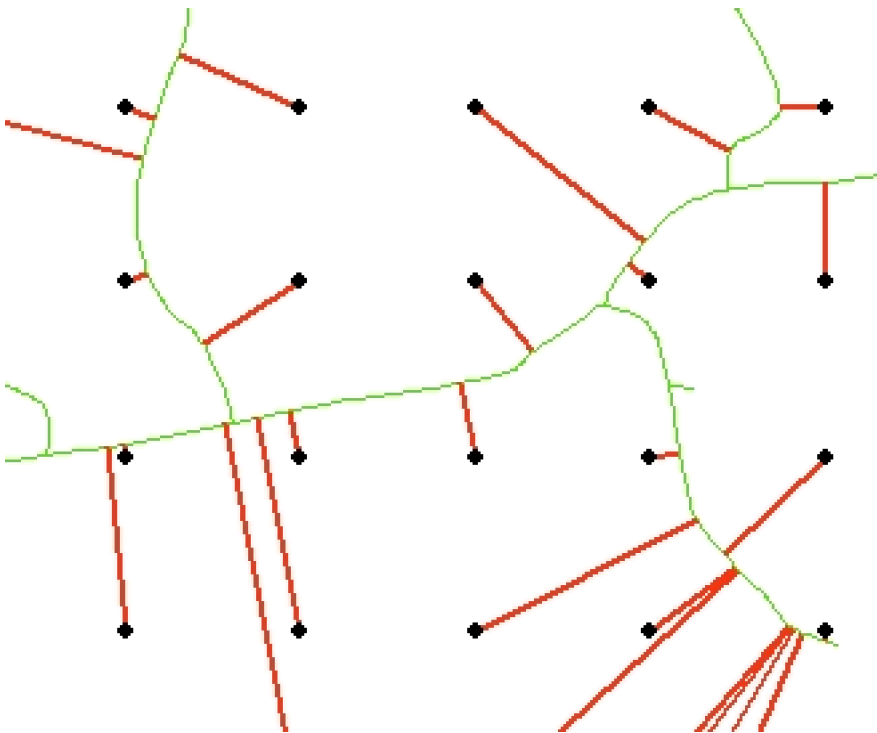
Outputs:

- New Polyline layer with lines from the source points and perpendicular to the closest polyline from the reference layer.
- The attributes of the original points are transferred to the resulting lines.
- The following fields are added to the attribute table of the resulting layer.
 - [ET_Dist] - the distance from the original point to the closest polyline (the length of the perpendicular line)
 - [ET_Pos] - the relative position of the original point along the closest polyline (in percent)
 - [ET_Angle] - the angle (0 to 360) of the resulting perpendicular line in degrees starting North clockwise
 - [ET_Station] - the absolute position of the original point along the closest polyline (in the units of the spatial reference of the input point layer)

Notes:

- The units of the Search Tolerance should be the units of spatial reference of the input dataset if the user has selected the Keep Source Spatial Reference option. Otherwise - the units of spatial reference of the reference dataset.
- The Source and the Snap datasets can have different spatial references as long as they have the same Geographic Coordinate systems.

Illustration:



[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PerpendicularsToPolylines
<input dataset>	A String representing the input layer. Must be of Point type.
<reference dataset>	A String representing the reference layer. Must be of Polyline type
<output dataset>	A String - the full name of the output layer.
<search tolerance>	A Double representing the Search tolerance to be used. The units of the tolerance are the units of spatial reference of the input dataset if

	KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the reference dataset.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PerpendicularsToPolylines", "input dataset", "reference dataset", "output dataset", "search tolerance", "KeepSourceSref"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PerpendicularsToPolylines" "input dataset" "reference dataset" "output dataset" "search tolerance" "KeepSourceSref"
.NET using ETGWOuX.dll	PerpendicularsToPolylines(input dataset,reference dataset, output dataset, search tolerance, KeepSourceSref)
ArcPy	arcpy.PerpendicularsToPolylines(input dataset, reference dataset, output dataset, "search tolerance" , "KeepSourceSref")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Point Angle and Position

[Running programmatically](#)

Identifies the closest polyline from the reference layer to each point and calculates: the angle of the closest polyline segment, the position & stationing of the point along the polyline and the distance to the polyline

Inputs:

- Point feature layer
- Reference polyline layer
- Search tolerance - the maximum distance to search for features in the distance layer
- Keep Input Spatial Reference - If selected the output will have the spatial reference of the input dataset, else the spatial reference of the reference dataset will be used

Outputs:

- A new Point feature class. The attribute table of the resulting feature class will have three new fields
 - [ET_Angle] - the angle of the closest segment of the closest to the point polyline. The angle is in degrees 0.00 = North, clockwise.
 - [ET_Pos] - the distance from the start point of the closest polyline to the point along the polyline as a percentage of the total length of the polyline.
 - [ET_Station] - the actual distance from the start point of the closest polyline to the point along the polyline, measured in the map units
 - [ET_Dist] - the shortest distance from the point to the closest polyline measured in the map units
 - [ET_Side] - indicates on which side of the polyline is the point (introduced in version 10.2).
 - [ET_M]/[ET_Z] - the M(Z) value interpolated from the closest polyline (if the reference dataset is of PolylineM(Z) type)

- [ET_Closest] - the ID of the closest polyline from the reference dataset.

Notes:

- If the distance from a point to the closest feature from the distance layer is larger than the Search Tolerance then the [ET_Angle] will have a value of 0, [ET_Pos] and [ET_Station] will have values of -1
- The units of the Search Tolerance should be the units of spatial reference of the input dataset if the user has selected the Keep Source Spatial Reference option. Otherwise - the units of spatial reference of the reference dataset.
- The distances are calculated in the units of spatial reference of the input dataset if the user has selected the Keep Source Spatial Reference option. Otherwise - the units of spatial reference of the reference dataset.
- All the attributes of the input point dataset are transferred to the output

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsAngleAndPosition
<input dataset>	A String representing the input layer. Must be of Point type.
<reference dataset>	A String representing the reference layer. Must be of Polyline type
<output dataset>	A String - the full name of the output layer.
<search tolerance>	A Double representing the Search tolerance to be used. The units of the tolerance are the units of spatial reference of the input dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the reference dataset.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsAngleAndPosition", "input dataset", "reference dataset", "output dataset", "search tolerance", "KeepSourceSref"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsAngleAndPosition" "input dataset" "reference dataset" "output dataset" "search tolerance" "KeepSourceSref"</code>
.NET using ETGWOuX.dll	<code>PointsAngleAndPosition(input dataset,reference dataset, output dataset, search tolerance, KeepSourceSref)</code>
ArcPy	<code>arcpy.PointsAngleAndPosition(input dataset, reference dataset, output dataset, "search tolerance" , "KeepSourceSref")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Point Global Snap

[Running programmatically](#)

Snaps the features of a point layer to another layer (Point, Polyline or Polygon)

Inputs

- A point layer to be snapped
- A reference layer - point, polyline or polygon
- Snap tolerance
- Snap options
- Keep Input Spatial Reference - If selected the output will have the spatial reference of the input dataset, else the spatial reference of the reference dataset will be used

Outputs

- A point layer - the points from the source layer will be moved to snap to the features of the Snap Layer (if within the snap tolerances)

Options:

- Vertices: The points will be snapped to the nearest vertex of the nearest feature from the Snap layer
- Nearest: If there is a vertex closer than the snap tolerance to the point to be snapped, the point will snap to it, otherwise it will snap to the nearest edge.
- Snap to reference Z values (only if the input and output are Z enabled)

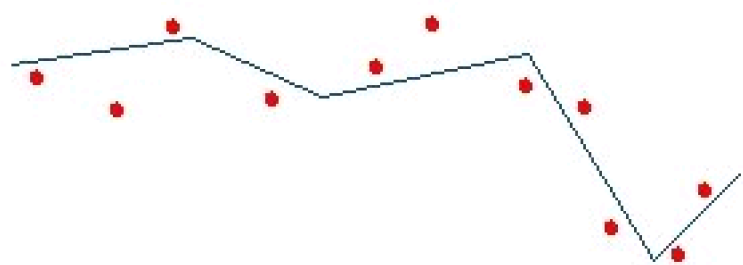
Notes:

- The units of the Snap Tolerance should be the units of spatial reference of the input dataset if the user has selected the Keep Source Spatial Reference option. Otherwise - the units of spatial reference of the reference dataset.

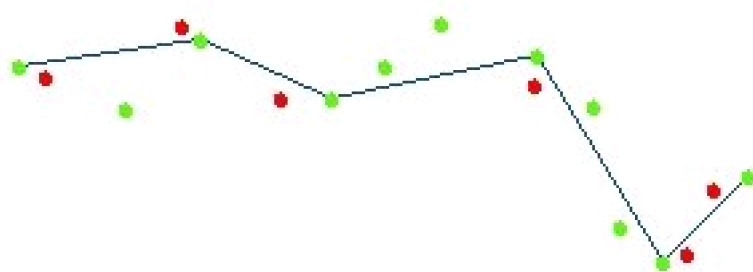
- The Source and the Snap datasets can have different spatial references as long as they have the same Geographic Coordinate systems.

Example:

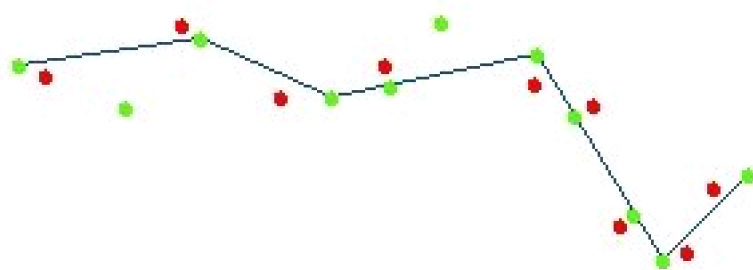
Before Snap



After Snap - Option: **Vertex**



After Snap - Option: **Nearest**



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SnapPoints
<input dataset>	A String representing the input layer. Must be of Point type.
<reference dataset>	A String representing the reference layer. Must be of Polyline type

<output dataset>	A String - the full name of the output layer.
< SnapTolerance>	A Double representing the Snap Tolerance to be used. The units of the tolerance are the units of spatial reference of the input dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the reference dataset.
{snap option}	A String indicating the snap option to be used. Valid inputs: "Vertex" or "Nearest". Default = "Nearest"
{snapToZ}	A Boolean indicating whether to snap to Z values (if the reference dataset has Z values). Default = False.
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "SnapPoints", "input dataset", "reference dataset", "output dataset", " SnapTolerance", "snap option", "snapToZ", "KeepSourceSref"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "SnapPoints" "input dataset" "reference dataset" "output dataset" "SnapTolerance" "snap option" "snapToZ" "KeepSourceSref"
.NET using ETGWOuX.dll	SnapPoints(input dataset,reference dataset, output dataset, SnapTolerance, snap option, snapToZ, KeepSourceSref)
ArcPy	arcpy.SnapPoints(input dataset, reference dataset, output dataset, "SnapTolerance" , "snap option", "snapToZ", "KeepSourceSref")

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Point Intersection

[Running programmatically](#)

Creates a point feature class with the intersection points of two polyline layers or a polyline layer and the boundaries of the polygons from a polygon layer.

Inputs:

- A polyline layer
- A polyline or a polygon layer

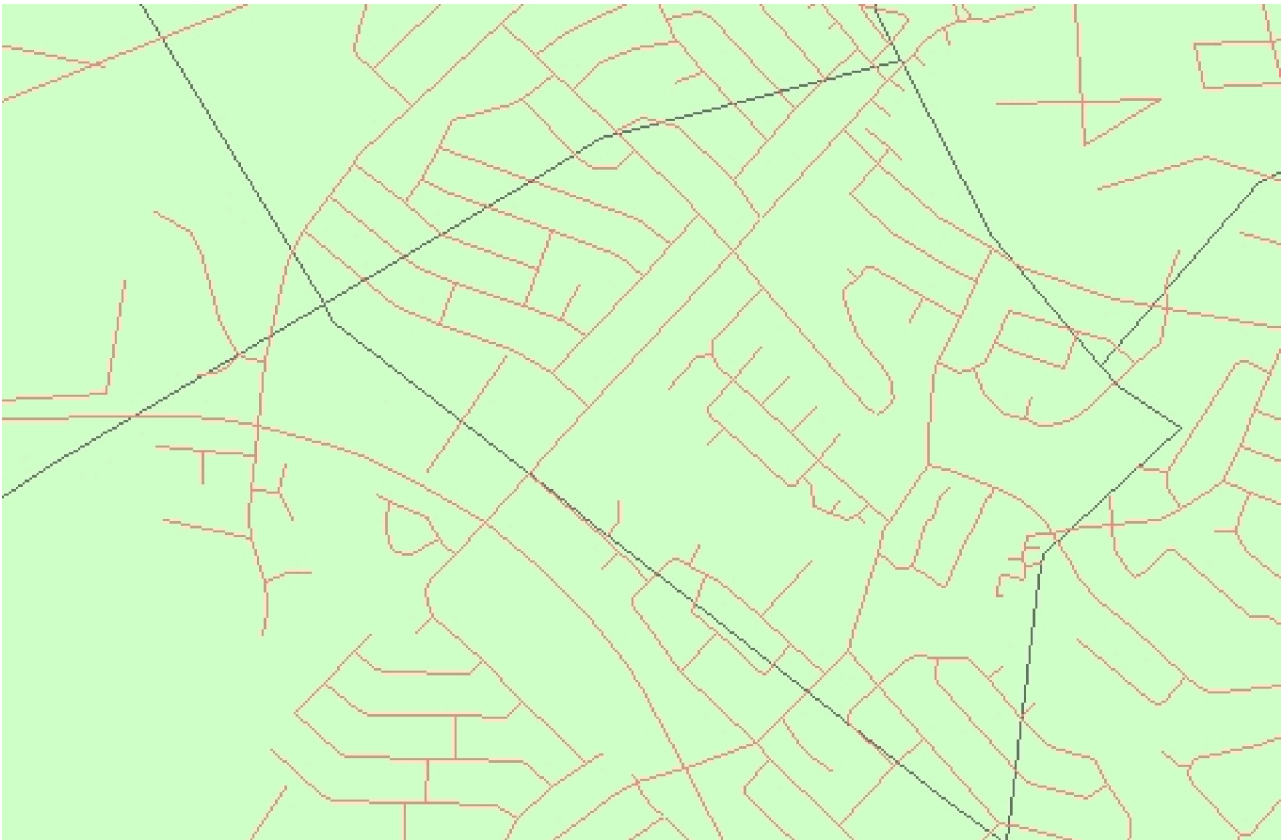
Outputs:

- A point layer - a point in each intersection between polylines from different layers .

Notes:

- The two layers should have spatial references with the same Geographic Coordinate Systems
- If you need a point intersection of the boundaries of two polygon layers, convert one of the layers to polyline first.

Sources: A polygon and a polyline datasets



Resulting points



[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointIntersection
<input dataset>	A String representing the input layer. Must be of Polyline type.
<reference dataset>	A String representing the reference layer. Must be of Polyline or Polygon type
<output dataset>	A String - the full name of the output layer.
{TransferSourceAtt}	A Boolean indicating whether the attributes of the source layer to be transferred to the resulting points.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointIntersection", "input dataset", "reference dataset", "output dataset", "TransferSourceAtt"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointIntersection" "input dataset" "reference dataset" "output dataset" "TransferSourceAtt"</code>
.NET using ETGWOuX.dll	<code>PointIntersection(input dataset,reference dataset, output dataset, TransferSourceAtt)</code>
ArcPy	<code>arcpy.PointIntersection(input dataset, reference dataset, output dataset, "TransferSourceAtt")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Rectangles

[Running programmatically](#)

Creates rectangles from points in a point dataset and user defined width, height rotation angle and location of the point.

Inputs:

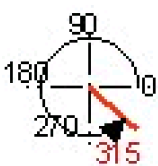
- A Point feature class
- Rectangle Width. The width can be fixed for all points or different assigned from the values in a numeric field of the point attribute table
- Rectangle Height. The height can be fixed for all points or different assigned from the values in a numeric field of the point attribute table
- Rotation angle. The angle can be fixed for all points or different assigned from the values in a numeric field of the point attribute table
- Point location. This parameter defines what will be the location of the resulting rectangles in relation to the original points.

Outputs:

- New polygon feature class. All the original field values will be transferred from the points to the polygons.

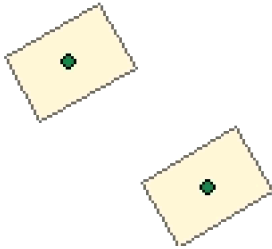
Notes:

- The values for the Width and the Height should be in the units of the spatial reference of the input Point dataset
- The angle (if used) should be in Decimal Degrees and have Polar orientation - East = 0, anti-clockwise

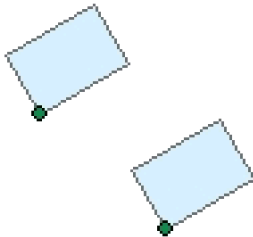


Examples:

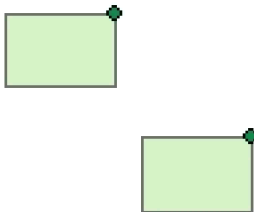
Width = 300, Height = 200, Angle = 15, Location = "Center"



Width = 300, Height = 200, Angle = 15, Location = "LL" (Lower Left)



Width = 300, Height = 200, Angle = 0, Location = "UR" (Upper Right)



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToRectangles
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<Location>	Required. A String -This parameter defines what will be the location of the resulting rectangles in relation to the original points: <ul style="list-style-type: none">"LL" - Lower Left corner of the rectangles will be located on the input point."LR" - Lower Right corner of the rectangles will be located on the

	<p>input point.</p> <ul style="list-style-type: none"> • "UL" - Upper Left corner of the rectangles will be located on the input point. • "UR" - Upper Right corner of the rectangles will be located on the input point. • Any other string used will cause the centers of the rectangles to be located on the input point.
{Width}	A Double representing the rectangle width
{WidthField}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the size of the polygons to be created.
{Height}	A Double representing the rectangle height
{HeightField}	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the height of the rectangles to be created.
{RotationAngle}	A Double representing the rotation angle.
{AngleField}	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the rotation angle of the rectangles to be created. The angle should be in Decimal Degrees and have Polar orientation - East = 0, anti-clockwise.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PointsToRectangles", "input dataset", "output dataset", "Location", "Width", "", "Height", "", "RotationAngle"])

.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToRectangles" "input dataset" "output dataset" "Location" "Width" "" "Height" "" "RotationAngle"</pre>
.NET using ETGWOuX.dll	<pre>PointsToRectangles(input dataset, output dataset, Location, Width, "", Height, "", RotationAngle)</pre>
ArcPy	<pre>arcpy.PointsToRectangles(input dataset, output dataset, "Location" , "Width", "", "Height", "", "RotationAngle")</pre>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Reverse Geocoding

[Running programmatically](#)

Uses a reference polyline (street centerlines) layer to assign addresses to the points from a point layer. Allows transfer of any additional attributes.

Inputs:

- A point layer which features are to be assigned addresses
- A reference polyline layer that will be used as a source for the addresses
- The type of address data
 - Single
 - Range Continuous - From and To address fields expected
 - Range Address - Two pairs (Left & Right) address fields expected. The side of the points taken into account.
- Search distance
- Additional fields to be transferred to the points
- Keep Input Spatial Reference - If selected the output will have the spatial reference of the input dataset, else the spatial reference of the reference dataset will be used

Outputs:

- A point feature class with addresses assigned to the points

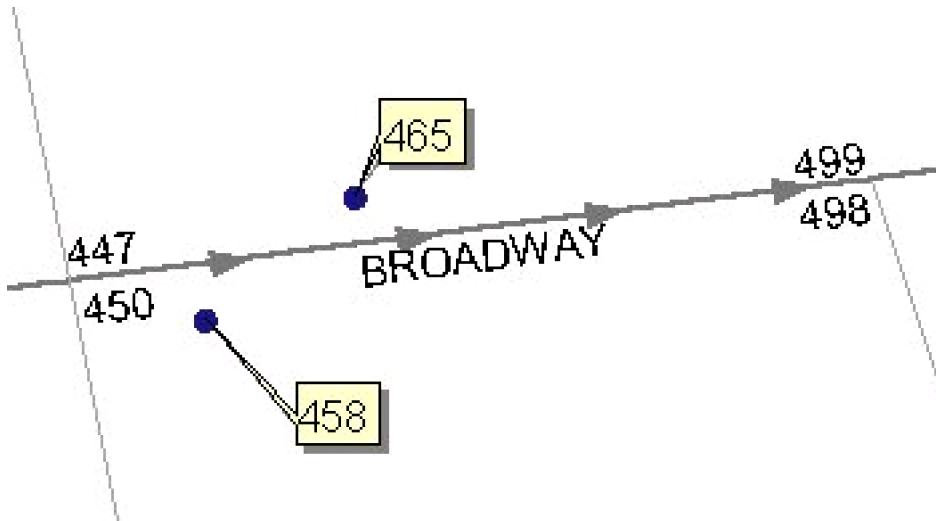
Notes:

- Range attributes fields should be numeric fields.
- The units of the Search Tolerance should be the units of spatial reference of the input dataset if the user has selected the Keep Source Spatial Reference option. Otherwise - the units of spatial reference of the reference dataset.

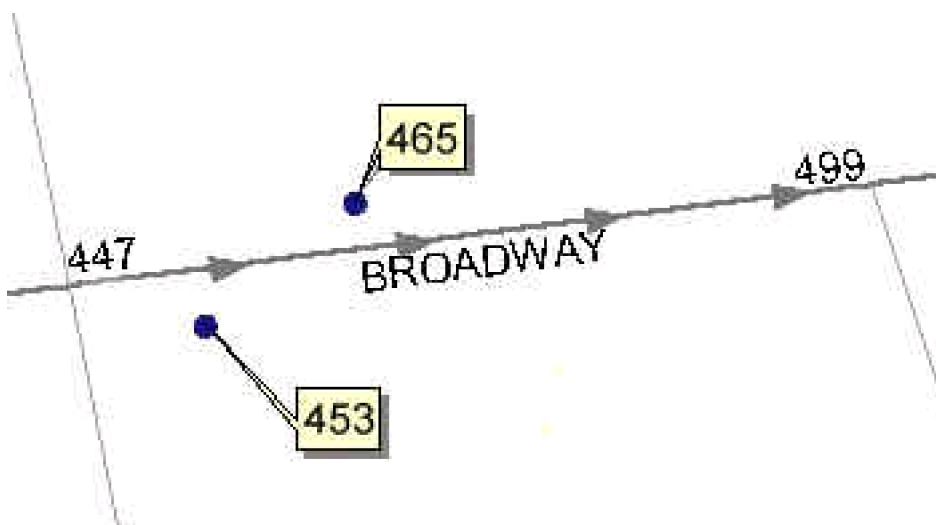
- The Source and the Snap datasets can have different spatial references as long as they have the same Geographic Coordinate systems.

Example:

Address range



Single range



[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ReverseGeocoding
<input dataset>	A String representing the input layer. Must be of Point type.
<reference dataset>	A String representing the reference layer. Must be of Polyline type

<output dataset>	A String - the full name of the output layer.
<search tolerance>	A Double representing the Search tolerance to be used. The units of the tolerance are the units of spatial reference of the input dataset if KeepSourceSref = TRUE. Otherwise - the units of spatial reference of the reference dataset.
<Type>	A String indicating the type of address data - valid values - Single, Continuous, Address
{LeftFromField}	A String representing a field name in the Reference dataset source for the start address on the left side of the road. If the type is Continuous - the start address
{LeftToField}	A String representing a field name in the Reference dataset source for the end address on the left side of the road. If the type is Continuous - the end address
{RightFromField}	A String representing a field name in the Reference dataset source for the start address on the right side of the road. If the type is Continuous not used.
{RightToField}	A String representing a field name in the Reference dataset source for the end address on the right side of the road. If the type is Continuous not used.
{AdditionalFields}	A String representing a list of additional field names which values will be transferred to the points - separator - semi-colon (;) - Example: "field1; field2;"
{KeepSourceSref}	A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPPath, "ReverseGeocoding", "input dataset", "reference dataset", "output dataset", "search tolerance", "Type",

	"LeftFromField", "LeftToField", "RightFromField", "RightToField", "AdditionalFields", "KeepSourceSref"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ReverseGeocoding" "input dataset" "reference dataset" "output dataset" "search tolerance" Type "LeftFromField" "LeftToField" "RightFromField" "RightToField" "AdditionalFields" "KeepSourceSref"
.NET using ETGWOuX.dll	ReverseGeocoding(input dataset,reference dataset, output dataset, search tolerance, Type, LeftFromField, LeftToField, RightFromField, RightToField, AdditionalFields, KeepSourceSref)
ArcPy	arcpy.ReverseGeocoding(input dataset, reference dataset, output dataset, "search tolerance", "Type", "LeftFromField", "LeftToField", "RightFromField", "RightToField", "AdditionalFields", "KeepSourceSref")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Station Points

[Running programmatically](#)

Creates equally spaced points based on the source polyline layer and the user specified distance between the points.

Inputs:

- A polyline feature layer
- Distance between stations
- Output spatial reference

Outputs:

- New Point layer with points distributed along the input polylines based on the user specified distance between the stations
- The attributes of the original polylines are preserved
- The following fields are added to the point attribute table
 - [ET_ID] - the FID of original polylines.
 - [ET_IDP] - this is a unique number identifying each part of the polylines. If a polyline with FID = 356 has 3 parts, the corresponding points will have values in this fields 356_0, 356_1 and 356_2.
 - [ET_X] - the X coordinates of the resulting points
 - [ET_Y] - the Y coordinates of the resulting points
 - [ET_Angle] - the angle of the polyline in this point.
 - [ET_Station] - the distance from the start point of the polyline to this point

Notes:

- The distance is measured in the units of the output spatial reference
- The default output spatial reference is the one of the input polyline dataset
- The user can specify a different output spatial reference, but it has to have the same Geographic Coordinate System as the one of the input dataset

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	StationPoints
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Station Distance>	A Double representing distance between the station lines.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "StationPoints", "input dataset", "output dataset", "Station Distance"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "StationPoints" "input dataset" "output dataset" "Station Distance"
.NET using ETGWOuX.dll	StationPoints(input dataset, output dataset, Station Distance)

ArcPy

```
arcpy.StationPoints(input dataset, output dataset , "Station Distance")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Thin Points

[Running programmatically](#)

Reduces the number of points in a point dataset based on their spatial location.

- Points that are closer to each other than the tolerance specified will be converted to a single point.
- The values in the field specified will be summarized for each cluster and saved in the point representing this cluster.
- Optionally a copy of the original layer will be created. The attribute table will have a field enabling a back link from the generalized points to the original ones to be created.

Inputs:

- A Point feature class
- Generalization tolerance - should be smaller than 20% of the smaller side of the extent envelope of the input dataset.
- Data field - if specified the output feature class will have a field in which the value for each resulting point will be the sum of the values of the points pertaining to this cluster.

Outputs:

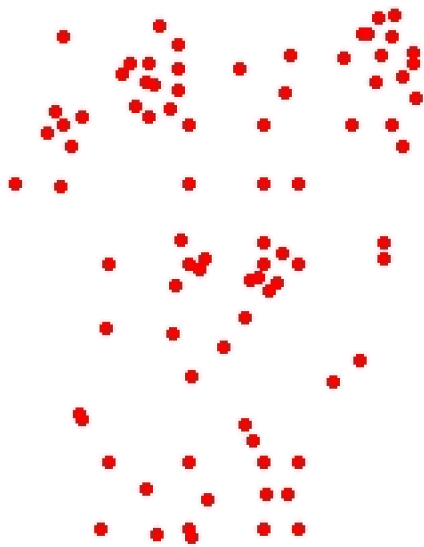
- New point layer. Fields:
 - [ET_Count] - the number of points from the input feature class represented by each output point
 - [ET_Link] - added only if the "Add Link from the original points" option is selected.
- Optional point layer - a copy of the original points with a link field added.

Notes:

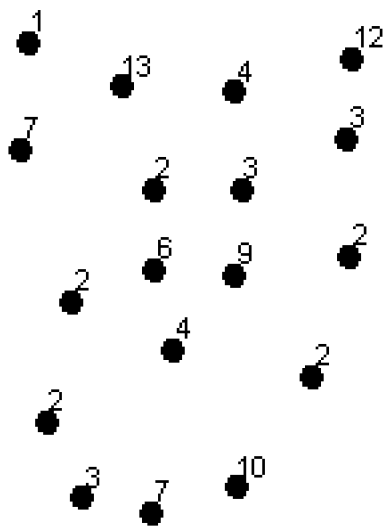
- If no Generalization tolerance of 0 is specified only the exact duplicates will be removed

Examples:

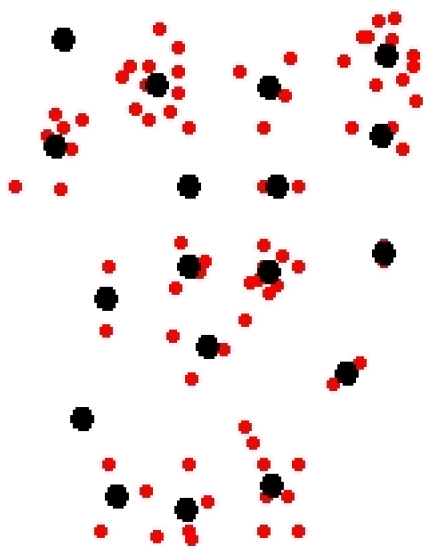
Input Dataset



Result Dataset labeled with the number of points they represent (ET_Count field)



Overlay



[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ThinPoints
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<tolerance>	A Double representing the Generalization tolerance (in the units of the spatial reference of the input dataset) to be used
{LinkName}	A String representing the full name of the optional output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ThinPoints", "input dataset", "output dataset", "tolerance", "LinkName"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ThinPoints" "input dataset" "output dataset" "tolerance" "LinkName"</code>
.NET using ETGWOuX.dll	<code>ThinPoints(input dataset, output dataset, tolerance, LinkName)</code>
ArcPy	<code>arcpy.ThinPoints(input dataset, output dataset, "tolerance" , "LinkName")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Regular Polygons

[Running programmatically](#)

Creates regular polygons from points in a point dataset and user defined number of sides, size and rotation angle. The source point will be located in the center of the polygons.

Inputs:

- A Point feature class
- Number of sides of the polygons to be created.
- Size represents option - depending on the user input the size parameter can represent
 - The side of the polygon
 - The radius of the circle inscribed in the polygon
 - The radius of the circle circumscribed around the polygon.
- Size of the polygon. The size can be fixed for all points or different assigned from the values in a numeric field of the point attribute table
- Rotation angle. The rotation angle can be fixed for all points or different assigned from the values in a numeric field of the point attribute table

Outputs:

- New polygon feature class. All the original field values will be transferred from the points to the polygons.

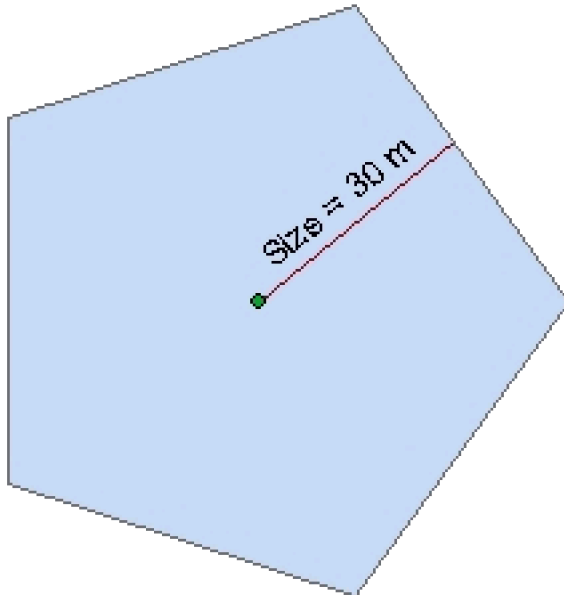
Notes:

- The values for the size should be in the units of the spatial reference of the input Point dataset
- The angle (if used) should be in Decimal Degrees and have Polar orientation - East = 0, anti-clockwise. The angle defines the location of the start vertex of the polygon.

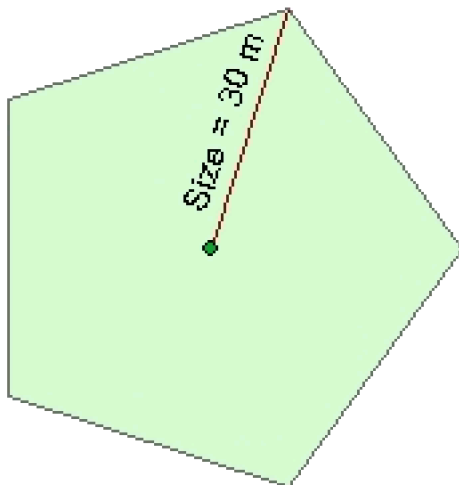


Examples:

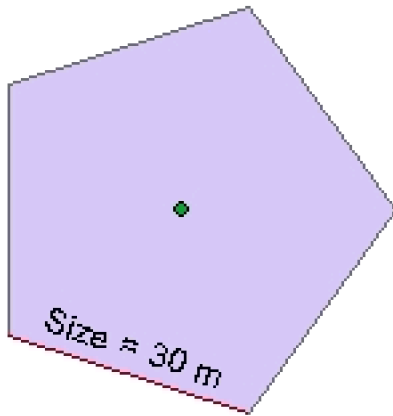
Size = 30 , Angle = 0, Number sides = 5, Option - Radius Inscribed



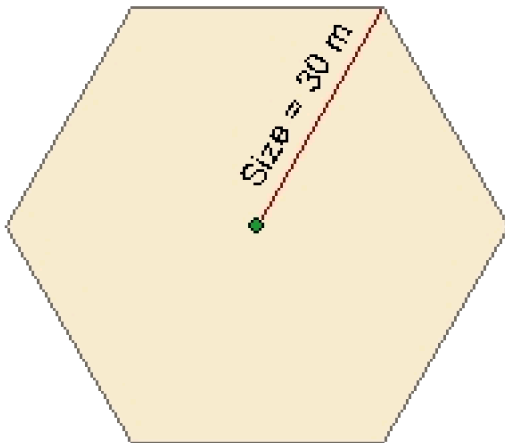
Size = 30 , Angle = 0, Number sides = 5, Option - Radius Circumscribed



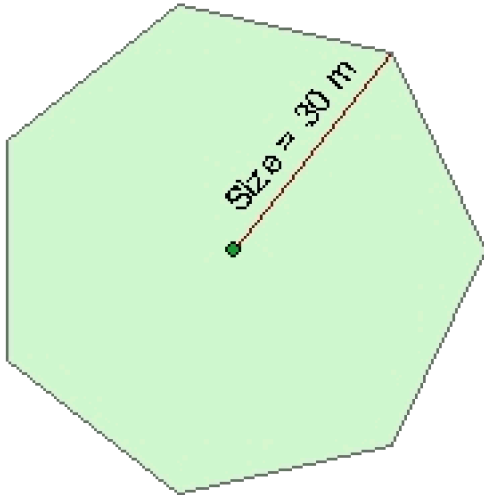
Size = 30 , Angle = 0, Number sides = 5, Option - Side



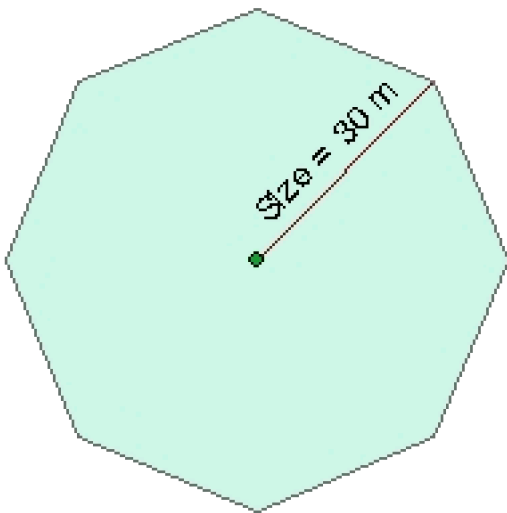
Size = 30 , Angle = 0, Number sides = 6, Option - Radius Circumscribed



Size = 30 , Angle = 0, Number sides = 7, Option - Radius Circumscribed



Size = 30 , Angle = 0, Number sides = 8, Option - Radius Circumscribed



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToRegularPolygons
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.

<NumberSides>	A Double representing the Generalization tolerance (in the units of the spatial reference of the input dataset) to be used
<SizeRepresents>	Required. A String indicating the meaning of the <SIZE> parameter <ul style="list-style-type: none"> • RadiusIn - radius of the inscribed circle • RadiusOut - radius of the circumscribed circle • Side - the side of the polygon
{Size}	A Double representing the size (see above for options)
{SizeField}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the size of the polygons to be created.
{RotationAngle}	A Double representing the rotation angle.
{AngleField}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the rotation angle of the polygons to be created.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToRegularPolygons", "input dataset", "output dataset", "NumberSides", "SizeRepresents", "Size", "", "RotationAngle"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToRegularPolygons" "input dataset" "output dataset" "NumberSides" "SizeRepresents" "Size" "" "RotationAngle"</pre>
.NET using ETGWOuX.dll	<code>PointsToRegularPolygons(input dataset, output dataset, NumberSides, SizeRepresents, Size, "", RotationAngle)</code>

ArcPy

```
arcpy.PointsToRegularPolygons(input dataset, output dataset,  
"NumberSides" , "SizeRepresents", "Size", "", "RotationAngle")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Polylines

[Running programmatically](#)

Ensures topological correctness of a polyline feature data set.

Inputs:

- A polyline feature layer
- Fuzzy tolerance

Outputs:

- New topologically correct Polyline dataset
 - Nodes will be created in all intersection points
 - Redundant data (vertices and nodes closer to each other than the fuzzy tolerance) will be eliminated.
 - Each set of duplicate lines (closer to each other than the fuzzy tolerance) will be replaced by a single polyline. This polyline will carry the attributes of one of the original polylines
 - The attributes of the input data set are preserved.
- Optional Polyline layer that identifies the duplicates in the input data set.
 - The duplicates layer has all the fields from the input data set.
 - The attributes of the polylines are these that have not been preserved in the clean layer. Example:. If two polylines with attributes "A" and "B" are running on top of each other. The clean layer will contain only one of them e.g. "A". The duplicates layer will contain the other one -"B"

Notes :

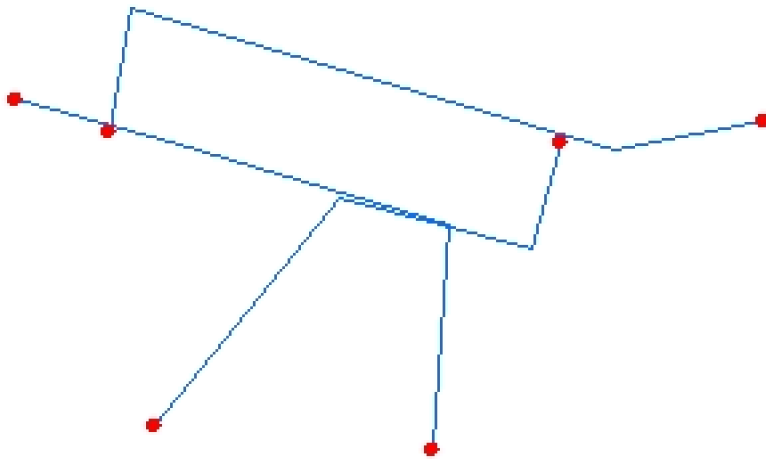
- The default Fuzzy tolerance is calculated from the extents of the input layer using the

formulae $(W + H) / 2000000$ where W and H are the with and height of the extent envelope.

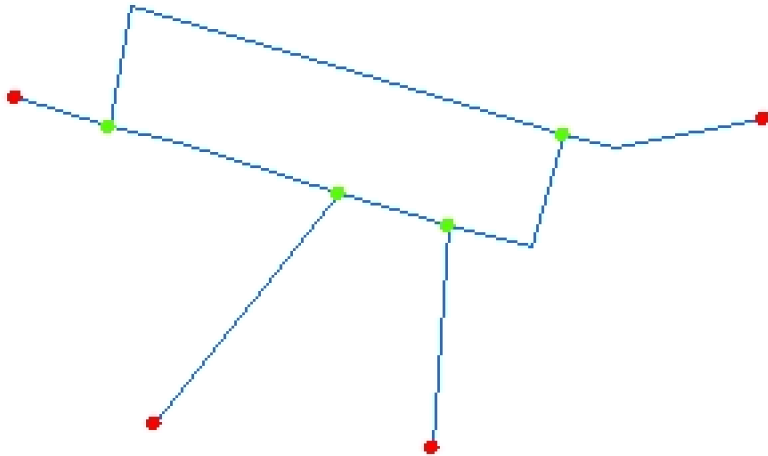
- Larger values of the Fuzzy tolerance may be used to clean some bigger Over and Under shoots, but it might lead to unwanted approximation of the input shapes. The better option is to use Fuzzy tolerance close to the default and then clean the remaining Dangling Nodes with the "Clean Dangling Nodes Wizard"
- Use Export Nodes Wizard to check the status of a data set. It will analyze the nodes of a polyline layer and will create a Point layer with classified nodes.

Example:

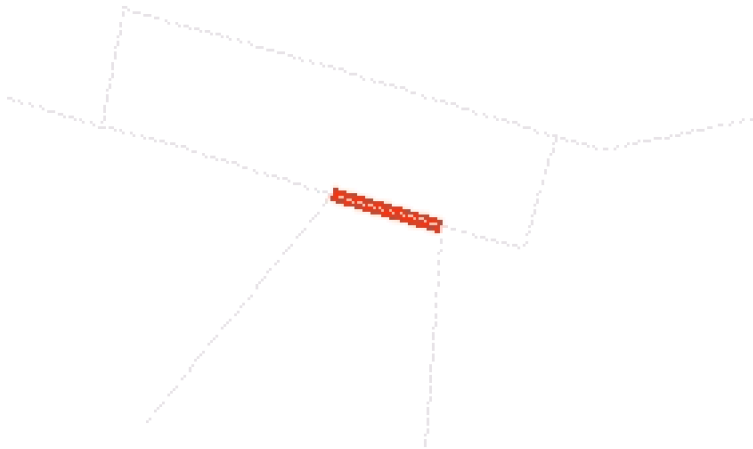
Input Layer (with analyzed nodes)



After Clean (with analyzed nodes)



Duplicates layer (The Clean layer as background)



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanPolylines

<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Fuzzy Tolerance>	A Double representing the Fuzzy Tolerance.
<duplicates dataset>	A String - the full name of the output layer containing the duplicates.
{Sort Field}	A String representing a field to be used to sort the data before processing. The first features in the input dataset after sorting using that field will have priority during the cleaning process.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CleanPolylines", "input dataset", "output dataset", "Fuzzy Tolerance", "duplicates dataset", "Sort Field"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanPolylines" "input dataset" "output dataset" "Fuzzy Tolerance" "duplicates dataset" "Sort Field"</code>
.NET using ETGWOuX.dll	<code>CleanPolylines(input dataset, output dataset, Fuzzy Tolerance, duplicates dataset, Sort Field)</code>
ArcPy	<code>arcpy.CleanPolylines(input dataset, output dataset, "Fuzzy Tolerance", "duplicates dataset", "Sort Field")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Dangling Nodes

[Running programmatically](#)

Cleans the dangling nodes from a polyline layer using user specified dangling tolerance

Inputs:

- A polyline feature layer
- Dangling tolerance

Outputs:

- New polyline feature class
 - Over-shoots: All the Polylines having a Dangling node with length less than the dangling tolerance will be deleted
 - Under-shoots: All the Polylines having a Dangling node and length larger than the tolerance will be processed.
 - The function first checks whether the extension of the polyline in the direction of the segment containing the dangling node intersects existing polyline (within the specified tolerance) and if so extends the segment to the intersection point
 - If the "Snap to nearest feature" option is selected the function checks whether the dangling node is closer than the tolerance specified to another feature and snaps to the closest feature
 - If the dangling feature is snapped to another feature, the latest is split - a regular node created.
 - The attributes of the input data set are preserved.

Notes :

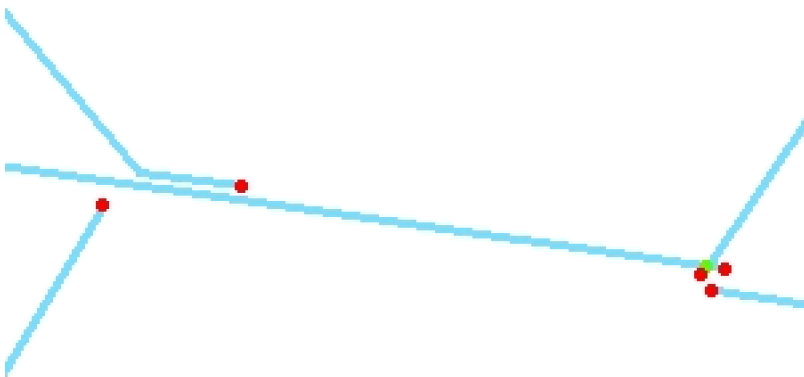
- Use [Export Nodes Wizard](#) to check the status of a data set. It will analyze the nodes of a

polyline layer and will create a Point feature class with classified nodes.

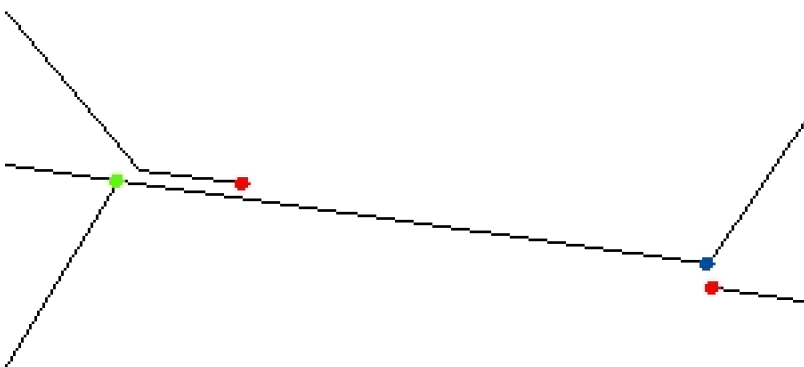
- It is recommended before proceeding with cleaning the dangling nodes to use the [Clean Polyline Wizard](#) to ensure that all the polylines are intersected and the very small over and undershoots are cleaned with the Fuzzy tolerance.
- A Fuzzy tolerance of 0 may be used if the original shapes have to be preserved exactly the same. In this case only the intersections will be created

Examples:

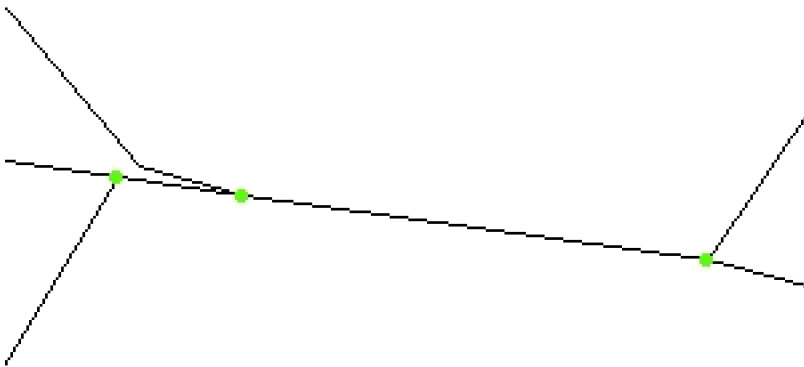
Before Clean Dangles



After Clean - No "Nearest Snap" used



After Clean - "Nearest Snap" used



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanDangles
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Dangling Tolerance>	A Double representing the tolerance to be used. The polylines with dangling nodes within distance smaller than this value to existing features (depending on the options used) will be extended/snapped to the closest existing feature
{Snap Nearest}	A Boolean - if True, the ends of the dangling polylines that could not be snapped to an existing polyline by extending in the direction of the last segment or to an existing node will be snapped (if the dangling node is within a distance smaller than the to an existing polyline) to the closest point of the closest polyline
{Fuzzy Tolerance}	A Double representing the Fuzzy tolerance (in the units of the input dataset).

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CleanDangles", "input dataset", "output dataset", "Dangling Tolerance", "Snap Nearest", "Fuzzy Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanDangles" "input dataset" "output dataset" "Dangling Tolerance" "Snap Nearest" "Fuzzy Tolerance"</code>
.NET using ETGWOuX.dll	<code>CleanDangles(input dataset, output dataset, Dangling Tolerance, Snap Nearest, Fuzzy Tolerance)</code>
ArcPy	<code>arcpy.CleanDangles(input dataset, output dataset, "Dangling Tolerance", "Snap Nearest", "Fuzzy Tolerance")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Pseudo Nodes

[Running programmatically](#)

Combines polylines, which share a pseudo node, based on user specified attributes. The resulting polyline data set does not contain multi-part polylines. The topology of the data set is preserved. The attribute update rules include range values update

Inputs

- A polyline feature layer
- Fields to be used for dissolving
- Update rules for the rest of the fields to be transferred

Outputs:

- An aggregated polyline feature class
- Only the polylines which share a pseudo node that have the same values for the dissolve fields will be aggregated
- No multi-part polylines will be created.
- The attributes will be transferred according the user specified rules. For the fields with no specified update rule, date and blob fields, the aggregated feature will carry the attributes of the first feature.

Notes:

- The Clean Pseudo Nodes function is similar to Dissolve but has several advantages:
 - Multiple dissolve fields can be used
 - Only the polylines which share a pseudo node that have the same values for the dissolve fields will be aggregated
 - It preserves the topology of the polyline layer - no regular node will be removed as a result of the procedure. See example

- The attribute update rules include rules to handle single and address ranges. See example

- It is recommended the Explode function to be used before dissolve in order to ensure proper distribution of the attribute values of the numeric fields.
- If no dissolve field is selected, all the pseudo nodes will be removed without considering the attribute values.
- Range attributes from numeric fields can be handled. The records that have range values containing non numeric characters will be copied arbitrary to the resulting features.

Example:

Input Layer. Dissolve field = "Streetname"



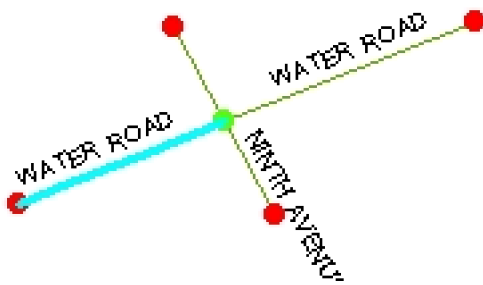
Attribute Table

STREETNAM	L_F_ADD	L_T_ADD	R_F_ADD	R_T_ADD
WATER ROAD	90	98	83	91
NINTH AVENUE	46	50	45	59
NINTH AVENUE	34	38	35	37
WATER ROAD	124	128	107	113
WATER ROAD	110	118	93	101
WATER ROAD	118	124	101	107

After Clean Pseudo Nodes. Update rules

"L_F_ADD" - Address Range - Paired field - "L_T_ADD"

"R_F_ADD" - Address Range - Paired field "R_T_ADD"



STREETNAM	L_F_ADD	L_T_ADD	R_F_ADD	R_T_ADD
WATER ROAD	90	98	83	91
NINTH AVENUE	46	50	45	59
NINTH AVENUE	34	38	35	37
WATER ROAD	110	128	93	113

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanPseudoNodes
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Dissolve Fields}	A String representing a list (separated with ";") with the field names to be used for dissolving.
{Update Rules}	A String representing a list (separated with ";") with the field names and update rules. Valid rules are: "FIRST", "LAST", "MIN", "MAXLENGTH", "MAX", "SUM", "RANGECONTINUOUS", "RANGEADDRESS" depending on the field type. Example: "Field1 RangeAddress Field2;Field3 First;Field3 MaxLength"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "CleanPseudoNodes", "input dataset", "output dataset", "Dissolve Fields", "Update Rules"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "CleanPseudoNodes" "input dataset" "output dataset" "Dissolve Fields" "Update Rules"</pre>
.NET using ETGWOutX.dll	<code>CleanPseudoNodes(input dataset, output dataset, Dissolve Fields, Update Rules)</code>
ArcPy	<code>arcpy.CleanPseudoNodes(input dataset, output dataset, "Dissolve Fields", "Update Rules")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Split Polylines

[Running programmatically](#)

Splits the features of a polyline layer.

Inputs

- A polyline layer which features are to be split
- A method to be used for splitting
 - In all vertices
 - Equal segments
 - Exact length + remainder - the polylines will be split using the exact length specified by the user. The last segment will have length equal to the remainder of the splitting. For example if a polyline with length = 60 meters is split using length of 25 meters, three segments with lengths 25, 25 and 10 will be created.
 - Equal Length - the user specified length is adjusted in order all resulting segments to have equal length. For example if a polyline with length = 60 meters is split using length of 25 meters, two segments with lengths of 30 meters will be created.
 - Number of vertices per feature
- Attribute Cell SizeY for each field

Outputs

- A polyline feature class - new node created in each split point. The attributes will be distributed according the user specified attribute Cell SizeY.

Notes:

- Range attributes from numeric fields can be handled.
- If the Segment length method is used

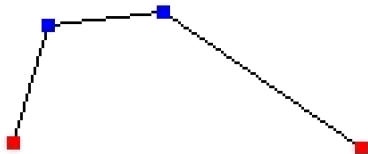
- all the segments will have the user specified length except for the last one
- if the assigned length is larger than the length of a specific polyline, the polyline will be copied to the output as is
- the splitting starts always from the start points (From Node) of the polylines

Examples:

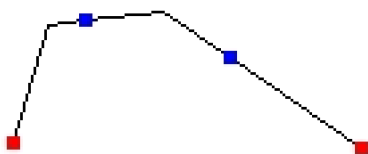
Before Split



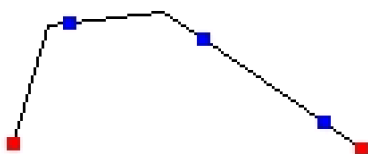
Split in all Vertices



Split in equal segments



Exact length + remainder



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SplitPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Split Option>	<p>Required. A String valid values:</p> <ul style="list-style-type: none">• "Vertex"• "NumberIntervals"• "EqualLength"• "Length"• "NumberVertices"
{Split Tolerance}	<p>A Number which value depending on the option defines:</p> <ul style="list-style-type: none">• "Vertex" - not used• "NumberIntervals" - the number of intervals per polyline• "EqualLength" - the desired length of the output polylines• "Length" - the length of the output polylines• "NumberVertices" - the number of vertices per polyline.
{Update Rules}	A String representing a list (separator ";") of fields and update rule for each field. Example: "Field1 AddressRange Field2; Field3 Copy; Field4 Proportion"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SplitPolylines", "input dataset", "output dataset", "Split Option", "Split Tolerance", "Update Rules"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SplitPolylines" "input dataset" "output dataset" "Split Option" "Split Tolerance" "Update Rules"</code>
.NET using ETGWOuX.dll	<code>SplitPolylines(input dataset, output dataset, Split Option, Split Tolerance", Update Rules)</code>
ArcPy	<code>arcpy.SplitPolylines(input dataset, output dataset, "Split Option" , "Split Tolerance", "Update Rules")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Split Polyline Layer

[Running programmatically](#)

Splits a polyline layer with the features of a Point, Polyline or Polygon layer.

Inputs

- A polyline layer which features are to be split
- A point, polyline or polygon layer which features will be used for splitting
- Attribute update rules for each field
- Search distance (Only if a point layer is used as a split layer)

Outputs

- A polyline feature class - a node created in each intersection point between the features from the input polyline layer and the split layer.

Notes:

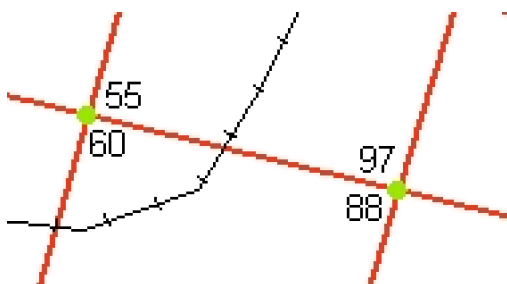
- Range attributes from string and numeric fields can be handled. The records that have range values containing non numeric characters will be copied to the resulting features.
- If a point layer is used as a split layer only the points that are within the search tolerance from the features of the input polyline layer will be used for splitting.

Examples:

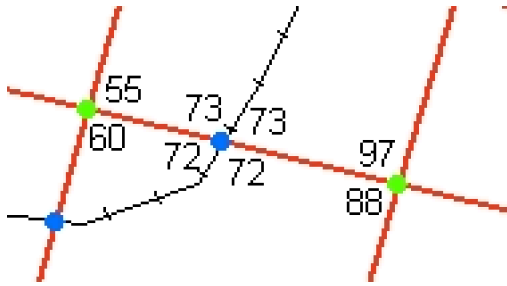
Polyline Layer with Polyline Layer.

Attributes updated with Range Address split rule

Before Split



After Split

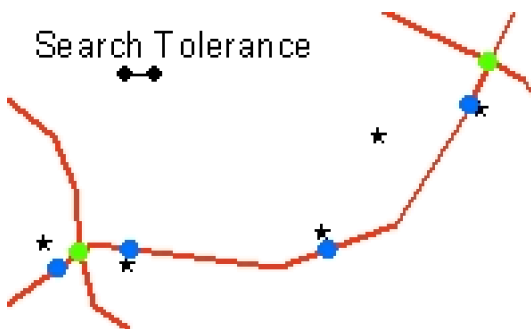


Polyline Layer with Point Layer. Only the points within the search tolerance from the polylines are used.

Before Split



After Split



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SplitPolylinesWithLayer
<input dataset>	A String representing the input layer. Must be of Polyline Type.

<Reference Dataset>	A String representing the reference layer.
<output dataset>	A String - the full name of the output layer.
{Search Tolerance}	A Double representing the Cut-Off Distance to be used. The units of the tolerance are the units of spatial reference. Used only if the Reference Layer is of Point type.
{Update Rules}	A String representing a list (separator ";") of fields and update rule for each field. Example: "Field1 AddressRange Field2; Field3 Copy; Field4 Proportion"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SplitPolylinesWithLayer", "input dataset", "Reference Dataset", "output dataset", " Search Tolerance", "Update Rules"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SplitPolylinesWithLayer" "input dataset" "Reference Dataset" "output dataset" "Search Tolerance" "Update Rules"</pre>
.NET using ETGWOutX.dll	<code>SplitPolylinesWithLayer(input dataset,Reference Dataset, output dataset, Search Tolerance, Update Rules)</code>
ArcPy	<code>arcpy.SplitPolylinesWithLayer(input dataset, Reference Dataset, output dataset, "Search Tolerance" , "Update Rules")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)

- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polyline Global Snap

[Running programmatically](#)

Snaps the features of a polyline layer to another layer (Point, Multipoint, Polyline or Polygon)

Inputs

- A polyline layer to be snapped
- A snap layer - point, polyline or polygon
- Snap tolerance
- Snap options1 (Snap What)
- Snap options2 (Snap To What)
- Snap to reference Z values (only if the input and output are Z enabled)

Outputs

- A polyline layer - the polylines from the source layer will be moved to snap to the features of the Snap Layer (if within the snap tolerance)

Options:

- Snap Options 1 (Snap What) - this options lets the user set which elements of the source polylines to be used for snapping
 - Nodes: Only nodes (end points) of the polylines will be snapped
 - Vertices: All the vertices of the source polylines will be used.
 - Insert Vertices: This option will get the vertices from the features of the snap layer and will insert new vertices into the source polylines. The new vertices together with the original ones will be used for snapping. This option is slower than the other ones, but gives the best snapping results especially if the polylines to be snapped have much less vertices than the ones from the Snap layer.

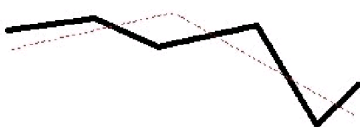
- Snap Options 2 (Snap To What)
 - Vertices: The polylines will be snapped to the nearest vertex of the nearest feature from the Snap layer
 - Nearest edge: The polylines will be snapped to the nearest point of the nearest feature from the Snap layer
 - Vertices and Edges: If there is a vertex closer than the snap tolerance to the polylines (their elements defined in Options 1) to be snapped, the polyline will snap to it, otherwise it will snap to the nearest edge.
- Snap to Z

Notes:

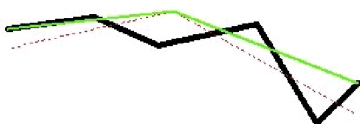
- The snap distance should be in the units of the input dataset.
- The Source and the Snap datasets can have different spatial references as long as they have the same Geographic Coordinate systems.

Example: Red - Source Polyline; Black - Snap Polyline; Green - Snapped Polyline

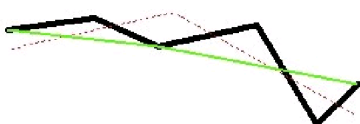
Before Snap



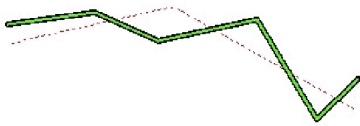
After Snap - Option1: Nodes, Option2: Vertices



After Snap - Option1: Vertices, Option2: Vertices



After Snap - Option1: Insert Vertices, Option2: Vertices and Edges



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SnapPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<Reference Dataset>	A String representing the layer to be used to snap to.
<output dataset>	A String - the full name of the output layer.
<Snap Tolerance>	A Double representing the Snap Tolerance (in the units of the spatial reference of the input dataset or the Reference Dataset is KeepSourceSref = FALSE).
<Snap What>	<p>Required. A String indicating what parts of the input polygons will be snapped. Possible values:</p> <ul style="list-style-type: none">• Node - only the nodes of the source polylines will be snapped.• Vertex - the vertices of the source polylines will be snapped.• InsertVertex - the vertices from the features of the Reference Dataset will be inserted (if closer to the input dataaset) as new vertices into the source polylines. The new vertices together with the original ones will be used for snapping. This option is slower than the other ones, but gives the best snapping results especially if the polygons to be snapped have much less vertices than the ones from the Reference Dataset.

<Snap To What>	<p>Required. A String indicating to what parts of the reference geometries the input polygons will try to snap. Possible values:</p> <ul style="list-style-type: none"> • Vertex - the input polylines will be snapped only to the vertices of the geometries from the reference dataset. • All - the input polylines will be snapped to the vertices or nearest edge of the geometries from the reference dataset.
{Snap To Z}	Optional. A Boolean indicating whether the input geometries will snap the Z values of the geometries from the Reference Dataset. Only if both dataset have Z values.
{KeepSourceSref}	Optional. A Boolean indicating whether the output to have the spatial reference of the input layer. If False or 0, the spatial reference of the reference layer will be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "SnapPolylines", "input dataset", "Reference Dataset", "output dataset", "Snap Tolerance", "Snap What", "Snap To What", "Snap To Z", "KeepSourceSref"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "SnapPolylines" "input dataset" "Reference Dataset" "output dataset" "Snap Tolerance" "Snap What" "Snap To What" "Snap To Z" "KeepSourceSref"
.NET using ETGWOutX.dll	SnapPolylines(input dataset, Reference Dataset, output dataset, Snap Tolerance, Snap What, Snap To What, Snap To Z, KeepSourceSref)
ArcPy	arcpy.SnapPolylines(input dataset, Reference Dataset, "output dataset" , "Snap Tolerance" , "Snap What" , "Snap To What" , "Snap To Z" , "KeepSourceSref")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Buffer Polylines

[Running programmatically](#)

Creates buffer polygons from the polylines of the input dataset. The user can specify the side of the polyline (Left, Right or Both) on which the buffer to be created as well as the shape of the buffer at the end of the polylines - Round or Flat. The buffer distance (in the units of the spatial reference of the input dataset) can be entered as a number (equal for all input polylines) or a numeric field. No negative buffer distance is accepted.

Inputs:

- A polyline feature layer
- Buffer distance - a number (the same buffer distance will be used for all input polylines) or the name of a numeric field in the polyline attribute table that has the buffer distance for each input polyline.
- Side of the buffer:
 - Left - the buffer will be created only on the left side of the polylines (the physical orientation of the polyline is used to define the side).
 - Right - the buffer will be created only on the right side of the polylines (the physical orientation of the polyline is used to define the side).
 - Both - the buffer will be created only on the both sides of the polylines.
- Shape of the buffer at the end of the polyline
 - Round - the buffer at the ends of the polyline will have a circular shape.
 - Flat - the buffer at the end of the polyline will be closed with a straight line passing through the start/end point of the polyline
- Dissolve option - the boundaries of the intersecting buffers will be dissolved. The original attributes will not be preserved if the dissolve option is used.

Outputs:

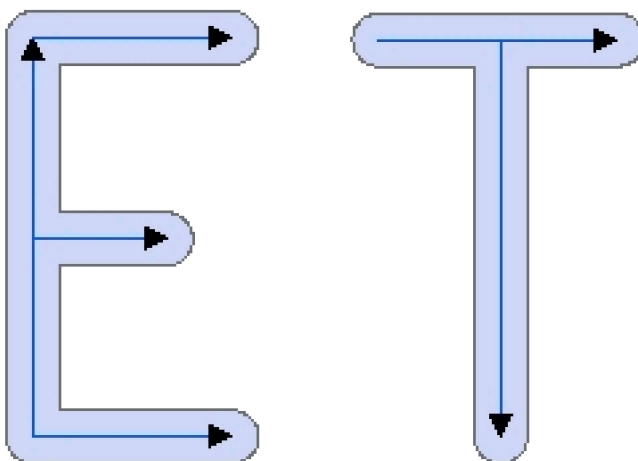
- New polygon feature class

Notes :

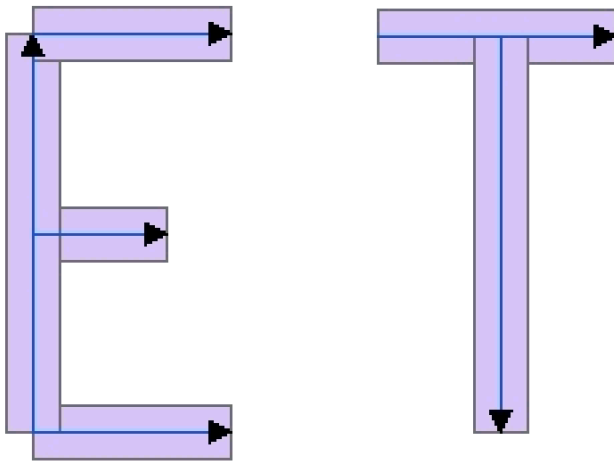
- No buffers will be created for the polylines that have buffer distance less or equal to 0.
- The attributes of the polylines will be transferred to the resulting polygons only if the Dissolve option is NOT used.
- If the "Left" or "Right" option is used, no buffers will be created for the polylines for which the Left and Right buffers intersect. The log file will contain a record for the IDs of the polylines for which the requested buffer could not be created (see example below).
- If the "Flat" option is used and the left and right boundary cannot be connected with a straight line with length equal to 2 x the buffer distance, a round end will be created. The log file will contain a record for the IDs of the polylines for which a "Flat" buffer could not be created (see example below).
- Self-intersecting polylines will be simplified and each part will be buffered separately.

Examples:

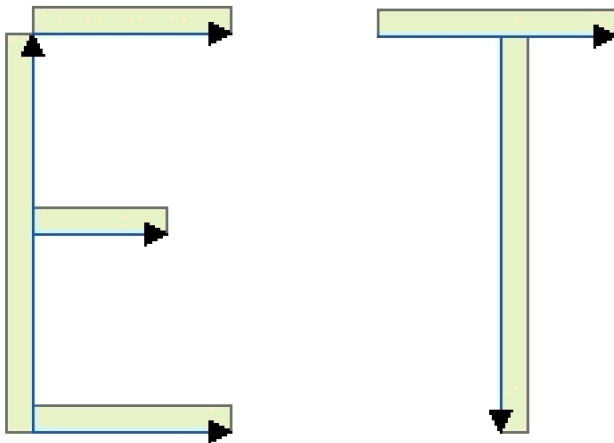
Round buffer both sides (with dissolve option)



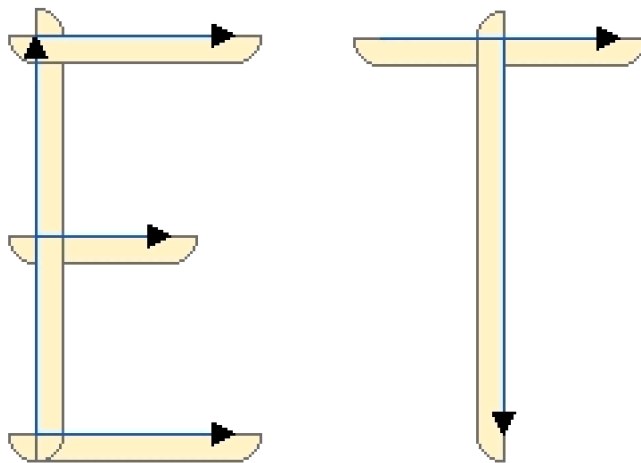
Flat buffer both sides



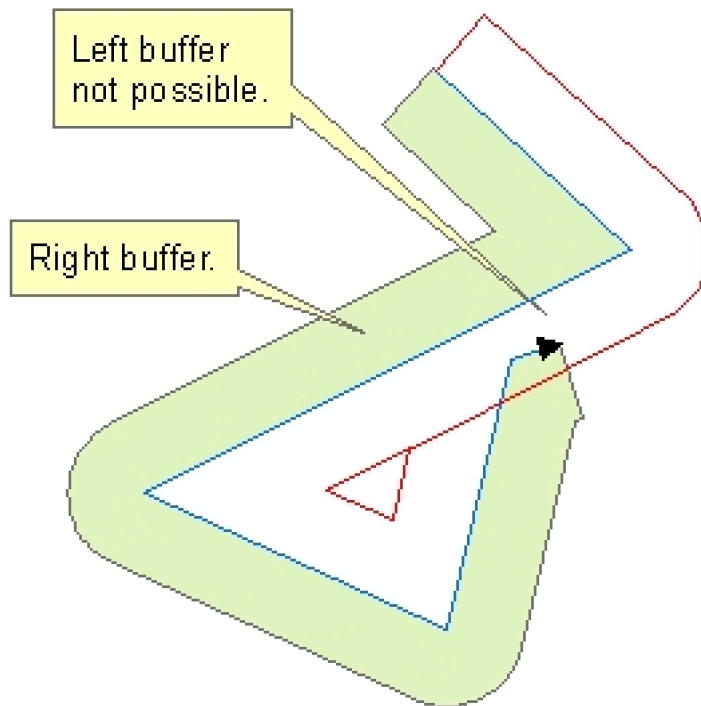
Flat buffer on the left side



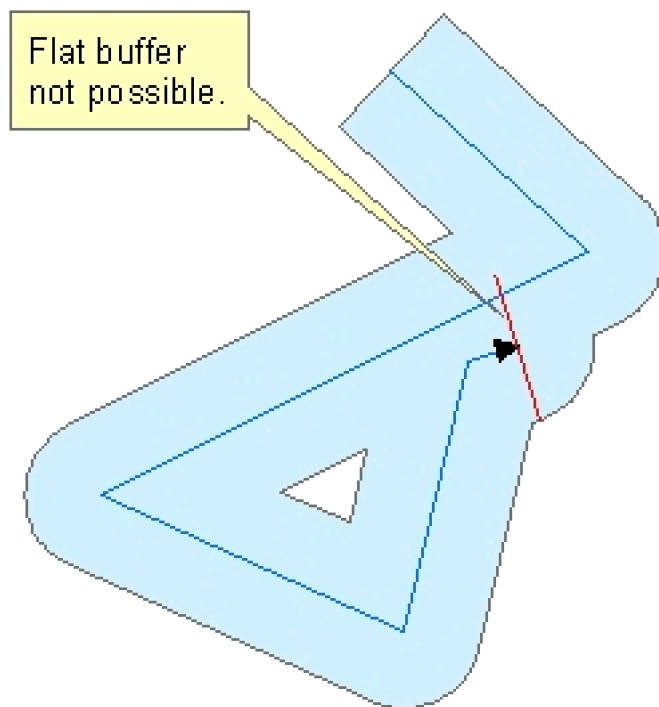
Round buffer on the right side



Buffer on the left side cannot be created.



Round end will be created if a complete flat end cannot be created.



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	BufferPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Buffer Side>	Required. A String representing the side of the polylines on which the buffers will be created. The available options are: Left, Right and Both
<Buffer End>	Required.Required. A String representing the shape of the buffer at the ends of the polylines. The available options are: Round - the buffer at the ends of the polyline will have a circular shape. Flat - the buffer at the end of the polyline will be closed with a straight line passing through the start/end point of the polyline

{Buffer Distance}	A Double representing the buffer distance in the units of the Spatial Reference of the input layer.
{Buffer Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the buffer distance.
{Dissolve Buffers}	A Boolean. If True - the boundaries of the intersecting buffers will be dissolved.
{Dissolve Fields}	A String representing a list (separated with ";") with the field names to be used for dissolving.
{Statistic Fields}	A String representing a list (separated with ";") with the field names for which statistics will be created. Example: "Field1 Sum;Field2 Max;Field3 Min"
{Create Multiparts}	A Boolean indicating whether the function will create multipart polygons.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "BufferPolylines", "input dataset", "output dataset", "Buffer Side", "Buffer End", "Buffer Distance", "", "Dissolve Buffers", "Dissolve Fields", "Statistic Fields", "Create Multiparts"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "BufferPolylines" "input dataset" "output dataset" "Buffer Side" "Buffer End" "Buffer Distance" "" "Dissolve Buffers""Dissolve Fields""Statistic Fields""Create Multiparts"</code>
.NET using ETGWOuX.dll	<code>BufferPolylines(input dataset, output dataset, Buffer Side, Buffer End, "", Buffer Distance, "", Dissolve Buffers,Dissolve Fields,Statistic Fields,Create Multiparts)</code>
ArcPy	<code>arcpy.BufferPolylines(input dataset, output dataset, "Buffer Side", "Buffer End", "Buffer Distance", "", "Dissolve Buffers", "Dissolve Fields", "Statistic Fields", "Create Multiparts")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Export Nodes

[Running programmatically](#)

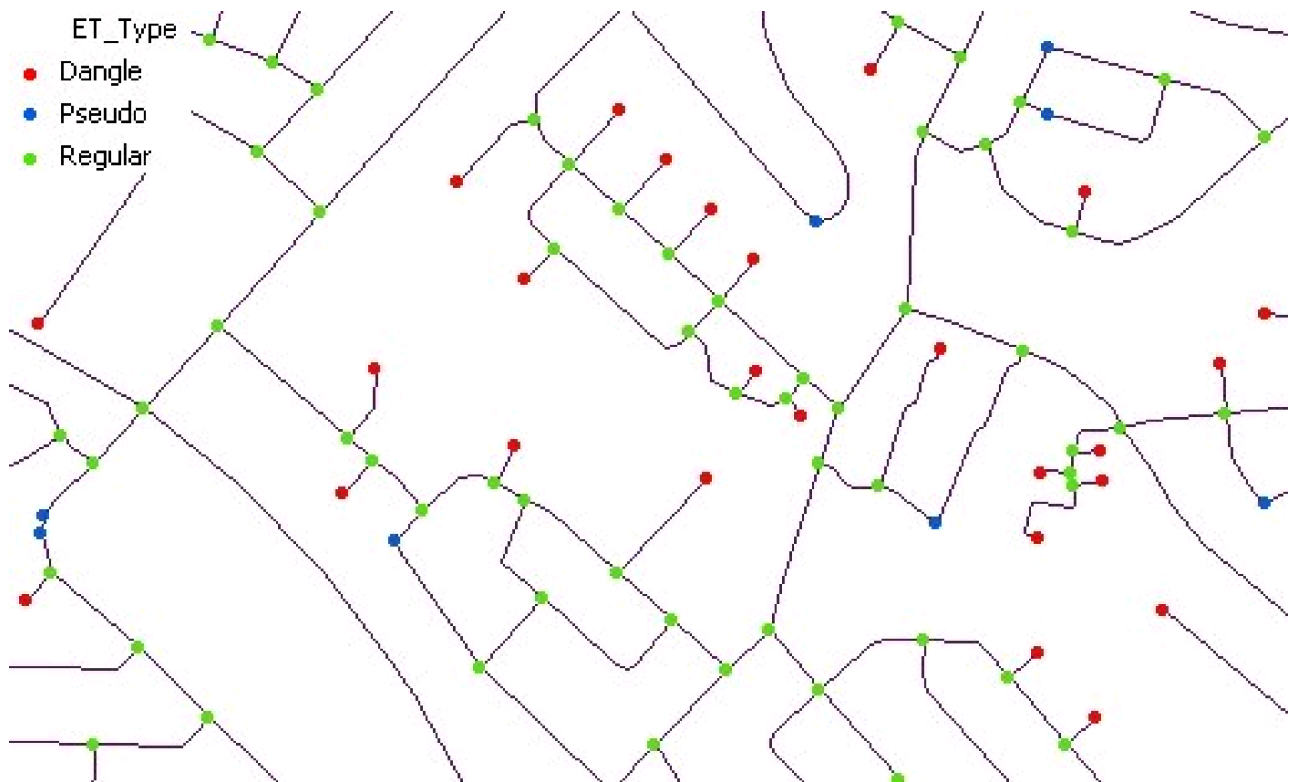
Analyzes the nodes of a polyline layer and exports them as a point layer.

Inputs:

- A polyline feature layer
- Types of nodes to be exported

Outputs:

- New point layer
 - Contains points representing the specified node types
 - Regular nodes - node where more than two polylines intersect
 - Pseudo nodes - occur where a single line connects with itself or where only two polylines intersect
 - Dangling nodes - unconnected nodes of dangling polylines
 - Several fields are added to the point attribute table:
 - [ET_Type] - the type of node.
 - [ET_Valency] - the number of polylines that connect in the node. For a Dangling node Valency = 1, for a Pseudo node Valency = 2, for a Regular node Valency ≥ 3
 - [PL_LINK1], [PL_LINK2] ...[PL_LINK9] carrying the values in the specified as a parameter link field of the polylines that intersect in a node



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ExportNodes
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Export Dangling>	A Boolean indicating whether the Dangling nodes will be exported
<Export Pseudo>	A Boolean indicating whether the Pseudo nodes will be exported.
<Export Regular>	A Boolean indicating whether the Regular nodes will be exported

<Link Field>	A String representing the name of the field which values will be used as links between the nodes and the source polylines
--------------	---

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ExportNodes", "input dataset", "output dataset", "Export Dangling", "Export Pseudo", "Export Regular", "Link Field"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ExportNodes" "input dataset" "output dataset" "Export Dangling" "Export Pseudo" "Export Regular" "Link Field"</pre>
.NET using ETGWOutX.dll	<code>ExportNodes(input dataset, output dataset, Export Dangling, Export Pseudo, Export Regular, Link Field)</code>
ArcPy	<code>arcpy.ExportNodes(input dataset, output dataset, "Export Dangling" , "Export Pseudo", "Export Regular", "Link Field")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Renode Polylines

[Running programmatically](#)

Analyzes the nodes of a polyline layer and exports them as a point feature class. Creates links between the nodes and corresponding polylines

Inputs:

- A polyline feature layer

Outputs:

- New point feature class
 - Contains points representing polyline nodes
 - Regular nodes - node where more than two polylines intersect
 - Pseudo nodes - occur where a single line connects with itself or where only two polylines intersect
 - Dangling nodes - unconnected nodes of dangling polylines
 - Two fields are added to the point attribute table :
 - [ET_Type] - the type of node.
 - [ET_Valency] - the number of polylines that connect in the node. For a Dangling node Valency = 1, for a Pseudo node Valency = 2, for a Regular node Valency ≥ 3
 - [ET_NodeId] - the ID of the nodes allowing link to the original polylines
- A copy of the original polyline layer with two fields are added to the polyline attribute table
 - [ET_FNode] - the id of the From Node of the polyline - links to a point in the Node feature class

- [ET_TNode] - the id of the To Node of the polyline - links to a point in the Node feature class

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RenodePolylines
<input dataset>	A String representing the input layer.
<Output Node Dataset>	A String - the full name of the output Nodes layer.
<Output Polyline Dataset>	A String representing the output polyline layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "RenodePolylines", "input dataset", "Output Node Dataset", "Output Polyline Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "RenodePolylines" "input dataset" "Output Node Dataset" "Output Polyline Dataset"
.NET using ETGWOuX.dll	RenodePolylines(input dataset, Output Node Dataset,Output Polyline Dataset)

ArcPy

```
arcpy.RenodePolylines(input dataset, Output Node Dataset,Output Polyline Dataset)
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Generalize Polylines

[Running programmatically](#)

Generalizes (reduces the number of vertices required to represent a polyline) the features of a polyline layer using the Douglas-Poiker algorithm

Inputs:

- A polyline feature layer
- Generalize tolerance (maximum offset) - the maximum distance that the generalized polyline will differ from the original one

Outputs:

- New polyline feature class
 - The output feature class will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	GeneralizePolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Generalize Tolerance>	A Double representing the Generalization tolerance (in the units of the spatial reference of the input layer).

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "GeneralizePolylines", "input dataset", "output dataset", "Generalize Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "GeneralizePolylines" "input dataset" "output dataset" "Generalize Tolerance"</code>
.NET using ETGWOuX.dll	<code>GeneralizePolylines(input dataset, output dataset, Generalize Tolerance)</code>
ArcPy	<code>arcpy.GeneralizePolylines(input dataset, output dataset, "Generalize Tolerance")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Densify Polylines

[Running programmatically](#)

Densifies (adds vertices to polyline at a user-specified tolerance) the features of a polyline layer.

Inputs:

- A polyline feature layer
- Maximum segment length

Outputs:

- New polyline feature class
 - The output feature class will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	DensifyPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Densify Tolerance>	A Double representing the at which new vertices will be introduced.

Running the function

ETGWPPath used in the table below is the full path to ETGWRUN.exe (E.G. "C:\Program

Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "DensifyPolylines", "input dataset", "output dataset", "Densify Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "DensifyPolylines" "input dataset" "output dataset" "Densify Tolerance"</code>
.NET using ETGWOutX.dll	<code>DensifyPolylines(input dataset, output dataset, Densify Tolerance)</code>
ArcPy	<code>arcpy.DensifyPolylines(input dataset, output dataset, "Densify Tolerance")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Smooth Polylines

[Running programmatically](#)

Smooth the features of a polyline layer using three different smoothing algorithms

Inputs:

- A polyline feature layer
- Smooth method
 - Bezier curve
 - The curve in general does not pass through any of the control points (vertices of original polyline) except the first and last.
 - The curve is always contained within the convex hull of the control points
 - Approximate the original shape rather freely
 - Fast - good for polylines with many vertices (control points) that will constrain the curve close to the original shape
 - B - Spline
 - The curve does not pass through any of the control points (vertices of original polyline) except the first and last
 - Follows better than the Bezier curve the original shape
 - Depending on the "Freedom" parameter the smoothing occurs only in the areas close to a vertex
 - B-Splines lie in the convex hull of the original polyline
 - Slower than the Bezier curve, but the results in many cases are much better

- T - Spline (Tension Spline)
 - The curve passes through all the vertices of the original polyline
 - The degree of fit can be controlled with the "Tension" parameter
 - Suitable for smoothing curves with comparatively equally spaced vertices
 - Fast with good approximation of the original polyline
- Parameters depending on the method
 - The "Smoothness" parameter (Used in all methods) defines the number of points in the output curve. The allowed values (2 to 20) in fact are point multiplier. The number of vertices of the original polyline multiplied by this value will give the number of vertices of the smoothed polyline. The larger the value of the Smoothness parameter, the slower the process will be. In most of the cases a value of 5 (default) will create smooth and representative polyline
 - The "Freedom" parameter (B-Spline only) defines how close to the original polyline the curve will be. The allowed values are from 3 to 10. Smaller values give better approximation. With large values the curve will become very similar to Bezier curve
 - The "Tension" parameter (T-Spline only) defines how close to the original polyline the curve will be. Increasing the tension is similar to pulling on the ends of a string constrained to pass through the polyline vertices. allowed values are from 1 to 100.
- Optional - Densification tolerance. In some cases the smooth parameters cannot restrict the smoothing enough. The user can restrict the effect of the smoothing by introducing new vertices in the shapes. See [Densify](#) function for details
- Optional - Generalization tolerance. The smoothing introduces in the shapes many new vertices. The user can decrease the number of vertices by using this option. See [Generalize](#) function for details.

Outputs:

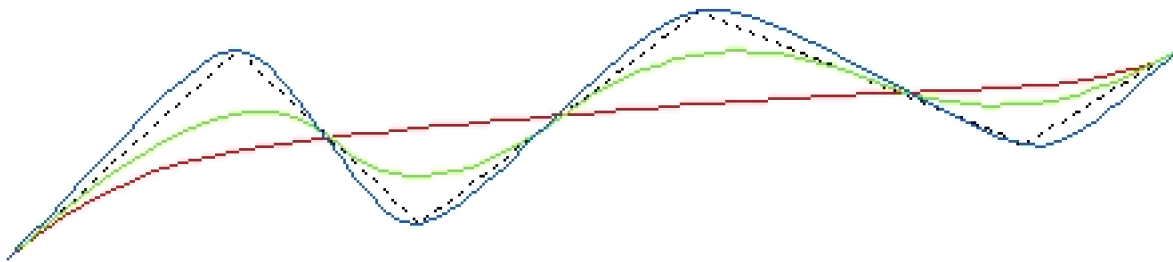
- New polyline layer
 - The output layer will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Notes :

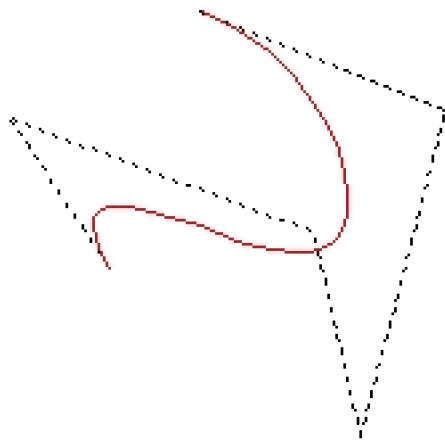
- With all methods the Start and End point of the polylines are preserved
- All the methods implement generic algorithms.
- The Generalization and Densification tolerances should be specified in the units of the spatial reference of the input layer

Examples:

Smooth results: Dashed - Original; Red - Bezier; Green - B-Spline; Blue- T-Spline (Default parameters)

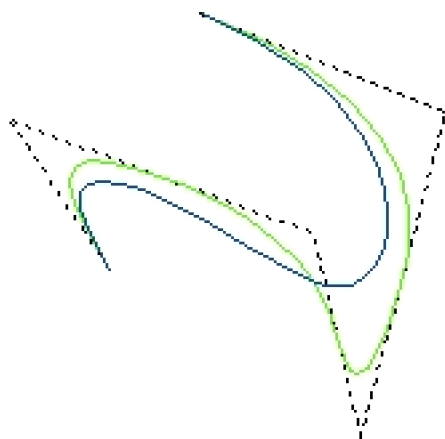


Bezier



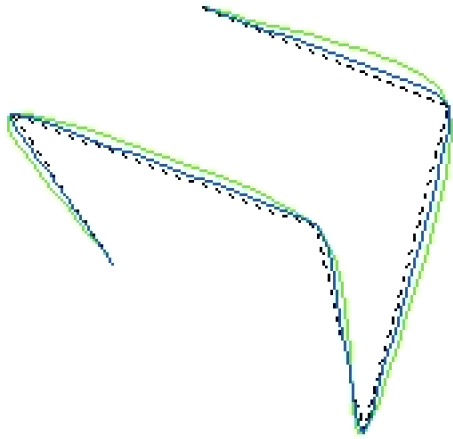
B-Spline

- Green - Freedom = 3
- Blue - Freedom = 5



T-Spline

- Green - Tension = 30
- Blue - Tension = 90



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SmoothBezier
Function Name	SmoothBSpline
Function Name	SmoothTSpline
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Smoothness>	An Integer that defines the number of points in the output curve. The allowed values (2 to 20) in fact are point multiplier. The number of vertices of the original polyline multiplied by this value will give the number of vertices of the smoothed polyline. The larger the value of the parameter, the slower the process will be.
<Freedom>	Only for B-Spline. An Integer that defines how close to the original polyline the curve will be. The allowed values are from 3 to 10. Smaller values give better approximation. With large values the curve will become very similar to Bezier curve

<Tension>	Only for T-Spline. An Integer that defines how close to the original polyline the curve will be. Increasing the tension is similar to pulling on the ends of a string constrained to pass through the polyline vertices. allowed values are from 1 to 100.
{Densify Before}	A Double representing the Densification tolerance. In some cases the smooth parameters cannot restrict the smoothing enough. The user can restrict the effect of the smoothing by introducing new vertices in the shapes. See Densify function for details
{Generalize After}	A Double representing the Generalization tolerance. The smoothing introduces in the shapes many new vertices. The user can decrease the number of vertices by using this option. See Generalize function for details.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

SmoothBezier

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SmoothBezier", "input dataset", "output dataset", "Smoothness", "Densify Before", "Generalize After"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SmoothBezier" "input dataset" "output dataset" "Smoothness" "Densify Before" "Generalize After"</code>
.NET using ETGWOuX.dll	<code>SmoothBezier(input dataset, output dataset, Smoothness, Densify Before, Generalize After)</code>
ArcPy	<code>arcpy.SmoothBezier(input dataset, output dataset, "Smoothness", "Densify Before", "Generalize After")</code>

SmoothBSpline

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SmoothBSpline", "input dataset", "output dataset", "Smoothness", "Freedom", "Densify Before", "Generalize After"])</code>

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "SmoothBSpline" "input dataset" "output dataset" "Smoothness" "Freedom" "Densify Before" "Generalize After"
.NET using ETGWOutX.dll	SmoothBSpline(input dataset, output dataset, Smoothness, Freedom, Densify Before, Generalize After)
ArcPy	arcpy.SmoothBSpline(input dataset, output dataset, "Smoothness" , "Freedom", "Densify Before", "Generalize After")

SmoothTSpline

Language	Syntax
Python	subprocess.call([ETGWPath, "SmoothTSpline", "input dataset", "output dataset", "Smoothness", "Tension", "Densify Before", "Generalize After"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "SmoothTSpline" "input dataset" "output dataset" "Smoothness" "Tension" "Densify Before" "Generalize After"
.NET using ETGWOutX.dll	SmoothTSpline(input dataset, output dataset, Smoothness, Tension, Densify Before, Generalize After)
ArcPy	arcpy.SmoothTSpline(input dataset, output dataset, "Smoothness" , "Tension", "Densify Before", "Generalize After")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Flip Polylines

[Running programmatically](#)

Changes the direction of the polylines from a polyline layer based on user defined start point.

Inputs:

- A polyline feature layer
- Start Point to be used

Outputs:

- New Polyline layer - all polylines have their from node closer to the user specified start points

Notes :

- The corners and the middle points of the smallest bounding rectangle of each polyline are used

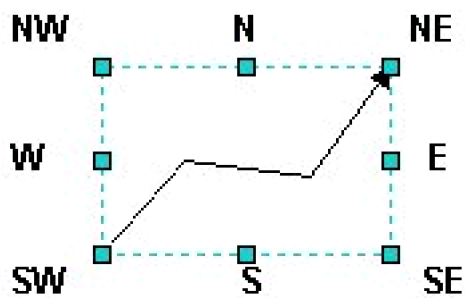
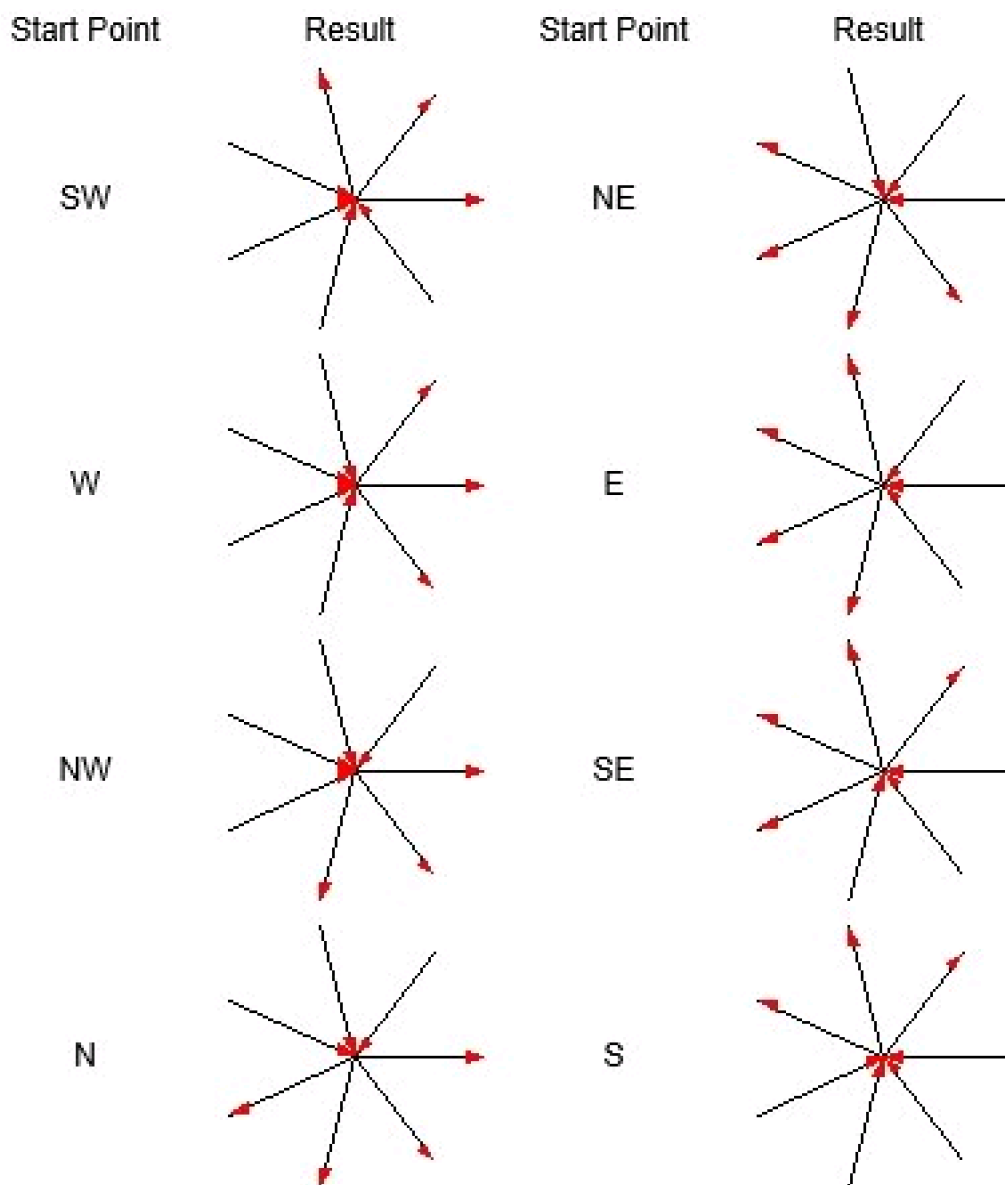


Illustration:



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FlipPolylinesBasedOnPoint
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.

>Flip Point Location>	<p>A String indicating the location of the flip point. See illustration above. Valid values:</p> <ul style="list-style-type: none"> • "ll", "lowerleft", "sw", "south-west" • "lr", "lowerright", "se", "south-east" • "ul", "upperleft", "nw", "north-west" • "ur", "upperright", "ne", "north-east" • "b", "bottom", "s", "south" • "t", "top", "n", "north" • "l", "left", "w", "west" • "r", "right", "e", "east"
-----------------------	---

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "FlipPolylinesBasedOnPoint", "input dataset", "output dataset", "Flip Point Location"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "FlipPolylinesBasedOnPoint" "input dataset" "output dataset" "Flip Point Location"
.NET using ETGWOuX.dll	FlipPolylinesBasedOnPoint(input dataset, output dataset,Flip Point Location)
ArcPy	arcpy.FlipPolylinesBasedOnPoint(input dataset, output dataset,Flip Point Location)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Contour Gaps

[Running programmatically](#)

Cleans the gaps in a polyline dataset representing contours.

Inputs:

- A Polyline dataset
- A field representing the elevation value of the contours
- Tolerance - the gaps smaller than this tolerance will be closed.

Outputs:

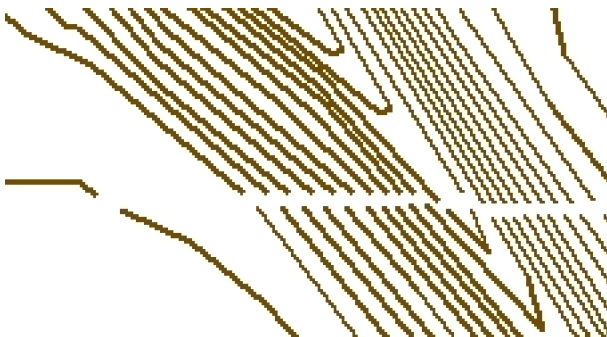
- New polyline feature class. The gaps in the contours that are smaller than the selected tolerance are closed.

Notes:

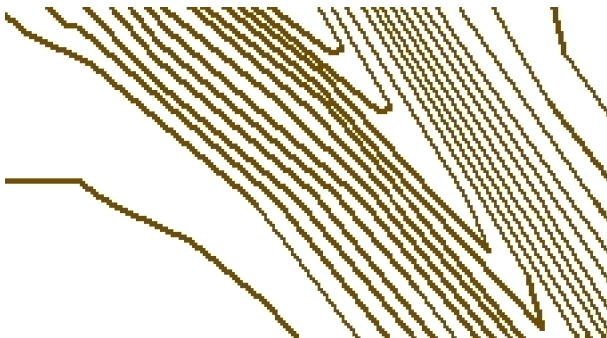
- The function is designed specifically for contours, but it can be used on datasets representing different features.
- Always inspect the results before accepting them as valid.

Example :

Input Contours



Resulting Polylines



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanContourGaps
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Gap Size>	A Double representing the maximum gaps to be removed (in the units of the input dataset).
<Elevation Field>	A String - the name of the field having the elevation value of the contours.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CleanContourGaps", "input dataset", "output dataset", "Gap Size", "Elevation Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanContourGaps" "input dataset" "output dataset" "Gap Size" "Elevation Field"

.NET using ETGWOtX.dll	CleanContourGaps(input dataset, output dataset, Gap Size, Elevation Field)
ArcPy	arcpy.CleanContourGaps(input dataset, output dataset, "Gap Size", "Elevation Field")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

PolylineZ Characteristics

[Running programmatically](#)

Calculates several characteristics of PolylineZs. The results are stored in fields in the attribute table of the output layer.

Inputs:

- A PolylineZ feature layer
- Linear precision - the number of digits after the decimal point for linear measures
- Angular precision - the number of digits after the decimal point for angular measures
- NODATA value - a number that represents undefined Z values. If the Z values of a geometry are interpolated from a surface and some of the vertices of the geometry are outside of the extent of the surface, they will not have Z values. Since ArcGIS does not accept NaN (Not a Number) values in Z enabled shapes, a numeric value is assigned to these vertices. If the Features To 3D function of ET GeoWizards is used to derive the Z values, the NODATA value is 999999. When calculating Z characteristics this values need to be ignored. Segments that have a vertex with NODATA Z value will be ignored in the calculations.

Outputs:

The results are stored in the attribute table of the output layer. The linear measures are in the units of the spatial reference of the input dataset. The slope is measured in decimal degrees (from -90 to +90). The following fields are added to the attribute table

- [3D_Length] - the true 3D length of the polyline
- [2D_Length] - the 2D length of the polyline
- [Max_Z] - Maximum Z value
- [Min_Z] - Minimum Z value
- [Len_Up] - distance uphill
- [Len_Down] - distance downhill
- [H_Up] - total increase in height

- [H_Down] - total decrease in height
- [Av_S_Up] - average slope uphill
- [Max_S_Up] - maximum slope uphill
- [Av_S_Down] - average slope downhill
- [Max_S_Down] -maximum slope downhill

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolylineZCharacteristics
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Linear Precision}	An Integer indicating the number of digits after the decimal point for linear measures.
{Angular Precision}	An Integer indicating the number of digits after the decimal point for angular measures.
{No Data}	A Double - represents undefined Z values."

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPath, "PolylineZCharacteristics", "input dataset", "output dataset", "Linear Precision", "Angular Precision", "No Data"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "PolylineZCharacteristics" "input dataset" "output dataset" "Linear Precision" "Angular Precision" "No Data"</code>
.NET using ETGWOuX.dll	<code>PolylineZCharacteristics(input dataset, output dataset, Linear Precision, Angular Precision", No Data)</code>
ArcPy	<code>arcpy.PolylineZCharacteristics(input dataset, output dataset, "Linear Precision" , "Angular Precision", "No Data")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polyline Characteristics

[Running programmatically](#)

Calculates some characteristics of the polylines from a polyline dataset

Inputs:

- A Polyline layer

Outputs:

- A new Polyline dataset
- All original attributes are preserved.
- New fields added to the attribute table
 - ET_Sinuous - the sinuosity of the polyline calculated as ratio of the length of the polyline and the length of the line connecting the start and end points of the polyline. The value ranges from 1 (case of straight line) to infinity (case of a closed polyline). In case of infinity a 0 is recorded in the attribute table. See illustration below.

- ET_Vert - the number of vertices of the polyline

- ET_Dir - the general direction of the polyline - the direction in decimal degrees



measured in North Azimuth of the line connecting the start and end points of the polyline (see illustration below).

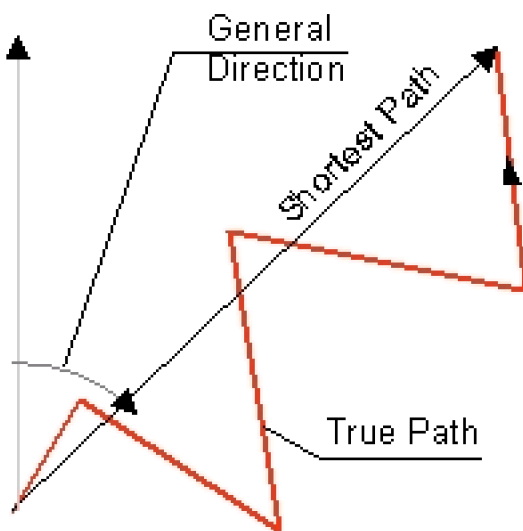
- ET_Parts - the number of parts that the polyline has
- ET_HasArcs - if the polyline has true arc segments - 1 otherwise - 0
- ET_Closed - if the polyline is closed - 1 otherwise - 0
- ET_Fract - the fractal dimension (indication of the complexity) of the polyline. The value

is between 1 and 2. The more complex the polyline is the larger the fractal dimension will be.

Notes:

- Fractal Dimension of the polylines is calculated using the Box Counting method (1)
- Calculating the Fractal Dimension is time consuming. If you don't need this characteristic, uncheck the option for faster processing.

Illustration:



$$\text{Sinuosity} = \frac{\text{True Path Length}}{\text{Shortest Path Length}}$$

References:

1. Bourke, P., 1993. Fractal Dimension Calculator User Manual, Online. Available: <http://paulbourke.net/fractals/fracdim/>

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	PolylineCharacteristics
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Calculate Fractal}	A Boolean indicating whether to calculate fractal dimension or not.
{Precision}	An Integer between 0 and 8 representing the number of places after the decimal point to be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolylineCharacteristics", "input dataset", "output dataset", "Calculate Fractal", "Precision"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolylineCharacteristics" "input dataset" "output dataset" "Calculate Fractal" "Precision"</pre>
.NET using ETGWOutX.dll	<code>PolylineCharacteristics(input dataset, output dataset, Calculate Fractal, Precision)</code>
ArcPy	<code>arcpy.PolylineCharacteristics(input dataset, output dataset, "Calculate Fractal", "Precision")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Flip PolylineZ

[Running programmatically](#)

Changes the directions of the polylines from a PolylineZ dataset. In the resulting feature class the polylines will be oriented Up Slope (start from the node with the lower Z value and finish at the node with higher Z value) or Down Slope (start from the node with the higher Z value and finish at the node with lower Z value)

Inputs:

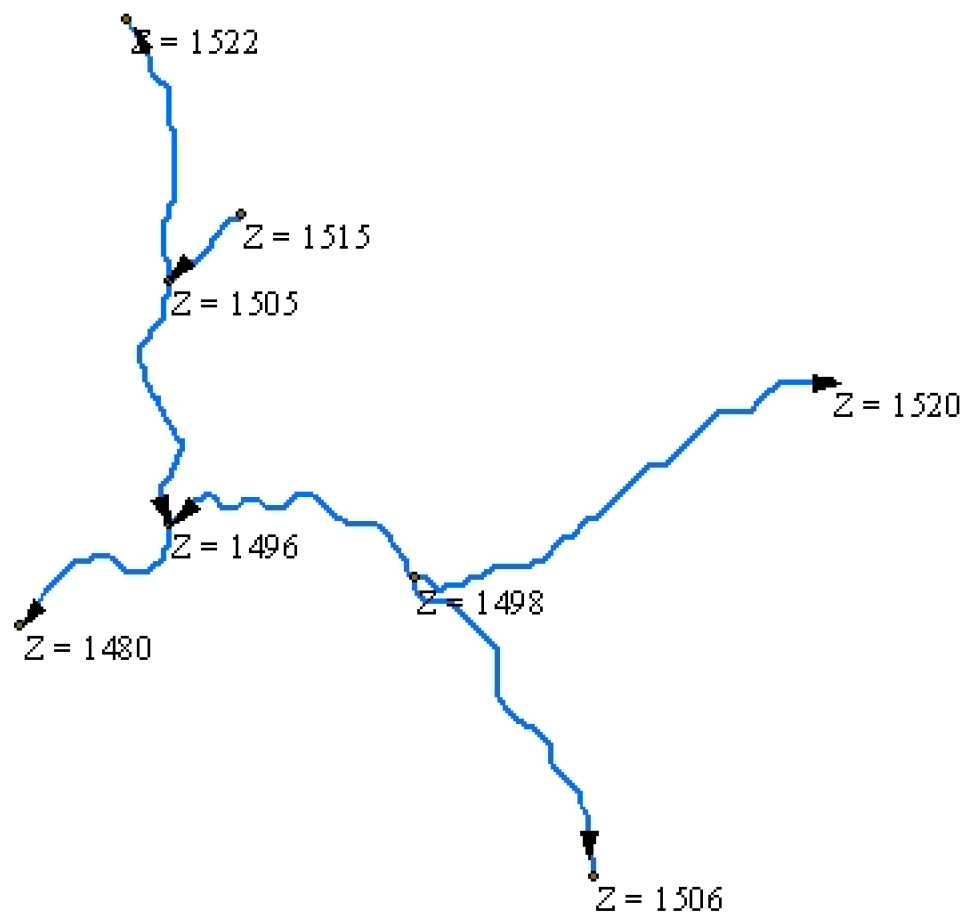
- A PolylineZ feature layer
- Flip Option

Outputs:

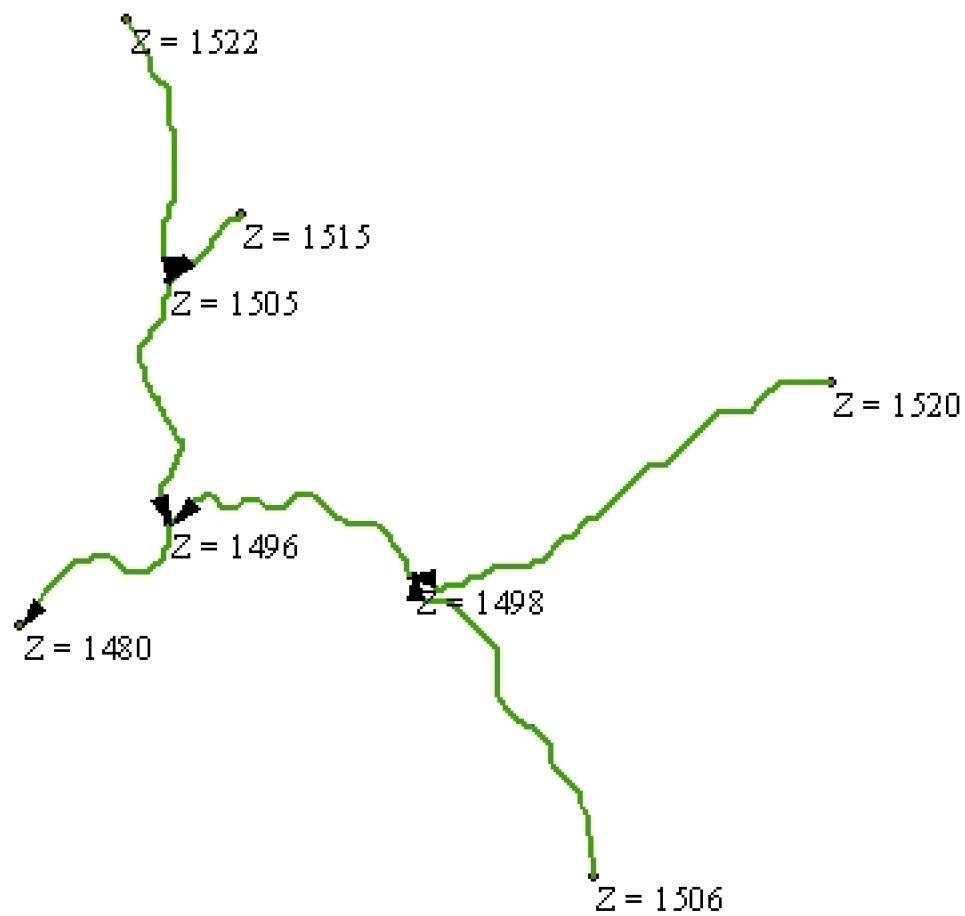
- A PolylineZ feature class.
 - A field called ET_Status will be added to the attribute table. The values in this field will indicate whether a polyline has been flipped ("Flipped") or not ("Original")

Example:

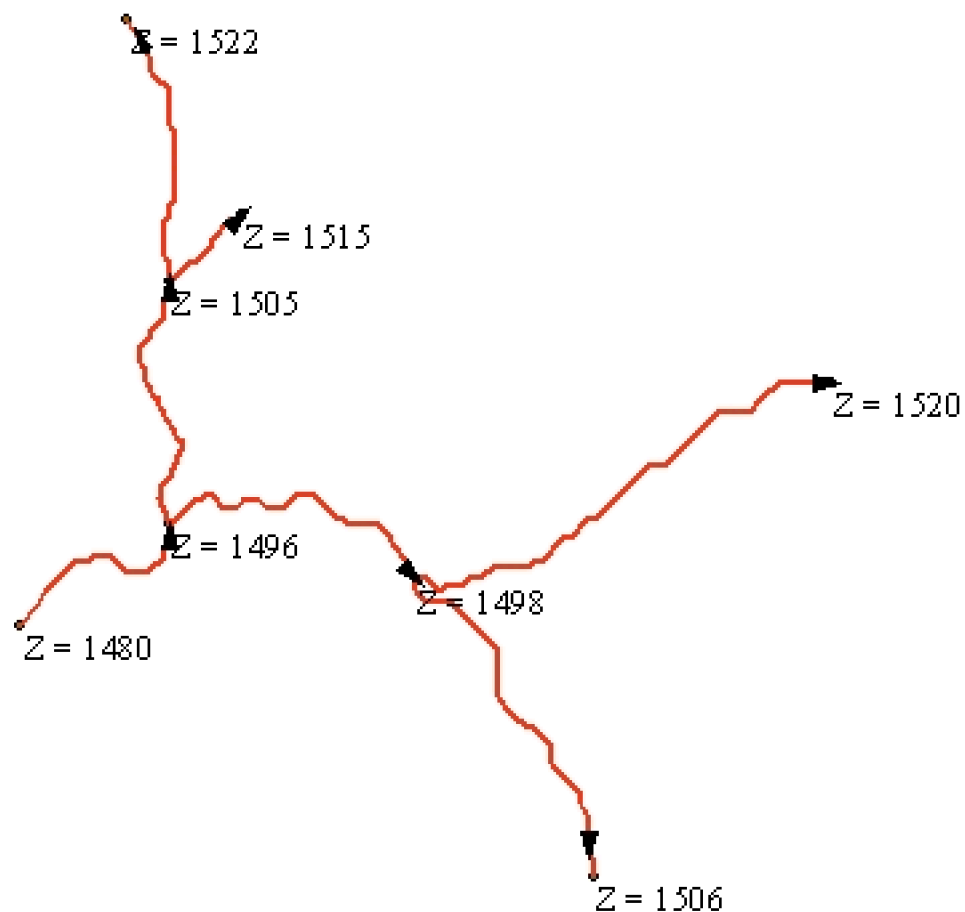
Input Dataset



Flipped - Down Slope option



Flipped - Up Slope option



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FlipPolylinesZ
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
>Down Slope>	A Boolean indicating the direction of the slope. TRUE = Down Slope.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FlipPolylinesZ", "input dataset", "output dataset", "Down Slope"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "FlipPolylinesZ" "input dataset" "output dataset" "Down Slope"</code>
.NET using ETGWOuX.dll	<code>FlipPolylinesZ(input dataset, output dataset,Down Slope)</code>
ArcPy	<code>arcpy.FlipPolylinesZ(input dataset, output dataset,Down Slope)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Polygons

[Running programmatically](#)

Ensures topological correctness of a polygon feature data set.

Inputs:

- A polygon feature layer
- Fuzzy tolerance

Outputs:

- New topologically correct Polygon dataset (no overlaps present)
 - Redundant data (overlaps and gaps smaller than the fuzzy tolerance) will be eliminated
 - The overlaps greater than the fuzzy tolerance are converted into new polygons.
 - Every new polygon carries the attributes of one of the source overlapping polygons
 - The attributes of the input data set are preserved
- Optional Point feature class that identifies the overlaps in the input data set. Each point represents an overlapping polygon in this location. A new field (OriginalID) is added to the point attribute table where the ID of the original polygon represented by this point is recorded.

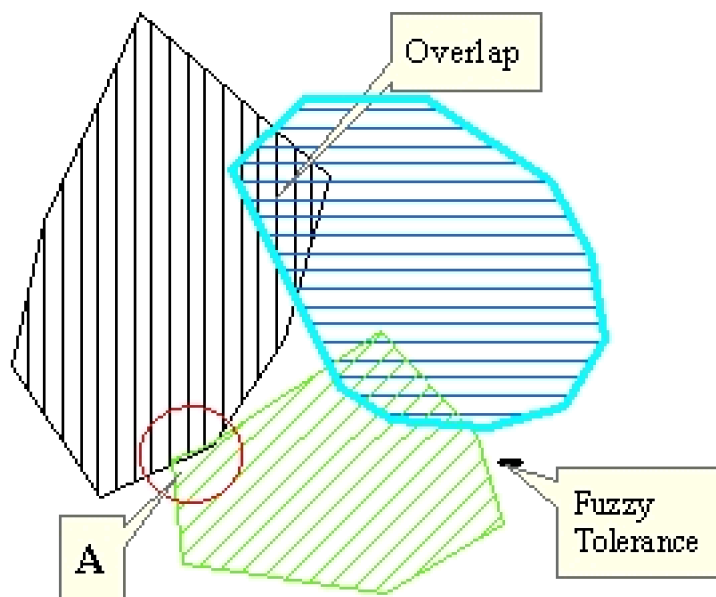
Notes :

- The default Fuzzy tolerance is calculated from the extents of the input layer using the formulae $(W + H) / 2000000$ where W and H are the with and height of the extent envelope.
- Larger values of the Fuzzy tolerance may be used to clean some bigger Gaps and Slivers, but it might lead to unwanted approximation of the input shapes. The better option is to use Fuzzy tolerance close to the default and then clean the remaining Gaps and Slivers with the Clean Gaps function and Eliminate function

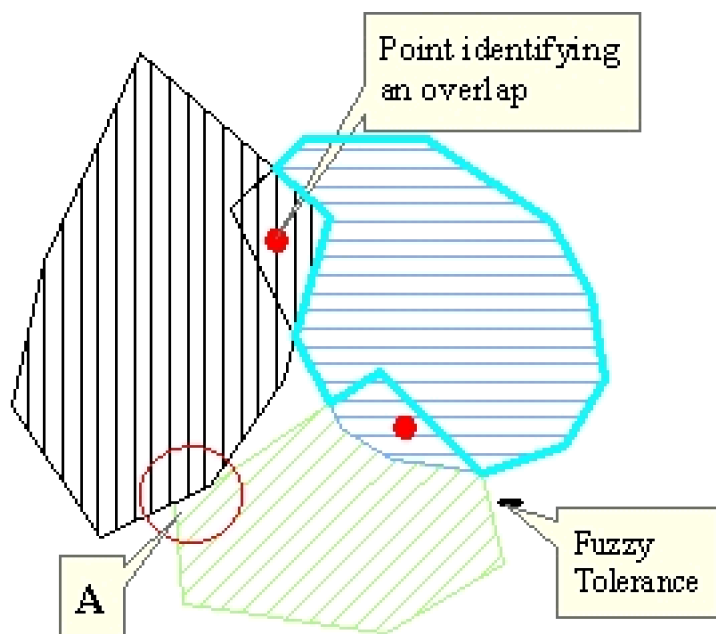
- If a Fuzzy tolerance of 0 is specified, the function will use the default Fuzzy Tolerance (see above).

Example:

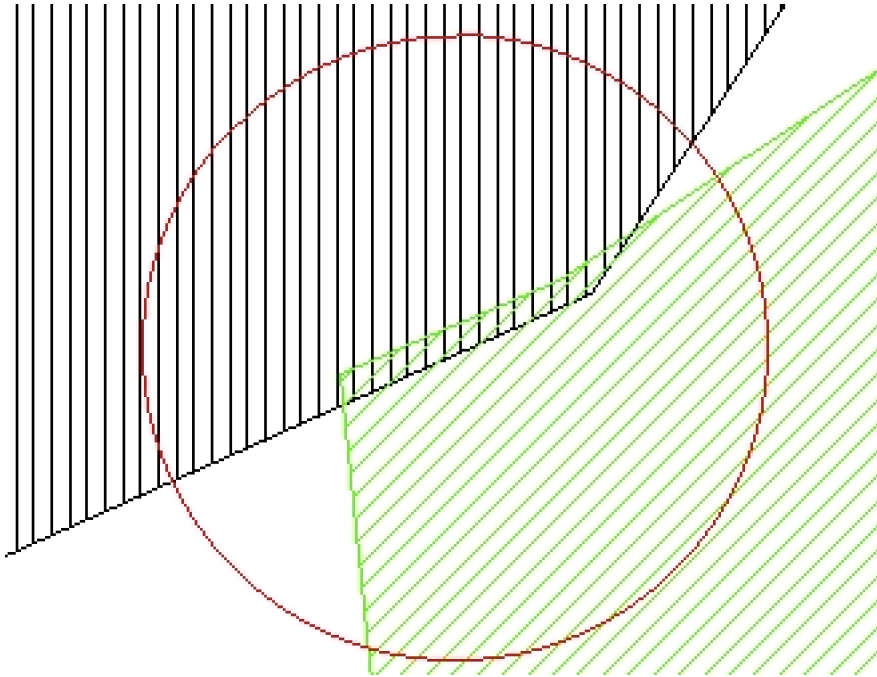
Input Layer



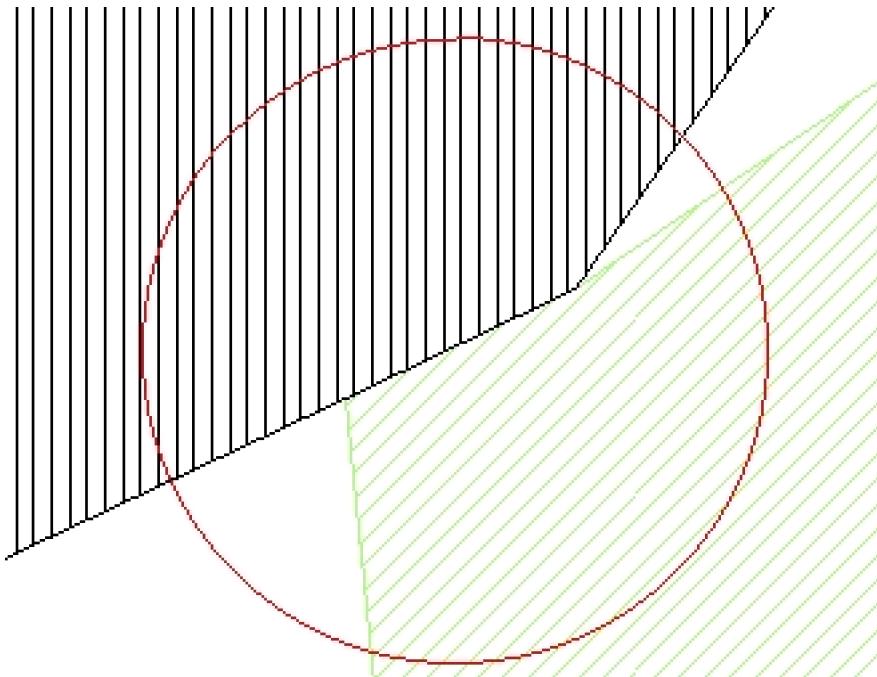
After Clean



Detail A before Clean



Detail A after Clean



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanPolygons

<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Fuzzy Tolerance>	A Double representing the Fuzzy Tolerance.
{Overlaps Name}	A String representing the output point layer indicating the overlaps in the original polygons.
{Sort Field}	A String representing a field to be used to sort the data before processing. The first features in the input dataset after sorting using that field will have priority during the cleaning process.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CleanPolygons", "input dataset", "output dataset", "Fuzzy Tolerance", "Overlaps Name", "Sort Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanPolygons" "input dataset" "output dataset" "Fuzzy Tolerance" "Overlaps Name" "Sort Field"
.NET using ETGWOuX.dll	CleanPolygons(input dataset, output dataset, Fuzzy Tolerance, Overlaps Name, Sort Field)
ArcPy	arcpy.CleanPolygons(input dataset, output dataset, "Fuzzy Tolerance", "Overlaps Name", "Sort Field")

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Eliminate

[Running programmatically](#)

Eliminates unwanted polygons (slivers) by merging them into the neighboring polygons.

Inputs:

- A polygon feature layer
- Selection method
 - Attribute query - introduces a query builder. The user can select the polygons to be eliminated using any query expression
 - Thickness ratio is expressed as a ratio of the polygon area versus the area of its minimal bounding square. The ratio will have value of 1 for a square. The smaller the value is, the thinner the polygon is. It is a good way of identifying thin polygons (possible slivers).
 - Circularity ratio - for a circle the circularity will be 1. The thinner the polygon is the smaller the circularity will be. This is another way of identifying slivers
- Elimination method
 - Join (Largest area) - will join selected polygons with neighboring polygons that have the largest area
 - Join (Longest boundary) - will join selected polygons with neighboring polygons with the longest common border .
 - Join to the neighbor with the same value in the selected field as the sliver polygon.

Outputs:

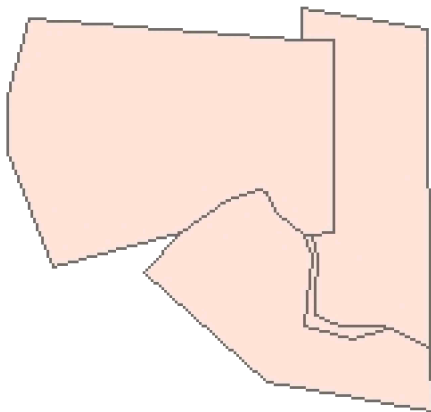
- New polygon feature class with selected polygons eliminated.

Notes :

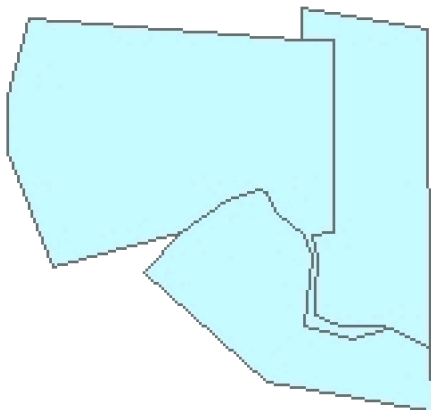
- Important! If the input layer has overlapping polygons, there might be some missing polygons in the output. It is strongly recommended to use the Clean Polygons function before executing Eliminate.
- Some of the very tiny slivers can be eliminated with the [Clean Polygon Wizard](#) using appropriate value for Fuzzy tolerance, however eliminating larger slivers using this method is not recommended since some unwanted approximation of the polygon shapes might occur.
- When using programatically the only selection method is Attribute Query. You can use the [Polygon Characteristics function](#) to get values for Circularity and Thickness for each polygon and then execute the Eliminate Function using Attribute Query.

Examples:

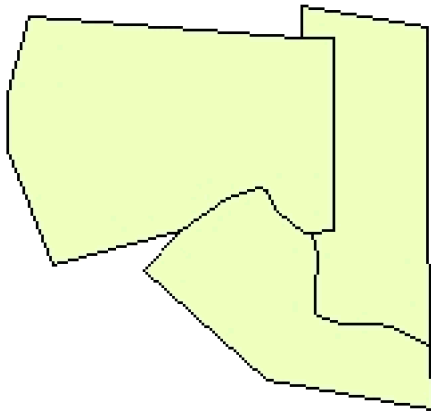
Before Eliminate



Eliminated with Largest Area option



Eliminated with Longest boundary option



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	Eliminate
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Eliminate Method>	Required. A String - valid values: <ul style="list-style-type: none">• "Largest" - The polygons will be eliminated by joining them to the neighboring polygons that have the largest area• "Longest" - The polygons will be eliminated by joining them to the neighboring polygons with the longest common border .• "Join Field" - The polygons will be eliminated by joining them to the neighboring polygons with the same value in the selected field as the sliver polygons.
<SQL Expression>	A String representing the selection expression. Example: Shape_Area < 200 AND Name = 'a'

{Join Field}	A String representing a field name to be used in the elimination process. Used only if the elimination method is "Join Field"
--------------	---

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "Eliminate", "input dataset", "output dataset", "Eliminate Method", "SQL Expression", "Join Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "Eliminate" "input dataset" "output dataset" "Eliminate Method" "SQL Expression" "Join Field"
.NET using ETGWOuX.dll	Eliminate(input dataset, output dataset, Eliminate Method, SQL Expression", Join Field)
ArcPy	arcpy.Eliminate(input dataset, output dataset, "Eliminate Method" , "SQL Expression", "Join Field")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clean Gaps

[Running programmatically](#)

Finds the gaps between polygons and holes within polygons and fills them with new polygons.

Inputs:

- A polygon feature layer

Outputs:

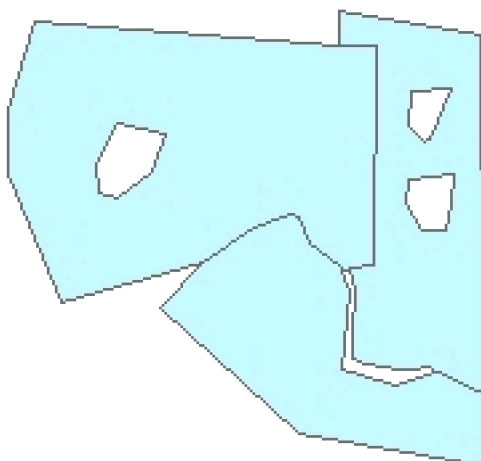
- New polygon layer with all the gaps converted to polygons.
- New field is added to the attribute table :
 - [ET_Gap] - the newly added polygons have value - "Gap"

Notes :

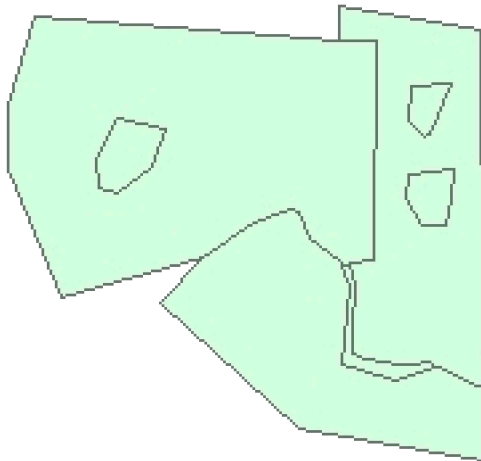
- The [Eliminate function](#) can be used to join the gaps to the neighboring polygons

Examples:

Before Clean Gaps



After Clean Gaps



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CleanGaps
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Clean Polygons}	A Boolean indicating whether the function to clean the polygons from overlaps.
{Fuzzy Tolerance}	A Double representing the Fuzzy Tolerance.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CleanGaps", "input dataset", "output dataset", "Clean Polygons", "Fuzzy Tolerance"])

.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CleanGaps" "input dataset" "output dataset" "Clean Polygons" "Fuzzy Tolerance"</pre>
.NET using ETGWOutX.dll	<pre>CleanGaps(input dataset, output dataset, Clean Polygons, Fuzzy Tolerance)</pre>
ArcPy	<pre>arcpy.CleanGaps(input dataset, output dataset, "Clean Polygons", "Fuzzy Tolerance")</pre>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Dissolve Polygons

[Running programmatically](#)

Dissolves (aggregates) polygons based on user specified attributes. The resulting polygon data set does not contain multi-part polygons

Inputs

- A polygon feature layer
- Fields to be used for dissolving.
- Update rules for the rest of the fields to be transferred.

Outputs

- An aggregated polygon feature class. Only the polygons with common boundaries that have the same values for the dissolve fields will be aggregated
- Multi-part polygons will be created if specified by the user.
- The attributes will be transferred according the user specified rules. For the fields with no specified update rule, date and blob fields, the aggregated feature will carry the attributes of the first feature.

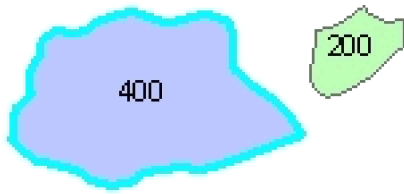
Example:

Input Layer Dissolve field = "Dissolve"



Dissolve	Population
aa	100
aa	200
aa	300

After Dissolve



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	DissolvePolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Dissolve Fields}	A String representing a list (separated with ";") with the field names to be used for dissolving.
{Statistic Fields}	A String representing a list (separated with ";") with the field names for which statistics will be created. Example: "Field1 Sum;Field2 Max;Field3 Min"
{Create Multiparts}	A Boolean indicating whether the function will create multipart polygons.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "DissolvePolygons", "input dataset", "output dataset", "Dissolve Fields", "Statistic Fields", "Create Multiparts"])

.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "DissolvePolygons" "input dataset" "output dataset" "Dissolve Fields" "Statistic Fields" "Create Multiparts"</pre>
.NET using ETGWOtX.dll	<pre>DissolvePolygons(input dataset, output dataset, Dissolve Fields, Statistic Fields, Create Multiparts)</pre>
ArcPy	<pre>arcpy.DissolvePolygons(input dataset, output dataset, "Dissolve Fields", "Statistic Fields", "Create Multiparts")</pre>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Build Polygons

[Running programmatically](#)

Builds polygon layer from a polyline layer

Inputs:

- A polyline feature layer
- Optional point dataset that represents polygon labels and is to be used for attaching attributes to the resulting polygons.

Outputs:

- New polygon feature class

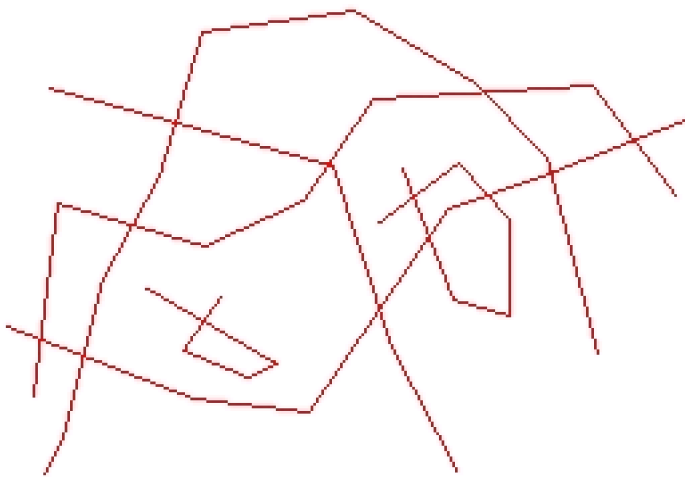
Notes :

- The process goes through several steps
 - Cleans the polyline theme with user specified Fuzzy tolerance - creates intersections and removes duplicate polylines. It is highly recommended the polyline layer to be cleaned beforehand with the Clean Polyline function.
 - During the second step the process removes all non polygon elements. All the polylines having a dangling node will be removed. Note that the function will snap the dangling nodes to the closest polylines only if they are within the Fuzzy tolerance, therefore it is very important to use [Clean Dangling Nodes function](#) in order to ensure that there will be no loss of data.
 - The third step is the actual building of the polygons

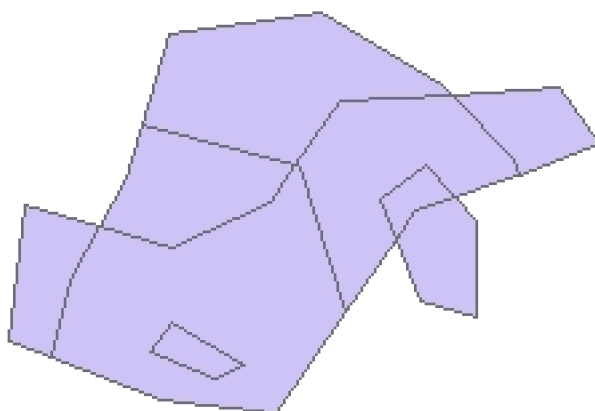
- Although not required, three data preparation steps are very important as mentioned above
 - Use [Clean Polyline function](#) to clean the input polyline data set.
 - Use [Clean Dangling Nodes function](#) to clean all dangling polylines.
 - Use [Export Nodes function](#) to verify that all the polylines are correctly connected.

Example:

Source Polyline Layer



Result Polygon Layer



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	BuildPolygons
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Fuzzy Tolerance>	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used to resolve intersections.
{Label Points}	A point layer to be used as a source for the polygon attributes.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "BuildPolygons", "input dataset", "output dataset", "Fuzzy Tolerance", "Label Points"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "BuildPolygons" "input dataset" "output dataset" "Fuzzy Tolerance" "Label Points"</code>
.NET using ETGWOtX.dll	<code>BuildPolygons(input dataset, output dataset, Fuzzy Tolerance, Label Points)</code>
ArcPy	<code>arcpy.BuildPolygons(input dataset, output dataset, "Fuzzy Tolerance", "Label Points")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Centerlines

[Running programmatically](#)

Creates centerlines from polygon features.

Inputs:

- A polygon feature class.
- Maximum Width.
- Minimum Width.
- Centerline type
 - Inside - the centerlines will be created inside the polygons - suitable for deriving centerlines from rivers and streets represented by polygons
 - Outside - the centerlines will be created in the gaps between polygons - suitable for deriving street centerlines from cadastral data.

Outputs:

- New polyline layer

Notes:

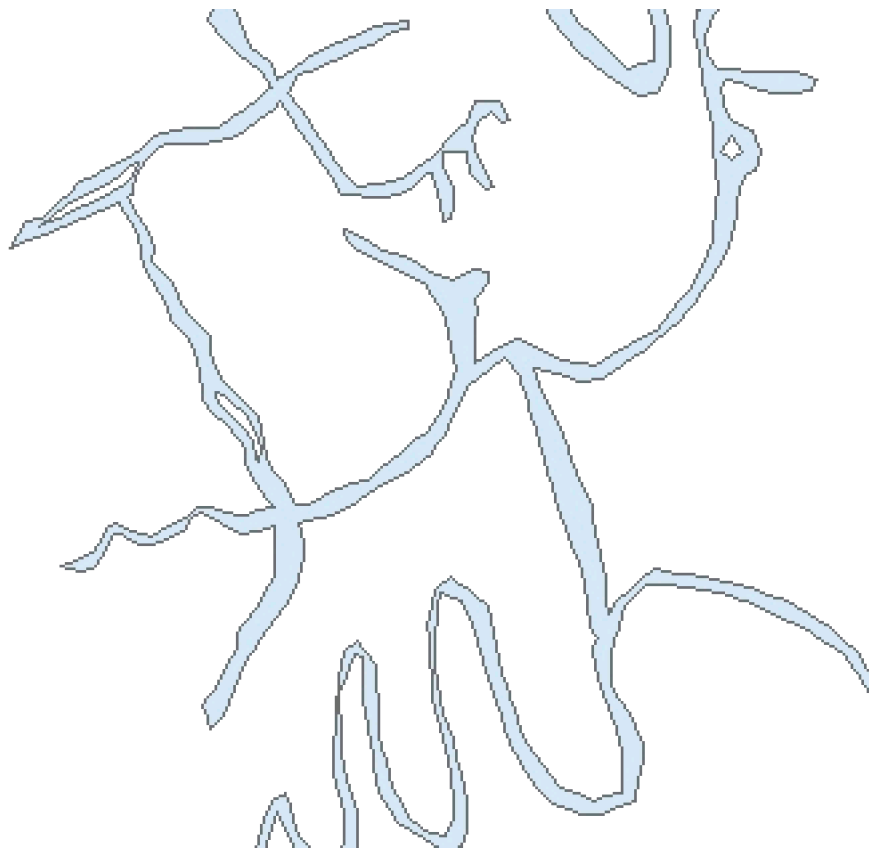
- If the "Inside" option is used, the input polygons should represent linear by nature features (rivers, roads, etc.)
- If the "Outside" option is used, the gaps between the input polygons should represent linear by nature features (streets, etc.)
- The new feature class will not have any attributes.
- Maximum and Minimum widths should be specified in the units of the spatial reference of the input feature class
- Use reasonable for your data Maximum and Minimum widths.

- The results might contain some unwanted lines. Inspect the results and remove undesired features.
- Use the [Smooth Polylines](#) function after inspecting the results.

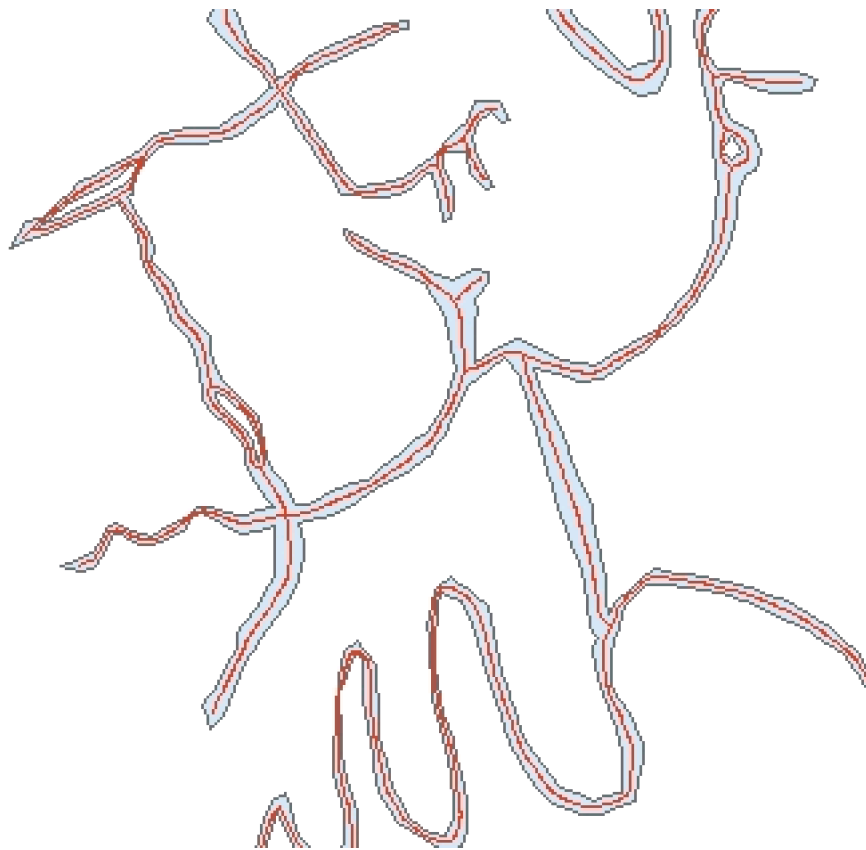
Examples:

Centerlines Inside Polygons

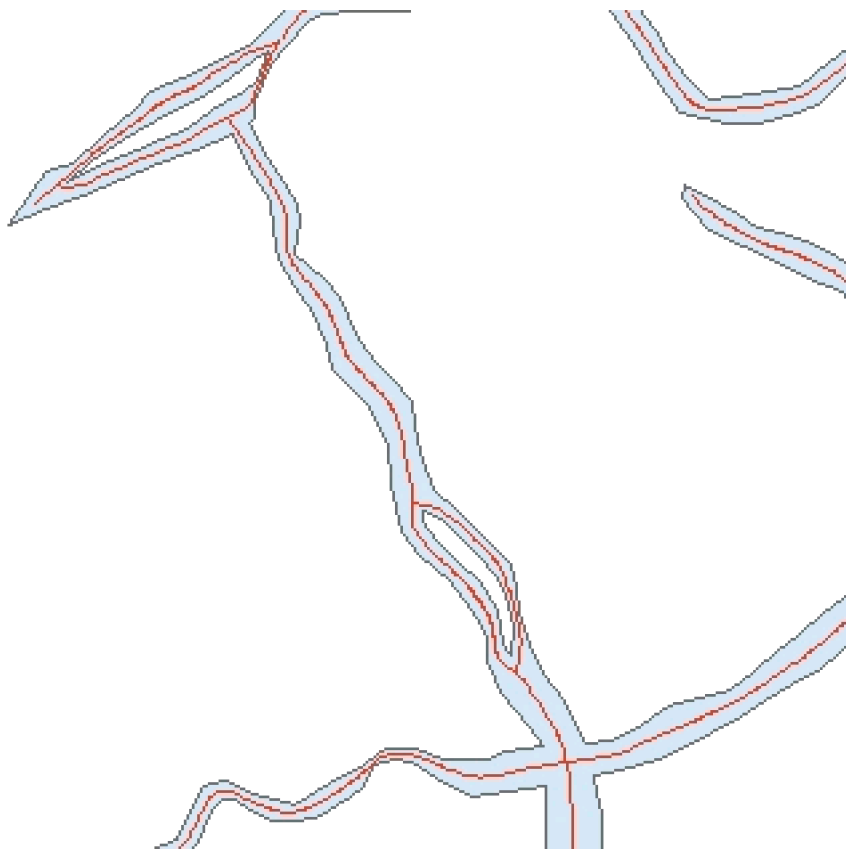
Input Dataset



Result



Result (Detail)

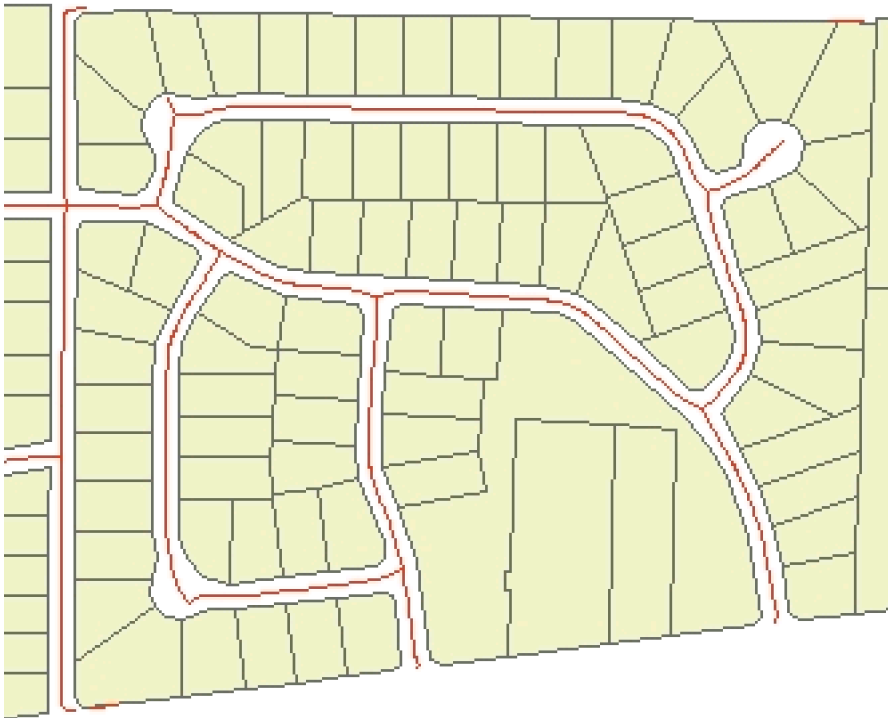


Centerlines Outside Polygons

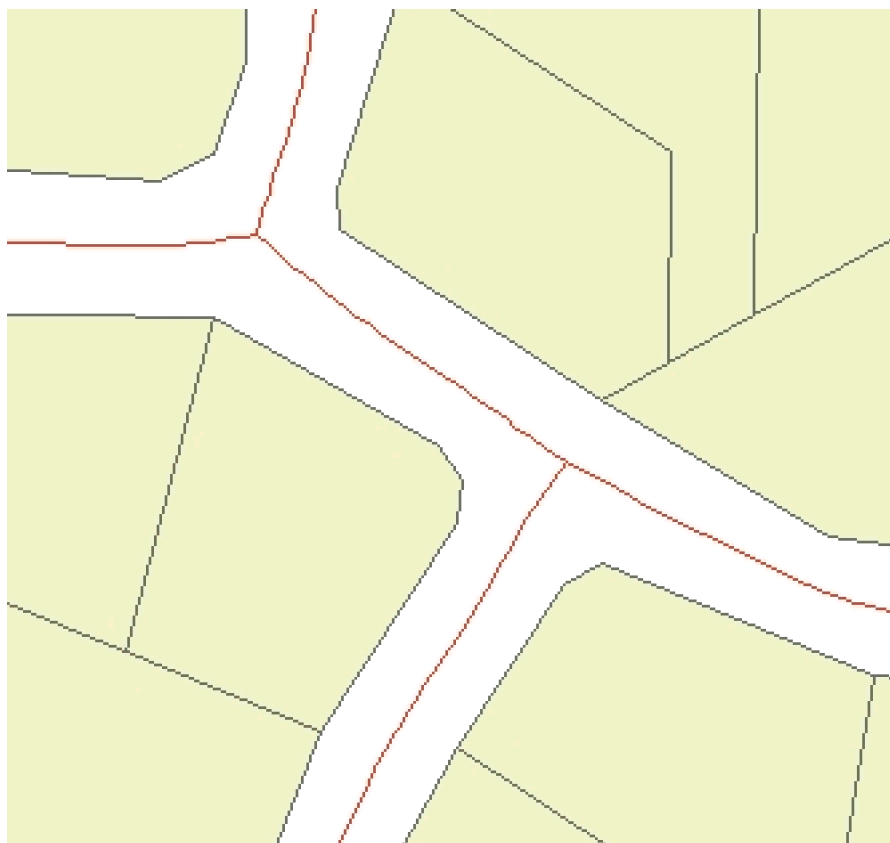
Input Dataset



Result



Result (Detail)



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CreateCenterlines
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<location option>	A String indicating the location of the centerlines to be created. "in" - centerlines inside the input polygons (rivers, streets, etc.) - "out" - centerlines outside the input polygons (cadastre, etc.)
<Max Width>	A Double representing the maximum width of the polygons (in the units of the spatial reference of the input dataset).

<Min Width<	A Double representing the minimum width of the polygons (in the units of the spatial reference of the input dataset.
-------------	--

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CreateCenterlines", "input dataset", "output dataset", "location option" "Max Width", "Min Width"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateCenterlines" "input dataset" "output dataset" "location option" "Max Width" "Min Width"</pre>
.NET using ETGWOuX.dll	<code>CreateCenterlines(input dataset, output dataset, location option, Max Width, Min Width)</code>
ArcPy	<code>arcpy.CreateCenterlines(input dataset, output dataset, location option, "Max Width", "Min Width")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygon Global Snap

[Running programmatically](#)

Snaps the features of a polygon layer to another layer (Point, Polyline or Polygon)

Inputs

- A polygon layer to be snapped
- A snap layer - point, multipoint polyline or polygon
- Snap tolerance
- Snap options 1 (Snap What)
- Snap options 2 (Snap To What)

Outputs

- A polygon feature class - the vertices from the source layer will be moved to snap to the features of the Snap Layer (if within the snap tolerance)

Options:

- Snap Options 1 (Snap What) - this options lets the user set which elements of the source polygonsto be used for snapping
 - Vertices: All the vertices of the source polylines will be used.
 - Insert Vertices: This option will get the vertices from the features of the snap layer and will insert new vertices into the source polylines. The new vertices together with the original ones will be used for snapping. This option is slower than the other ones, but gives the best snapping results especially if the polygons to be snapped have much less vertices than the ones from the Snap layer.
- Snap Options 2 (Snap To What)
 - Vertices: The polygons will be snapped to the nearest vertex of the nearest feature from the Snap layer

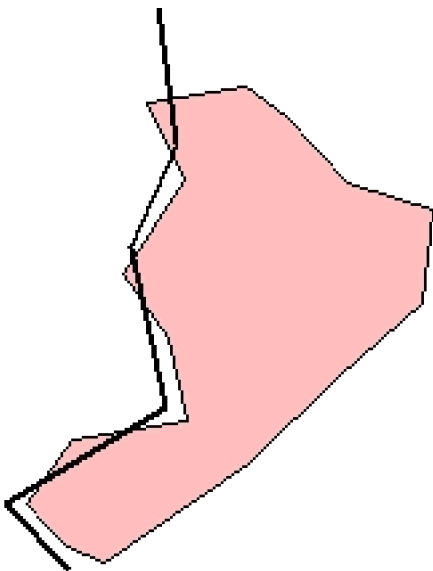
- Nearest edge: The polygons will be snapped to the nearest point of the nearest feature from the Snap layer
- Vertices and Edges: If there is a vertex closer than the snap tolerance to the polygons (their elements defined in Options 1) to be snapped, the polygon will snap to it, otherwise it will snap to the nearest edge.

Notes:

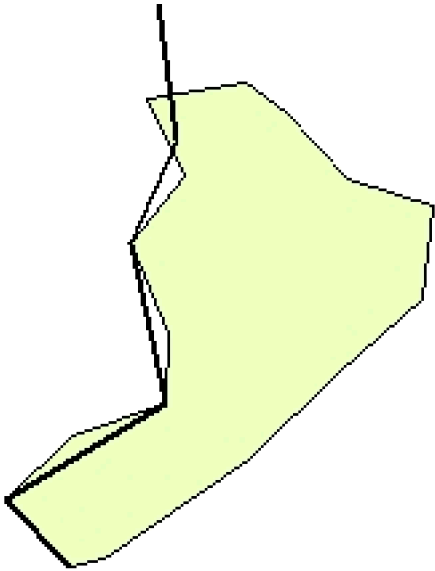
- The polygons will be cleaned from overlaps. It is recommended that the Clean Polygons function to be used before the Snap Polygons to ensure a better control over the cleaning process.
- The snap distance should be in the units of the spatial reference of the input dataset.
- The Source and the Reference Datasets can have different spatial references as long as they have the same Geographic Coordinate systems.

Example:

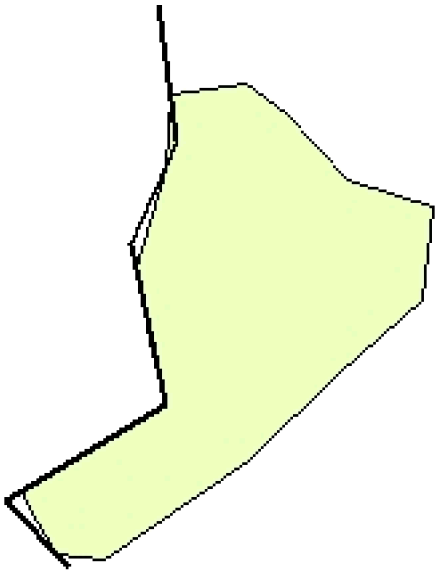
Before Snap



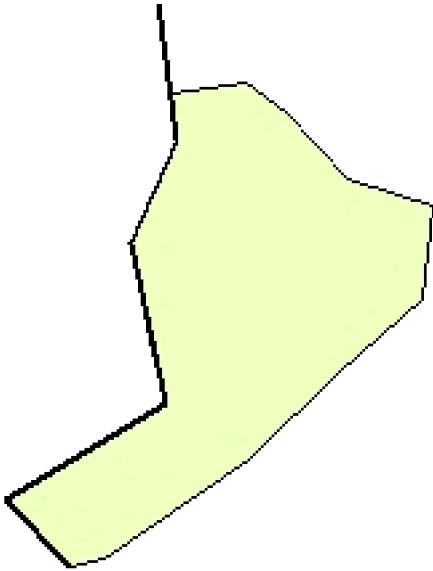
After Snap - Option: Vertices



After Snap - Option: Nearest Edge



After Snap - Option: Vertices and Edges



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SnapPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<Reference Dataset>	A String representing the layer to be used to snap to.
<output dataset>	A String - the full name of the output layer.
<Snap Tolerance>	A Double representing the Snap Tolerance (in the units of the spatial reference of the input dataset).
<Snap What>	<p>Required. A String indicating what parts of the input polygons will be snapped. Possible values:</p> <ul style="list-style-type: none">• Vertex - the vertices of the source polygons will be snapped.• InsertVertex - the vertices from the features of the Reference Dataset

	<p>will be inserted (if closer to the input dataset) as new vertices into the source polygon boundaries. The new vertices together with the original ones will be used for snapping. This option is slower than the other ones, but gives the best snapping results especially if the polygons to be snapped have much less vertices than the ones from the Reference Dataset.</p>
<Snap To What>	<p>Required. A String indicating to what parts of the reference geometries the input polygons will try to snap. Possible values:</p> <ul style="list-style-type: none"> • Vertex - the input polygons will be snapped only to the vertices of the geometries from the reference dataset. • All - the input polygons will be snapped to the vertices or nearest edge of the geometries from the reference dataset.
{Snap To Z}	<p>Optional. A Boolean indicating whether the input geometries will snap the the Z values of the geometries from the Reference Dataset. Only if both dataset have Z values.</p>

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<pre>subprocess.call([ETGWPath, "SnapPolygons", "input dataset", "Reference Dataset", "output dataset", "Snap Tolerance", "Snap What", "Snap To What", "Snap To Z"])</pre>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SnapPolygons" "input dataset" "Reference Dataset" "output dataset" "Snap Tolerance" "Snap What" "Snap To What" "Snap To Z"</pre>
.NET using ETGWOuX.dll	<pre>SnapPolygons(input dataset, Reference Dataset, output dataset, Snap Tolerance, Snap What, Snap To What, Snap To Z)</pre>
ArcPy	<pre>arcpy.SnapPolygons(input dataset, Reference Dataset, "output dataset", "Snap Tolerance", "Snap What", "Snap To What", "Snap To Z")</pre>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Get Adjacent Polygons

[Running programmatically](#)

Determines for each polygon of the dataset the adjacent polygons and stores the result in the attribute table as a comma delimited string.

Inputs:

- A polygon feature layer.
- Link Field - the field which values will be used to save in the adjacency string

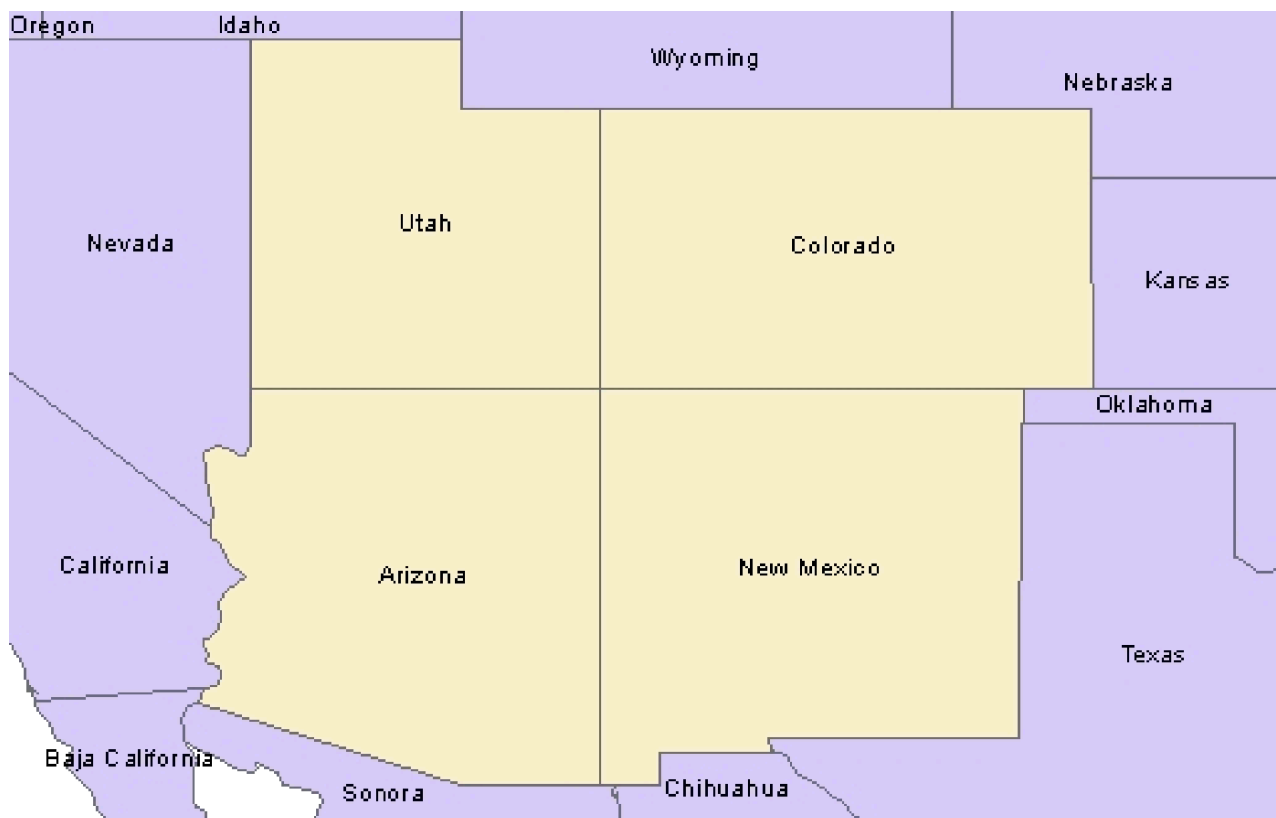
Outputs:

- New polygon feature class. Two fields will be added to the attribute table
 - [ET_Adj] - the field that will contain the adjacency string.
 - [ET_Count] - the count of the adjacent polygons for each polygon

Notes:

- If the input polygon layer has overlaps it is strongly recommended to use the [Clean Polygons](#) function beforehand
- A polygon is considered adjacent to another polygon only if the two polygons have a common boundary. Two polygons that share only a common point are not considered adjacent

Example:



State Name	ET Adj	ET Count
Arizona	California, Sonora, Nevada, NewMexico, BajaCalifornia, Utah	6
Colorado	NewMexico, Utah, Wyoming, Kansas,Nebraska, Oklahoma	6
New Mexico	Chihuahua, Sonora, Arizona, Colorado, Oklahoma, Texas	6
Utah	Nevada, Arizona, Colorado, Wyoming, Idaho	5

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonAdjacency
<input dataset>	A String representing the input layer. Must be of Polygon type.

<output dataset>	A String - the full name of the output layer.
{Link Field}	A Double representing the Generalization tolerance (in the units of the spatial reference of the input layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolygonAdjacency", "input dataset", "output dataset", "Link Field"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonAdjacency" "input dataset" "output dataset" "Link Field"</pre>
.NET using ETGWOuX.dll	<code>PolygonAdjacency(input dataset, output dataset, Link Field)</code>
ArcPy	<code>arcpy.PolygonAdjacency(input dataset, output dataset, "Link Field")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Partition Polygons with Polylines

[Running programmatically](#)

Partitions (splits) a polygon dataset with the polylines of a polyline dataset.

Inputs:

- A polygon feature class
- A polyline feature class to be used for splitting

Outputs:

- New polygon feature class. The attributes of the input data set are preserved.

Notes:

- Both datasets should have the spatial reference with the same Geographic Coordinate System.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PartitionPolygonsWithPolylines
<input dataset>	A String representing the input layer. Must be of Polygon type.
<Split Dataset>	A String representing the layer to be merged. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PartitionPolygonsWithPolylines", "input dataset", "Split Dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PartitionPolygonsWithPolylines" "input dataset" "Split Dataset" "output dataset"</code>
.NET using ETGWOuX.dll	<code>PartitionPolygonsWithPolylines(input dataset, Split Dataset, output dataset)</code>
ArcPy	<code>arcpy.PartitionPolygonsWithPolylines(input dataset, Split Dataset, "output dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Aggregate Polygons

[Running programmatically](#)

Combines the polygons from the input layer that are within the user specified distance into new polygons. Can be used also to generalize buildings.

Inputs:

- A polygon layer
- Aggregation distance. The polygons that are closer to each other than this distance will be combined
- Minimum area of the holes to be preserved - all holes with area less than this tolerance will be removed.

Outputs:

- New polygon layer

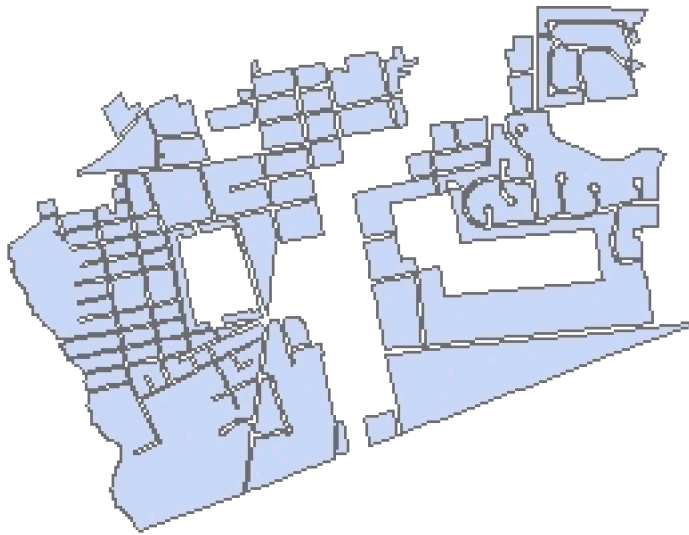
Notes:

- The new layer will not have any attributes. The [Transfer Attributes](#) function can be used to get summarized attributes from the original polygons.
- The Aggregation distance and the Minimum area of holes should be specified in the units of the spatial reference of the input layer
- If no Minimum area of holes is specified only the holes with area smaller than $2 \times \text{Aggregation distance} \times \text{Aggregation distance}$ will be removed

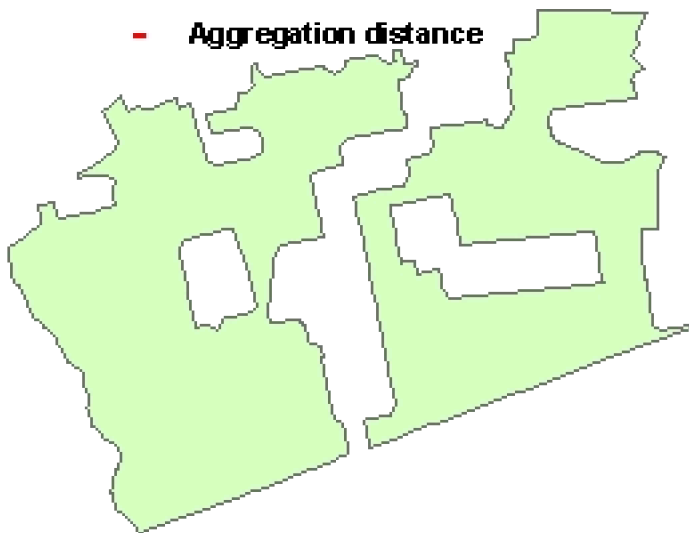
Examples:

General polygons

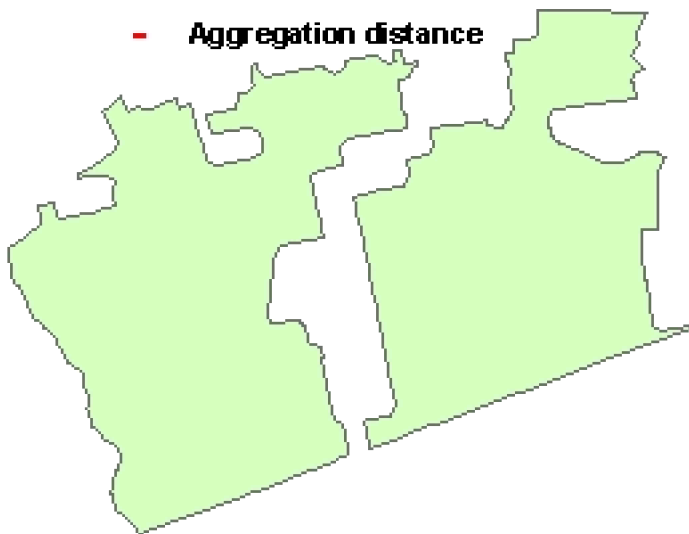
Input Dataset



Result (No Minimum area of holes specified)



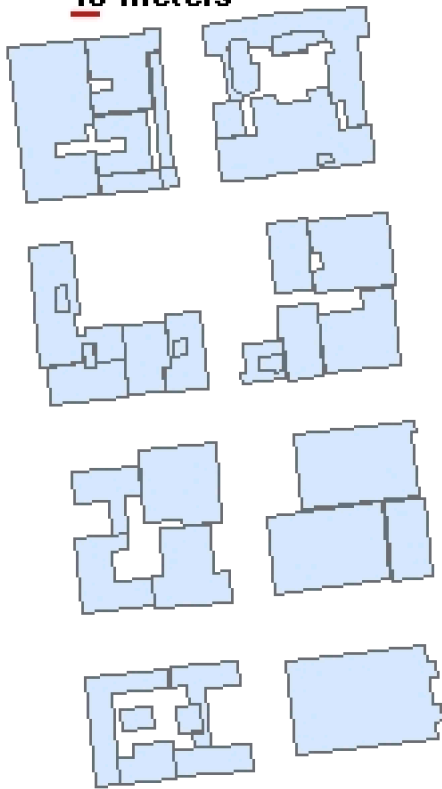
Result (Minimum area of holes specified)



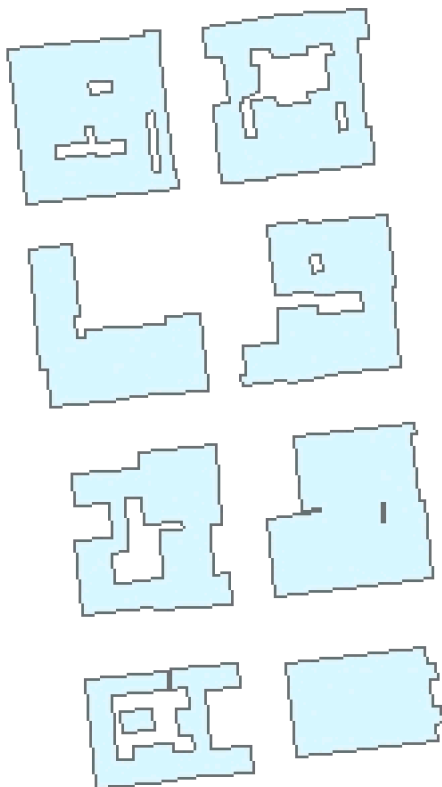
Buildings

Input Dataset

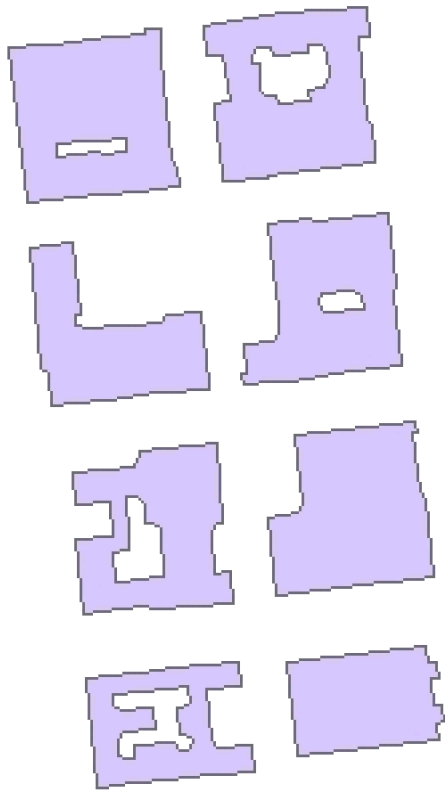
10 meters



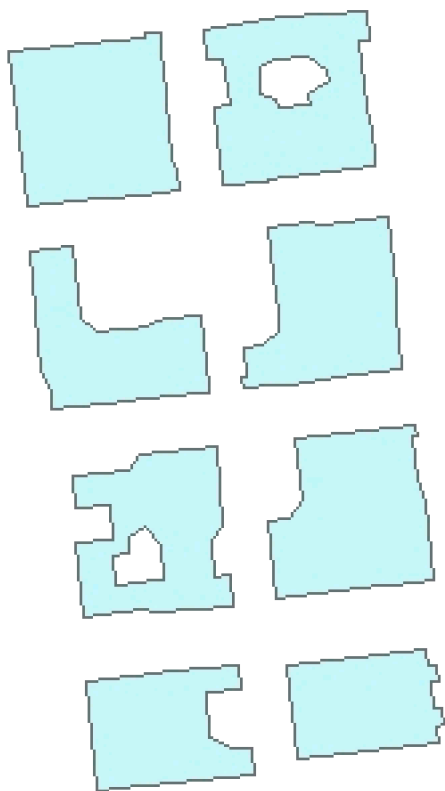
Aggregate distance = 1 meter



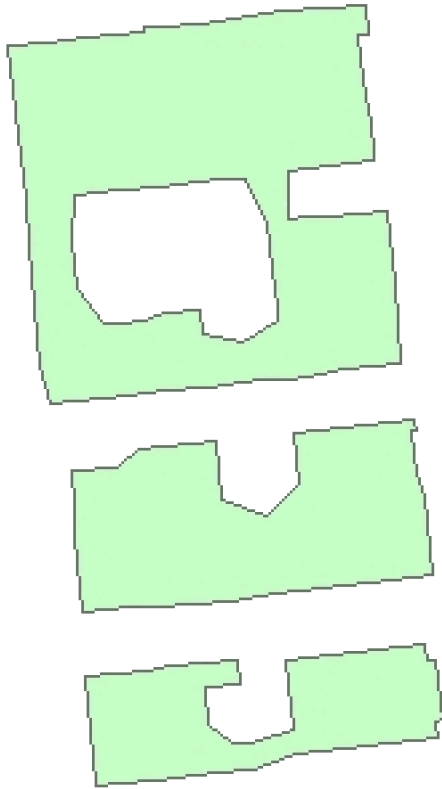
Aggregate distance = 5 meters



Aggregate distance = 10 m



Aggregate distance = 20 m



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	AggregatePolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Aggregation Distance>	A Double representing the aggregation distance. The polygons that are closer to each other than this distance will be combined.
{Area Tolerance}	A Double representing the minimum area of holes to be preserved. All holes with area less than this tolerance will be removed.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "AggregatePolygons", "input dataset", "output dataset", "Aggregation Distance", "Area Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "AggregatePolygons" "input dataset" "output dataset" "Aggregation Distance" "Area Tolerance"</code>
.NET using ETGWOutX.dll	<code>AggregatePolygons(input dataset, output dataset, Aggregation Distance, Area Tolerance)</code>
ArcPy	<code>arcpy.AggregatePolygons(input dataset, output dataset, "Aggregation Distance", "Area Tolerance")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygon To Polyline Advanced

[Running programmatically](#)

Converts the polygon boundaries to polylines.

- Creates a topologically correct (nodes at intersections, no overlaps) Polyline dataset
- For each polyline the left and right polygon attributes are added.
- Optionally the labels of the polygons are exported as points. The point attribute table contains all original attributes.

Inputs:

- A polygon feature layer
- Link Field - the values of this field will be saved as Left and Right polygons for each polyline
- Fuzzy tolerance - will be used to clean the polygon boundaries

Outputs:

- New topologically correct Polyline layer. Fields added to the polyline attribute table
 - [ET_Left] - stores the Left polygon link values
 - [ET_Right] - stores the Right polygon link values
- Optional Point layer representing the labels of the input polygons. The attributes of the input polygons are preserved in the Point Attribute Table
- Optional Polygon layer - the input layer clean from overlaps.

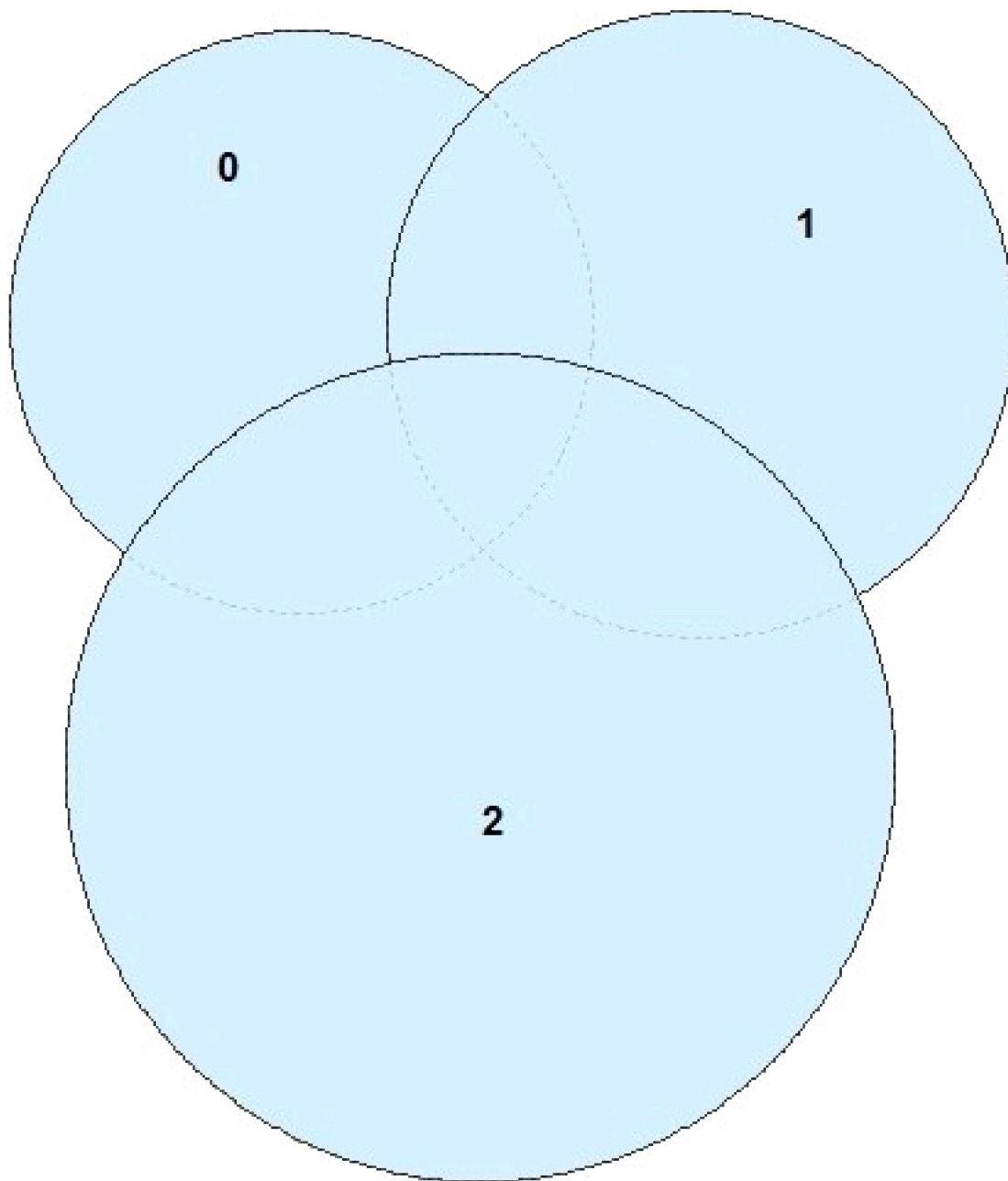
Notes :

- The default Fuzzy tolerance is calculated from the extents of the input layer using the formulae $(W + H) / 2000000$ where W and H are the width and height of the extent envelope.

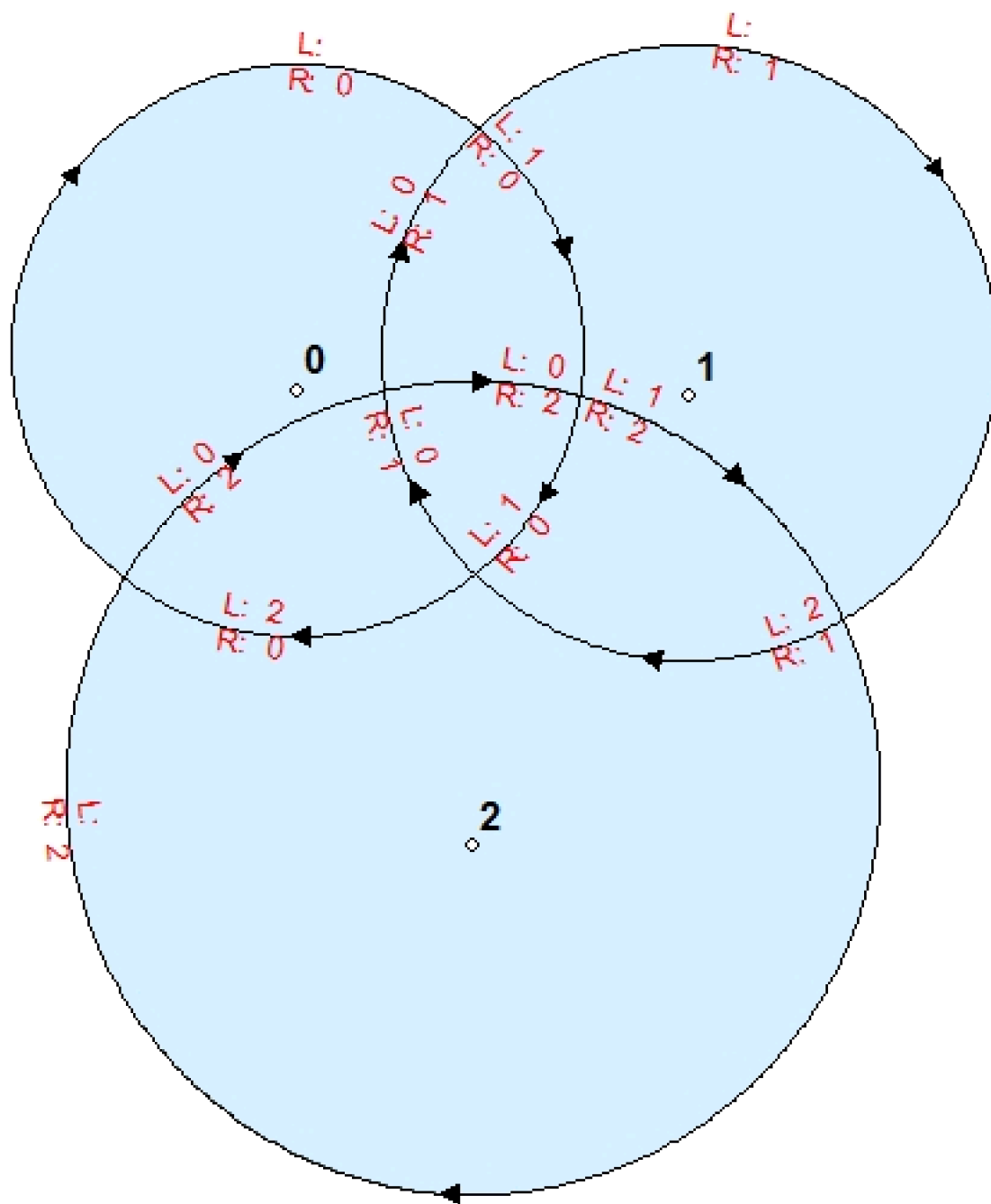
- Larger values of the Fuzzy tolerance may be used to clean some bigger Gaps and Slivers, but it might lead to unwanted approximation of the input shapes.
- A Fuzzy tolerance = 0 will be replaced by the default value
- If a polyline does not have Left polygon, the value of the ET_Left field will be set to empty string. In topologically correct polygon dataset this should indicate the outer boundary (neighboring with the so-called Universal Polygon. Empty values in the interior of the polygon dataset will indicate gaps or overlaps in the data.

If the input polygon layer has overlaps it is strongly recommended to use the [Clean Polygons](#) function beforehand

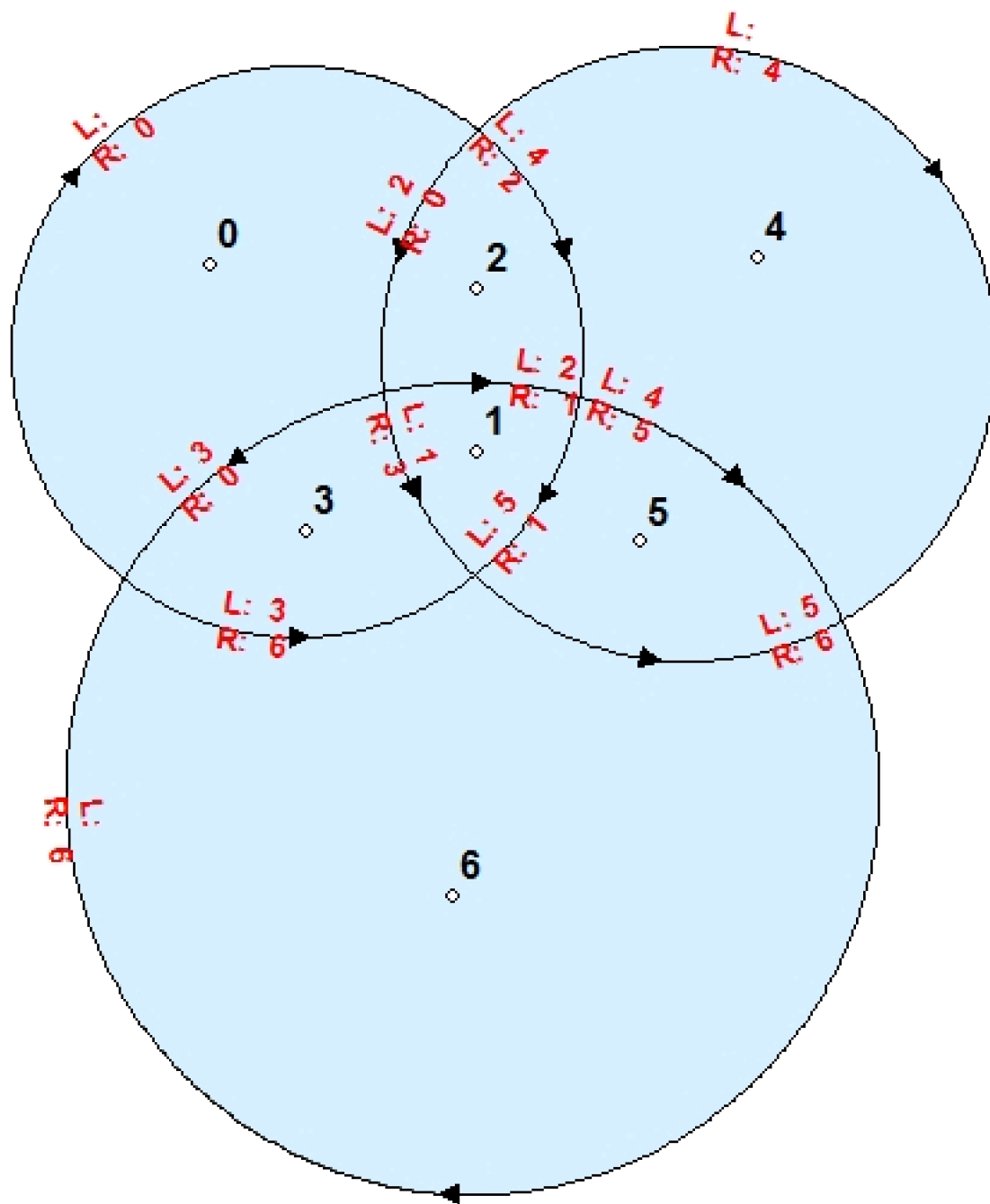
Input Polygons with overlaps



Result if the function is performed on the overlapping polygons

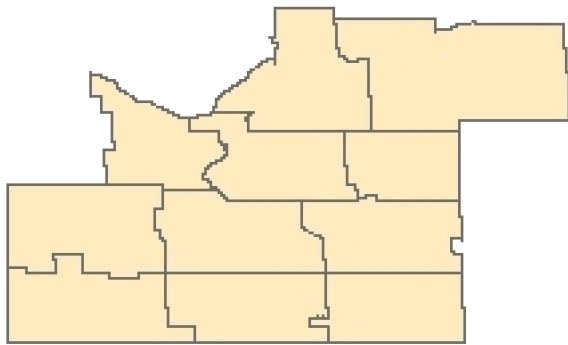


Result if the overlaps are cleaned beforehand



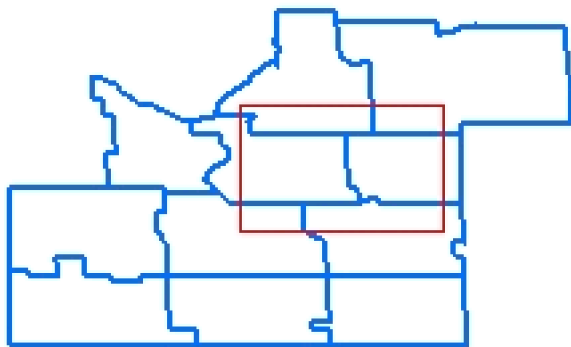
Example:

Original polygons

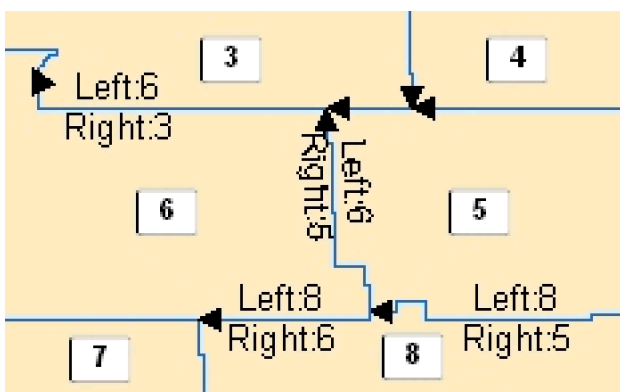


Derived polylines.

- Nodes in intersections
- No duplicates
- The polylines on the boundary of the area will have empty ET_Left



Polylines labeled with their Left and Right polygons. Polygons labeled with the Link field used in the function.



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToPolylinesAdvanced
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Fuzzy Tolerance>	A Double representing the Fuzzy Tolerance.
{Labels Dataset}	A String - the full name of the output label points layer.
{Link Field}	A String - the name of the field in the input dataset to be used as a link.

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call ([ETGWPPath, "PolygonsToPolylinesAdvanced", "input dataset", "output dataset", "Fuzzy Tolerance", "Labels Dataset" "Link Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPPath StartInfo.Arguments = "PolygonsToPolylinesAdvanced" "input dataset" "output dataset" "Fuzzy Tolerance" "Labels Dataset" "Link Field"
.NET using ETGWOutX.dll	PolygonsToPolylinesAdvanced (input dataset, output dataset, Fuzzy Tolerance, Labels Dataset,Link Field)
ArcPy	arcpy. PolygonsToPolylinesAdvanced (input dataset, output dataset, "Fuzzy Tolerance", "Labels Dataset", "Link Field")

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Generalize Polygons

[Running programmatically](#)

Generalizes (reduces the number of vertices required to represent a polygon) the features of a polygon layer using the Douglas-Poiker algorithm. Preserves the polygon topology. If the input polygon dataset has only stand alone polygons (no common boundaries), the Stand Alone option can be used for faster processing.

Inputs:

- A polygon feature class
- Generalization Tolerance (maximum offset) - the maximum distance that the generalized polyline will differ from the original one
- Option to treat the input as Stand Alone Polygons. Note that if this option is set to TRUE and there are polygons that share common boundaries, the topology might be destroyed.

Outputs:

- New polygon feature class
 - The output feature class will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Notes:

- **Make sure that the input polygon dataset does not have any overlaps!**
- The Generalization tolerance should be specified in the units of the spatial reference of the input feature class

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	GeneralizePolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Generalize Tolerance>	A Double representing the Generalization tolerance (in the units of the spatial reference of the input layer).
{Stand Alone Only}	A Boolean indicating whether the input contains only stand alone polygons.

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "GeneralizePolygons", "input dataset", "output dataset", "Generalize Tolerance", "Stand Alone Only"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "GeneralizePolygons" "input dataset" "output dataset" "Generalize Tolerance" "Stand Alone Only"</pre>
.NET using ETGWOutX.dll	<code>GeneralizePolygons(input dataset, output dataset, Generalize Tolerance, Stand Alone Only)</code>
ArcPy	<code>arcpy.GeneralizePolygons(input dataset, output dataset, "Generalize Tolerance", "Stand Alone Only")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Densify Polygons

[Running programmatically](#)

Densifies (adds vertices to polygons boundaries at a user-specified tolerance) the features of a polygon layer. Preserves the polygon topology. If the input polygon dataset has only stand alone polygons (no common boundaries), the Stand Alone option can be used for faster processing.

Inputs:

- A polygon feature class
- Maximum segment length
- Option to treat the input as Stand Alone Polygons. Note that if this option is set to TRUE and there are polygons that share common boundaries, the topology might be destroyed.

Outputs:

- New polygon feature class
 - The output feature class will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Notes:

- **Make sure that the input polygon dataset does not have any overlaps!**
- The Densify tolerance should be specified in the units of the spatial reference of the input feature class

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	DensifyPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Densify Tolerance>	A Double representing the Generalization tolerance (in the units of the spatial reference of the input layer).
{Stand Alone Only}	A Boolean indicating whether the input contains only stand alone polygons.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "DensifyPolygons", "input dataset", "output dataset", "Densify Tolerance", "Stand Alone Only"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "DensifyPolygons" "input dataset" "output dataset" "Densify Tolerance" "Stand Alone Only"</pre>
.NET using ETGWOuX.dll	<code>DensifyPolygons(input dataset, output dataset, Densify Tolerance, Stand Alone Only)</code>
ArcPy	<code>arcpy.DensifyPolygons(input dataset, output dataset, "Densify Tolerance", "Stand Alone Only")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Smooth Polygons

[Running programmatically](#)

Smooth the features of a polygon layer using three different smoothing algorithms. The polygons are cleaned from overlaps before proceeding with smoothing.

Inputs:

- A polygon feature layer
- Smooth method
 - Bezier curve
 - The curve in general does not pass through any of the control points (vertices of original polygon) except the first and last.
 - The curve is always contained within the convex hull of the control points
 - Approximate the original shape rather freely
 - Fast - good for polygons with many vertices (control points) that will constrain the curve close to the original shape
 - B - Spline
 - The curve does not pass through any of the control points (vertices of original polygon) except the first and last
 - Follows better than the Bezier curve the original shape
 - Depending on the "Freedom" parameter the smoothing occurs only in the areas close to a vertex
 - B-Splines lie in the convex hull of the original polygon
 - Slower than the Bezier curve, but the results in many cases are much better

- T - Spline (Tension Spline)
 - The curve passes through all the vertices of the original polygon
 - The degree of fit can be controlled with the "Tension" parameter
 - Suitable for smoothing curves with comparatively equally spaced vertices
 - Fast with good approximation of the original polygon
- Parameters depending on the method
 - The "Smoothness" parameter (Used in all methods) defines the number of points in the output curve. The allowed values (2 to 20) in fact are point multiplier. The number of vertices of the original polygon multiplied by this value will give the number of vertices of the smoothed polygon. The larger the value of the Smoothness parameter, the slower the process will be. In most of the cases a value of 5 (default) will create smooth and representative polygon
 - The "Freedom" parameter (B-Spline only) defines how close to the original polygon the curve will be. The allowed values are from 3 to 10. Smaller values give better approximation. With large values the curve will become very similar to Bezier curve
 - The "Tension" parameter (T-Spline only) defines how close to the original polygon the curve will be. Increasing the tension is similar to pulling on the ends of a string constrained to pass through the polygon vertices. allowed values are from 1 to 100.
- Optional - Densification tolerance. In some cases the smooth parameters cannot restrict the smoothing enough. The user can restrict the effect of the smoothing by introducing new vertices in the shapes. See [Densify](#) function for details
- Optional - Generalization tolerance. The smoothing introduces in the shapes many new vertices. The user can decrease the number of vertices by using this option. See [Generalize](#) function for details.

Outputs:

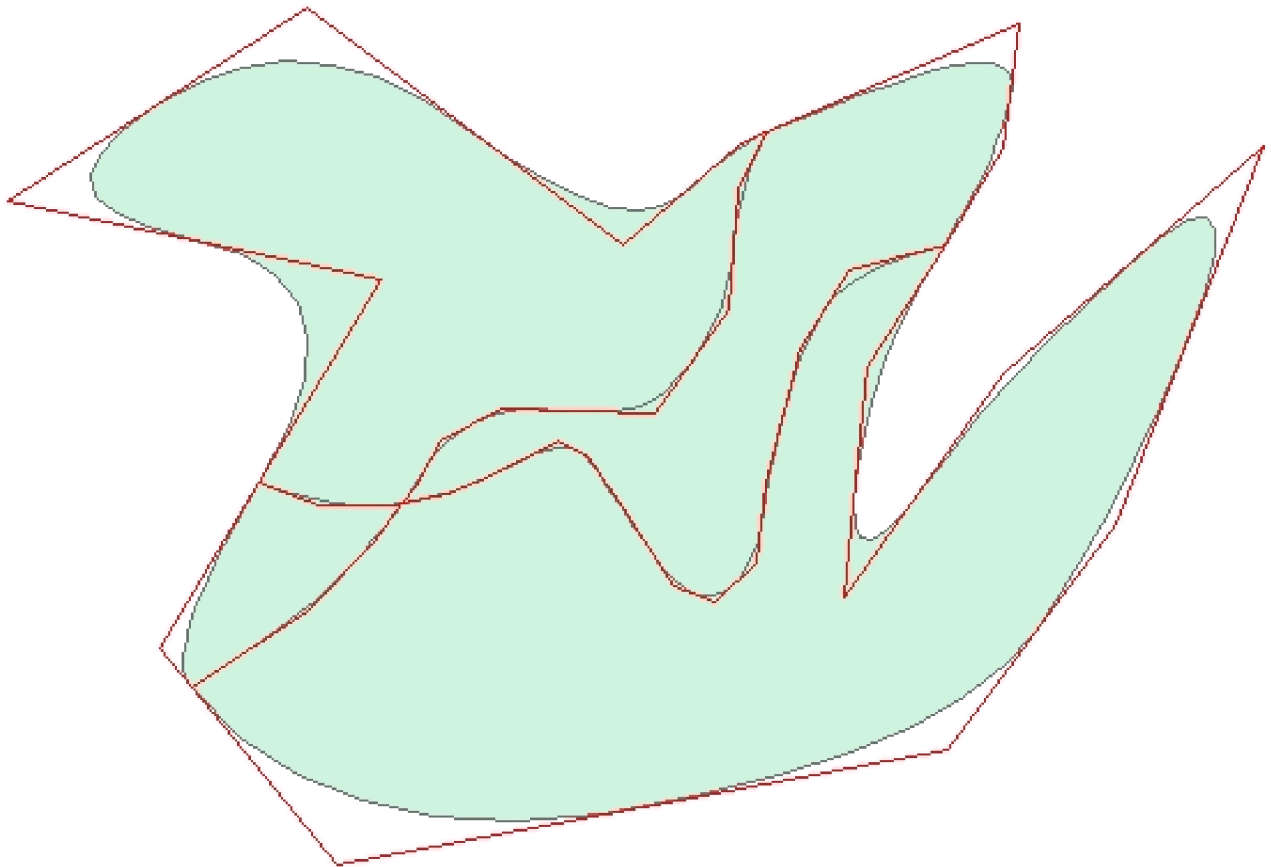
- New polygon layer
 - The output layer will contain all the features of the original data set
 - The attributes of the input data set are preserved.

Notes :

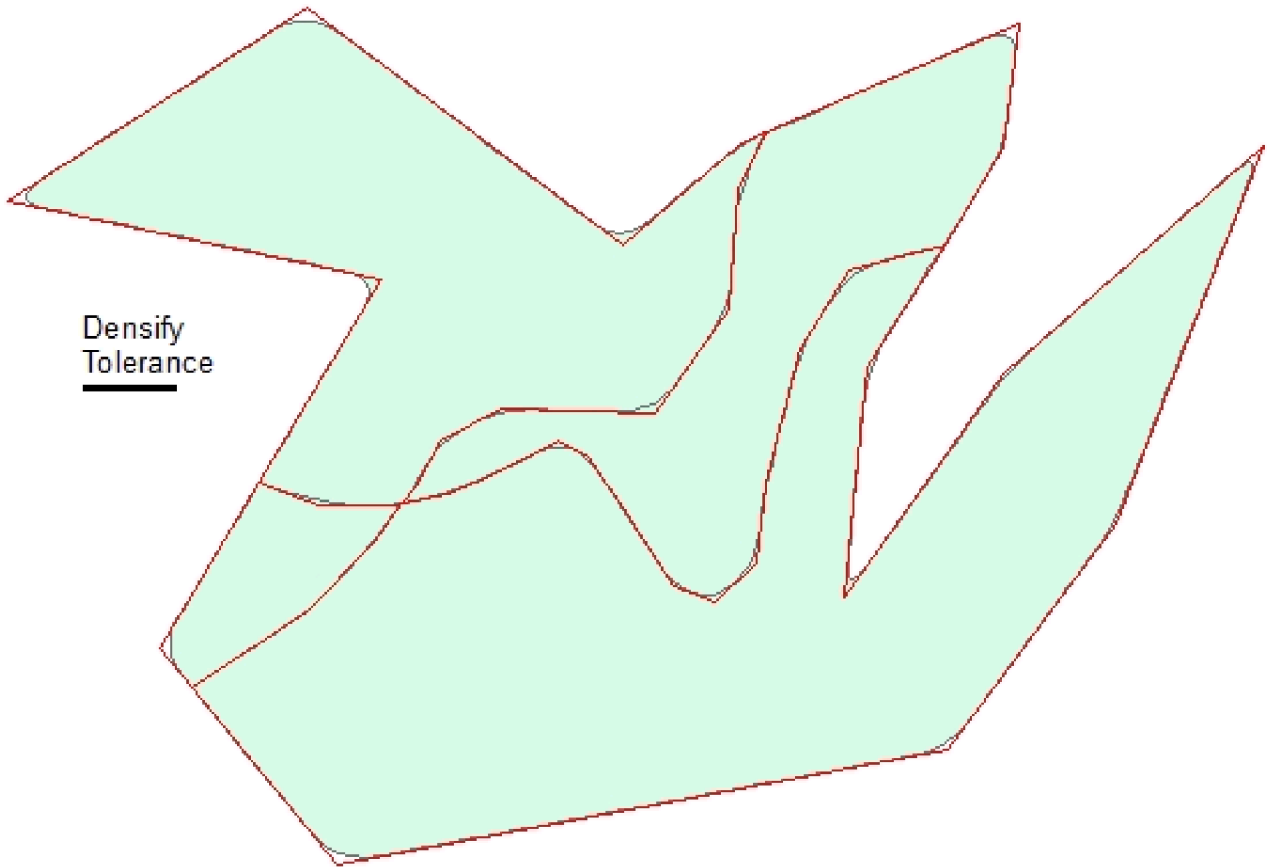
- It is highly recommended to use the Clean Polygons function before smoothing the polygons.
- All the methods implement generic algorithms.
- The Generalization and Densification tolerances should be specified in the units of the spatial reference of the input layer

Examples:

B-Spline - default values for smoothness and freedom



B-Spline - default values for smoothness and freedom. Densified before smoothing



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SmoothPolygonsBezier
Function Name	SmoothPolygonsBSpline
Function Name	SmoothPolygonsTSpline
<input dataset>	A String representing the input layer. Must be of polygon type.
<output dataset>	A String - the full name of the output layer.
<Smoothness>	An Integer that defines the number of points in the output curve. The allowed values (2 to 20) in fact are point multiplier. The number of vertices of the original polygon multiplied by this value will give the number of vertices of the smoothed polygon. The larger the value of the parameter, the slower the process will be.
<Freedom>	Only for B-Spline. An Integer that defines how close to the original polygon the curve will be. The allowed values are from 3 to 10. Smaller values give better approximation. With large values the curve will become very similar to Bezier curve
<Tension>	Only for T-Spline. An Integer that defines how close to the original polygon the curve will be. Increasing the tension is similar to pulling on the ends of a string constrained to pass through the polygon vertices. allowed values are from 1 to 100.
{Densify Before}	A Double representing the Densification tolerance. In some cases the smooth parameters cannot restrict the smoothing enough. The user can restrict the effect of the smoothing by introducing new vertices in the shapes. See Densify function for details
{Generalize After}	A Double representing the Generalization tolerance. The smoothing introduces in the shapes many new vertices. The user can decrease the

number of vertices by using this option. See Generalize function for details.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

SmoothPolygonsBezier

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SmoothPolygonsBezier", "input dataset", "output dataset", "Smoothness", "Densify Before", "Generalize After"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SmoothPolygonsBezier" "input dataset" "output dataset" "Smoothness" "Densify Before" "Generalize After"</code>
.NET using ETGWOutX.dll	<code>SmoothPolygonsBezier(input dataset, output dataset, Smoothness, Densify Before, Generalize After)</code>
ArcPy	<code>arcpy.SmoothPolygonsBezier(input dataset, output dataset, "Smoothness", "Densify Before", "Generalize After")</code>

SmoothPolygonsBSpline

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SmoothPolygonsBSpline", "input dataset", "output dataset", "Smoothness", "Freedom", "Densify Before", "Generalize After"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SmoothPolygonsBSpline" "input dataset" "output dataset" "Smoothness" "Freedom" "Densify Before" "Generalize After"</code>
.NET using ETGWOutX.dll	<code>SmoothPolygonsBSpline(input dataset, output dataset, Smoothness, Freedom, Densify Before, Generalize After)</code>
ArcPy	<code>arcpy.SmoothPolygonsBSpline(input dataset, output dataset, "Smoothness", "Freedom", "Densify Before", "Generalize After")</code>

SmoothPolygonsTSpline

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "SmoothPolygonsTSpline", "input dataset", "output dataset", "Smoothness", "Tension", "Densify Before", "Generalize After"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "SmoothPolygonsTSpline" "input dataset" "output dataset" "Smoothness" "Tension" "Densify Before" "Generalize After"</code>
.NET using ETGWOutX.dll	<code>SmoothPolygonsTSpline(input dataset, output dataset, Smoothness, Tension, Densify Before, Generalize After)</code>
ArcPy	<code>arcpy.SmoothPolygonsTSpline(input dataset, output dataset, "Smoothness", "Tension", "Densify Before", "Generalize After")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygon Characteristics

[Running programmatically](#)

Calculates some characteristics of the polygons from a polygon dataset

Inputs:

- A Polygon feature class

Outputs:

- A new Polygon dataset
- All original attributes are preserved.
- New fields added to the attribute table
 - ET_Length - the length of the longest axis in the units of the Spatial Reference of the input feature class.
 - ET Width - the length of shortest side of the bounding rectangle aligned with the longest axis in the units of the Spatial Reference of the input feature class.
 - ET Circ - Circularity ratio - for a circle the circularity will be 1. The thinner the polygon is the smaller the circularity will be.
 - ET Thick - Thickness ratio expressed as a ratio of the polygon area versus the area of its minimum bounding square. The ratio will have value of 1 for a square. The smaller the value is, the thinner the polygon is.
 - ET_Parts - the number of parts that the polygon has
 - ET_Holes - the number of holes in the polygon
 - ET_HasArcs - if the polygon has true arc segments - 1 otherwise - 0
 - ET_Vert - the number of vertices of the polygon
 - ET_Depth - the distance from the deepest point (the center of the maximum inscribed circle) to the polygon boundary. See [Polygon To Maximum Inscribed](#)

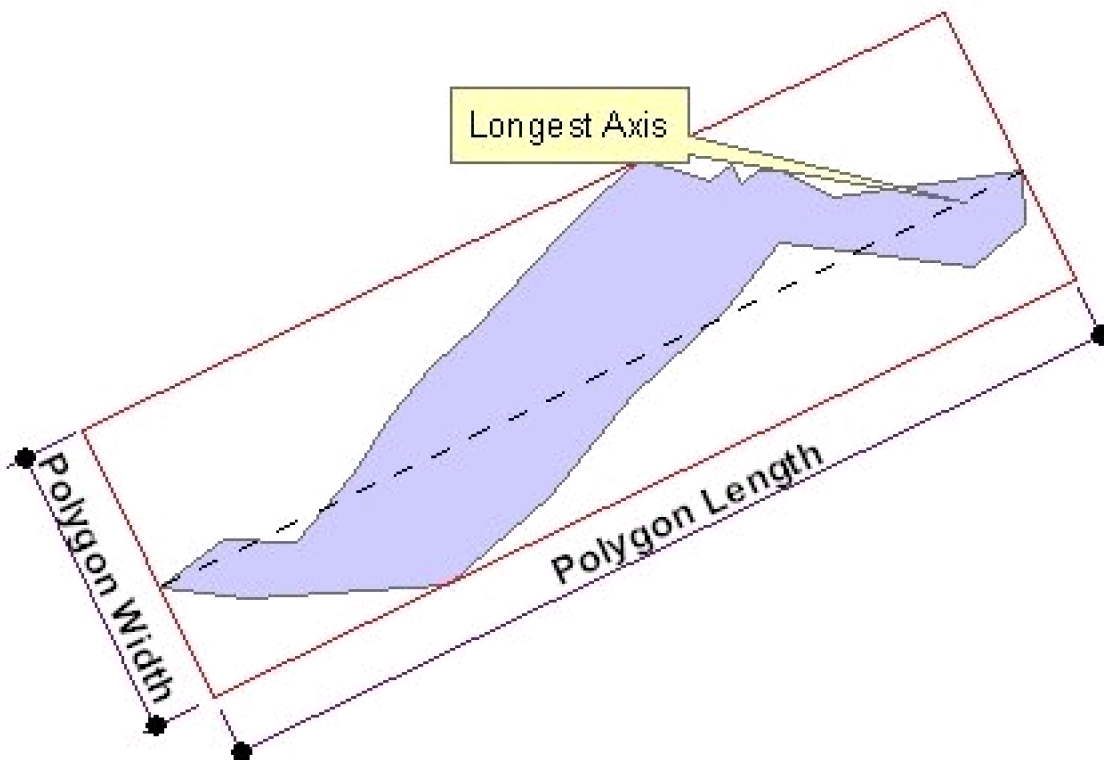
[Circle](#) function

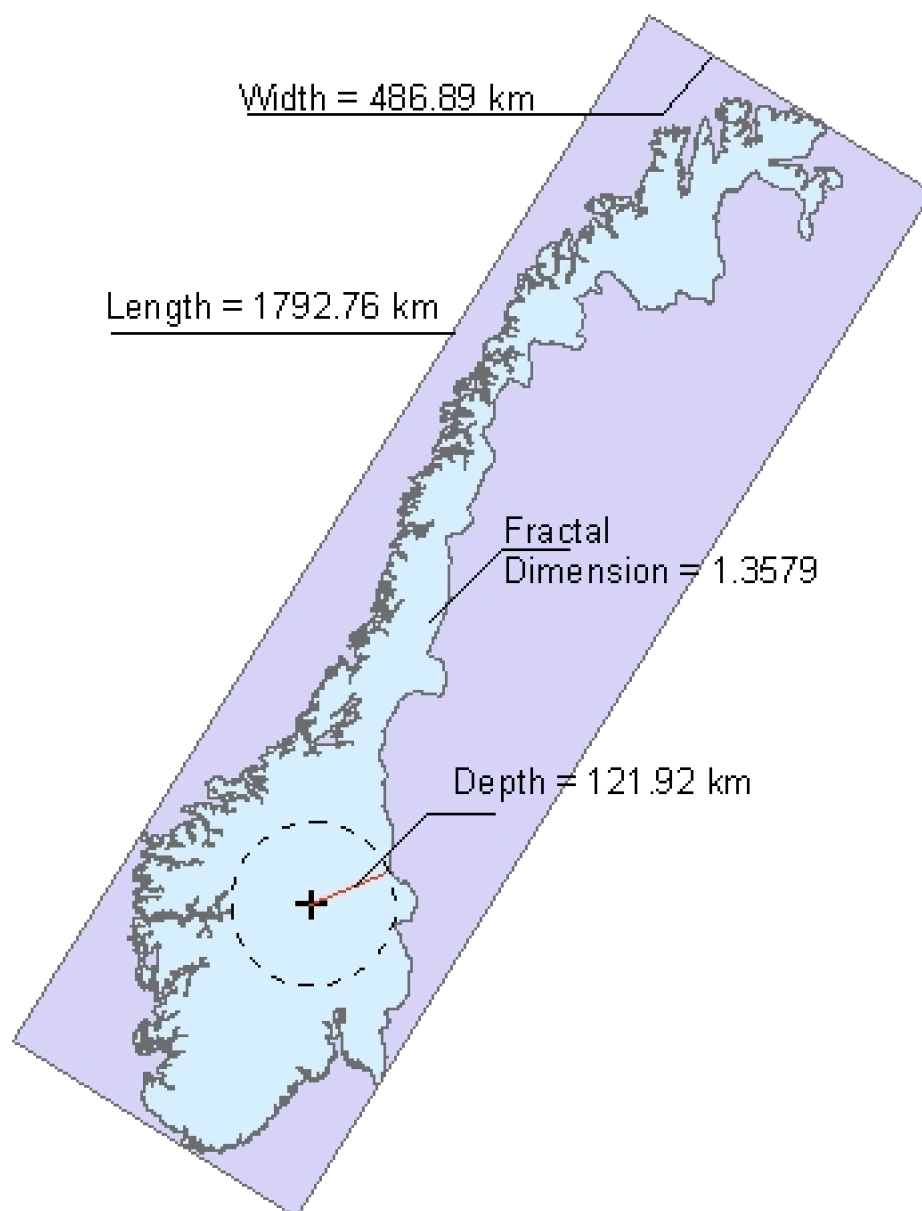
- ET_Fract - the fractal dimension (indication of the complexity) of the polygon boundary. The value is between 1 and 2. The more complex the polygon boundary is the larger the fractal dimension will be.

Notes:

- Fractal Dimension of the polygon boundaries is calculated using the Box Counting method (1)
- Calculating the Fractal Dimension and Polygon Depth is time consuming. If you don't need these characteristics, uncheck the options for faster processing.

Illustrations:





References:

1. Bourke, P., 1993. Fractal Dimension Calculator User Manual, Online. Available: <http://paulbourke.net/fractals/fracdim/>

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonCharacteristics

<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Calculate Depth}	A Boolean indicating whether to calculate polygon depth or not.
{Calculate Fractal}	A Boolean indicating whether to calculate fractal dimension or not.
{Precision}	An Integer between 0 and 8 representing the number of places after the decimal point to be used.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolygonCharacteristics", "input dataset", "output dataset", "Calculate Depth", "Calculate Fractal", "Precision"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonCharacteristics" "input dataset" "output dataset" "Calculate Depth" "Calculate Fractal" "Precision"</code>
.NET using ETGWOuX.dll	<code>PolygonCharacteristics(input dataset, output dataset, Calculate Depth, Calculate Fractal, Precision)</code>
ArcPy	<code>arcpy.PolygonCharacteristics(input dataset, output dataset, "Calculate Depth", "Calculate Fractal", "Precision")</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Fill Polygon Holes

[Running programmatically](#)

Fills the holes in a polygon dataset.

Inputs:

- A polygon feature layer
- Maximum area of the holes to be removed

Outputs:

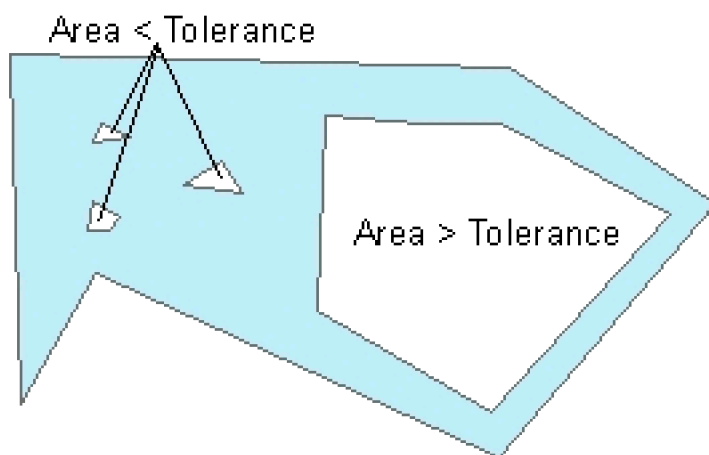
- New polygon dataset. The holes with area smaller than the user tolerance will be removed

Notes :

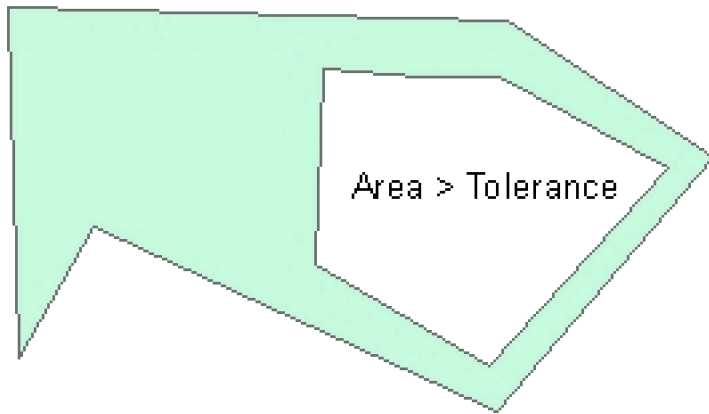
- If there are island polygons present, the function will create overlaps. Use the Clean Polygons function to restore the topology.

Example:

Input Dataset



After Fill Polygon Holes



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FillPolygonHoles
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Max Area}	A Double. Holes with larger area than this tolerance will not be removed.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "FillPolygonHoles", "input dataset", "output dataset", "Max Area"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "FillPolygonHoles" "input dataset" "output dataset" "Max Area"

.NET using ETGWOtX.dll	FillPolygonHoles(input dataset, output dataset,Max Area)
ArcPy	arcpy.FillPolygonHoles(input dataset, output dataset,Max Area)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygon To Polyline

[Running programmatically](#)

Converts a polygon data set to a polyline feature class

Inputs:

- A polygon feature layer

Outputs:

- New polyline layer

Notes :

- Each ring of a polygon will be represented by a single polyline. The common boundaries between the polygons will be represented by duplicate polylines. The [Clean Polylines Wizard](#) can be used to create intersections and remove duplicate lines.
- The attributes of the original polygons are transferred to the resulting polylines.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToPolylines
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.

{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.
----------	--

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolygonsToPolylines", "input dataset", "output dataset", "Keep Z", "Keep M"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonsToPolylines" "input dataset" "output dataset" "Keep Z" "Keep M"</pre>
.NET using ETGWOuX.dll	<code>PolygonsToPolylines(input dataset, output dataset, Keep Z, Keep M)</code>
ArcPy	<code>arcpy.PolygonsToPolylines(input dataset, output dataset, "Keep Z", "Keep M")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygons To Points

[Running programmatically](#)

Converts a polygon dataset to a point dataset

Inputs:

- A polygon feature layer
- Conversion option
 - Vertices - the vertices of all polygons will be converted to points.
 - Labels - the Label point is always located inside the polygon. The algorithm makes sure that the point is not close to the boundary of the polygon. Points created using this algorithm are suitable for spatial transfer of attributes (See Smooth Polygons and Generalize Polygons functions).
 - Centers - the Center points represent the centroid of a polygon. Therefore sometimes they might be located outside of the polygon
 - Centers Inside - points representing the centroids of the polygons. If the centroid occurs outside of the polygon, the point is moved to be in the polygon.
- More options
 - Remove Duplicate Points - the duplicate points created from the vertices of two adjacent polygons will be represented by one point. Note that if this option is used the attempt to convert back these points to polygons will produce incorrect result
 - Calculate point Position along boundaries
 - If used the [ET_Order] field will be populated with the relative location of the vertex (0 to 1) from the start of the polygon boundary.
 - If not used, the [ET_Order] field will be populated with the order of the vertex in the polygon ring (from 0 to number of vertices)
 - Preserve Z(M) available only if the input feature class is of PolygonZ(M) type. If

selected, the result will be of PointZ(M) type, otherwise the result will be of plain points (no Z or M values)

Outputs:

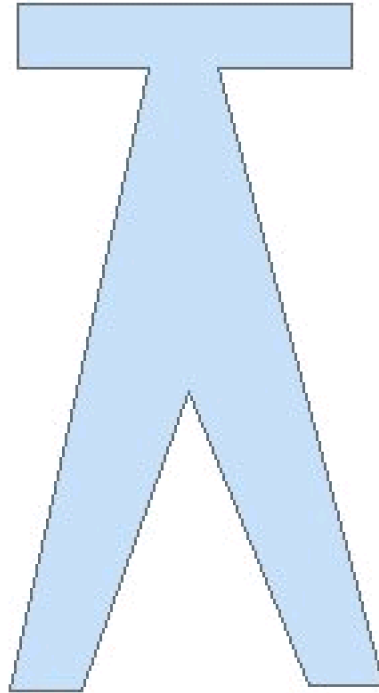
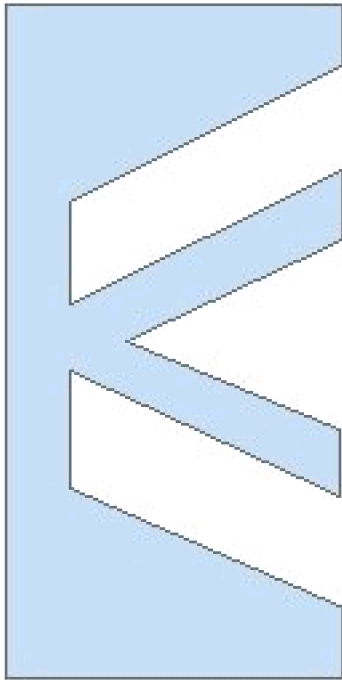
- New point layer
 - All the original attributes of the polygons are transferred to the point attribute table
 - New fields are added to the point attribute table
 - [ET_Order] - the position of the point along the polygon's boundary. The value can be from 0 to 1 (if the Calculate point Position option is used) or from 0 to number of vertices (if not). The value of this attribute can be used if the polygons have to be recreated from these points. - only if "Vertices" conversion option is used
 - [ET_IDP] - the FID of original polygons. The values can be used to link the points back to the polygons.
 - [ET_IDR] - this is a unique number identifying each ring of the polygons. If a polygon with FID = 356 has 3 rings, the corresponding points will have values in this fields 356_0, 356_1 and 356_2. This field can be used to recreate the polygons from the points without losing the rings. - only if "Vertices" conversion option is used
 - [ET_X] - the X coordinates of the resulting points
 - [ET_Y] - the Y coordinates of the resulting points
 - [ET_Z] - if the input feature class is of PolygonZ(M) type.
 - [ET_M] - if the input feature class is of PolygonZ(M) type.

Notes :

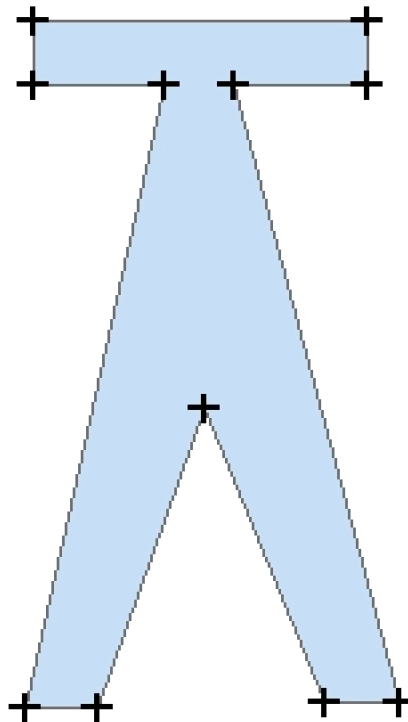
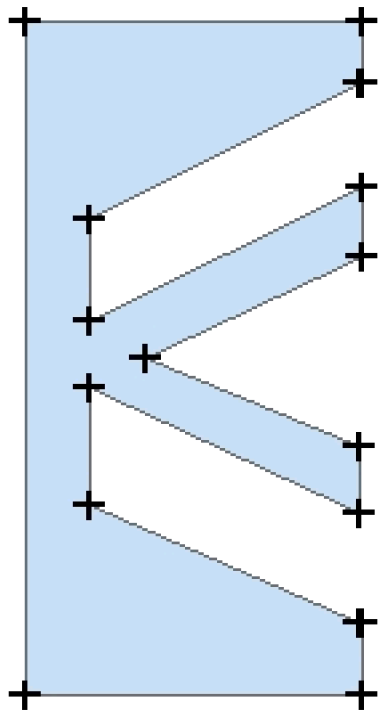
- See above for the use of the "Remove duplicate points" option

Examples:

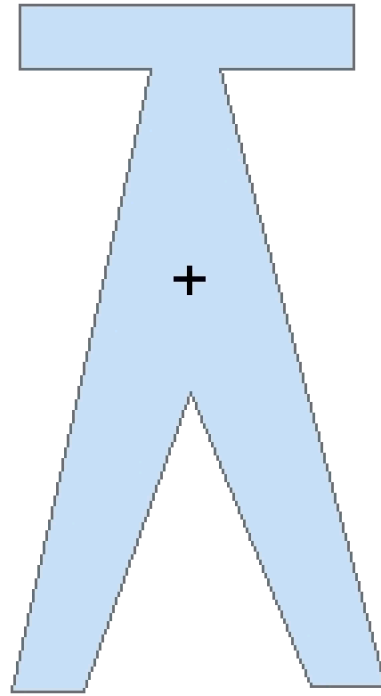
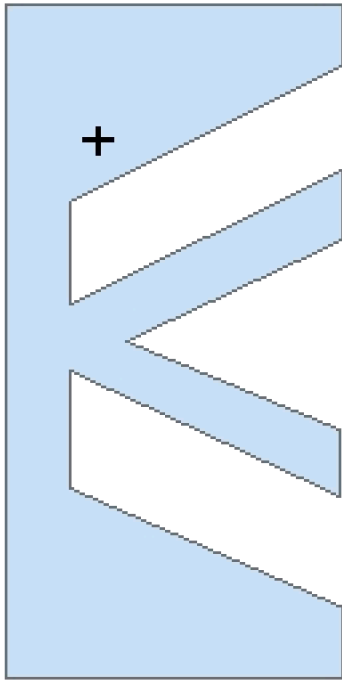
Input Dataset



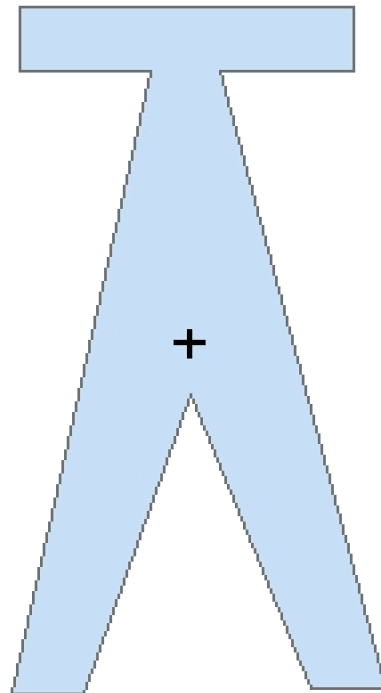
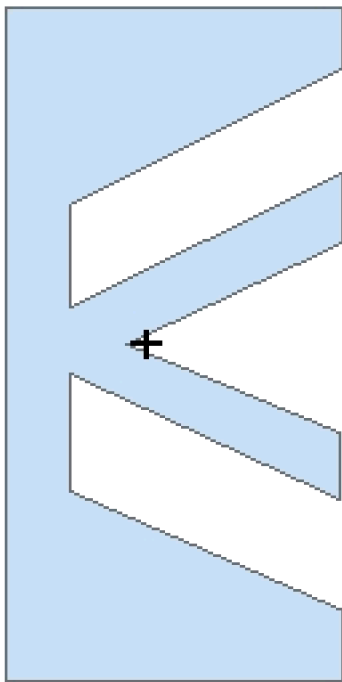
Result Vertices option



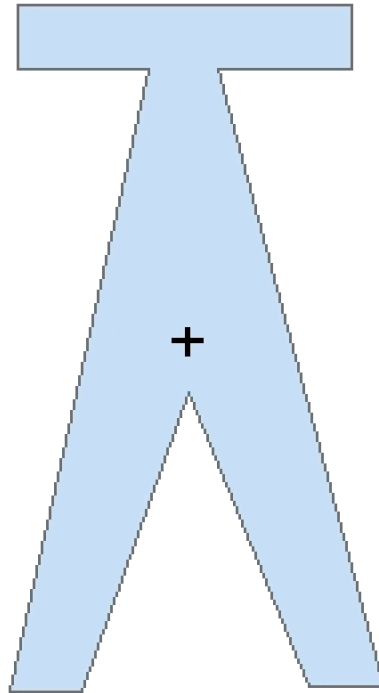
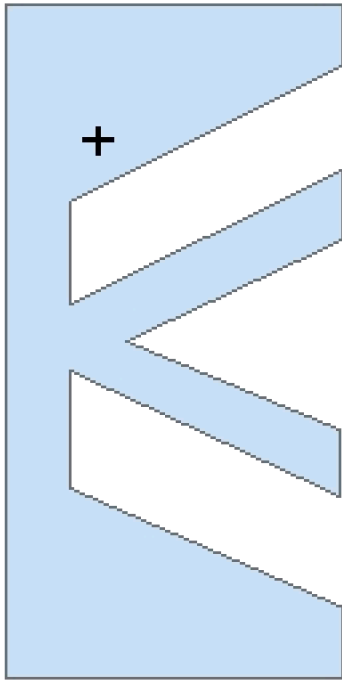
Result Labels option



Result Center option



Result Center in option



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToPoints
<input dataset>	A String representing the input layer. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
<Option>	<div>A String - the conversion option. Valid values.<ul style="list-style-type: none">• "Vertex"• "Center"• "Label"</div>

	<ul style="list-style-type: none"> • "CenterIn"
{Remove Duplicates}	A Boolean indicating whether the duplicate points are to be removed from the output. Used only for "Vertex" option.
{Calculate Position}	A Boolean indicating whether the position of the points along the polygon boundary to be calculated. Used only for "Vertex" option.
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.
{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PolygonsToPoints", "input dataset", "output dataset", "Option", "Remove Duplicates", "Calculate Position", "Keep Z", "Keep M"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonsToPoints" "input dataset" "output dataset" "Option" "Remove Duplicates" "Calculate Position" "Keep Z" "Keep M"
.NET using ETGWOuX.dll	PolygonsToPoints(input dataset, output dataset, Option, Remove Duplicates, Calculate Position, Keep Z, Keep M)
ArcPy	arcpy.PolygonsToPoints(input dataset, output dataset,"Option", "Remove Duplicates", "Calculate Position", "Keep Z", "Keep M")

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polylines To Points

[Running programmatically](#)

Converts a polyline data set to a point dataset

Inputs:

- A polyline feature layer
- Conversion option
 - Vertices - the vertices of all polylines will be converted to points. If the "Remove duplicate points" option is selected the duplicate points created from the nodes of two polylines sharing a common node will be represented by one point. Note that if this option is used the attempt to convert back these points to polylines will produce incorrect result.
 - Nodes - only the nodes of each polyline will be exported.
 - Middle points - only the middle point of each polyline will be exported.
- More options
 - Remove Duplicate Points - the duplicate points created from the vertices of two adjacent polygons will be represented by one point. Note that if this option is used the attempt to convert back these points to polygons will produce incorrect result
 - Calculate point Position along boundaries
 - If used the [ET_Order] field will be populated with the relative location of the vertex (0 to 1) from the start of the polylines.
 - If not used, the [ET_Order] field will be populated with the order of the vertex in the polyline (from 0 to number of vertices)
 - Preserve Z(M) available only if the input feature class is of PolygonZ(M) type. If selected, the result will be of PointZ(M) type, otherwise the result will be of plain points (no Z or M values)

Outputs:

- New point feature class
 - All the original attributes of the polylines are transferred to the point attribute table
 - New fields are added to the point attribute table
 - [ET_ID] - the FID of original polylines. The values can be used to link the points back to the polylines.
 - [ET_IDP] - this is a unique number identifying each part of the polylines. If a polyline with FID = 356 has 3 parts, the corresponding points will have values in this fields 356_0, 356_1 and 356_2.
 - [ET_X] - the X coordinates of the resulting points
 - [ET_Y] - the Y coordinates of the resulting points
 - If the conversion option is "Vertices" or "Nodes" an Order field is added
 - [ET_Order] - the position of the point along the polyline . The value can be from 0 to 1 (if the Calculate point Position option is used) or from 0 to number of vertices (if not). The value of this attribute can be used if the polyline have to be recreated from these points.
 - If the "Assign angle attribute" option is used an angle field is added
 - [ET_Angle] - the angle of the polyline in this point.

Notes :

- See above for the use of the "Remove duplicate points" option
- If the "Assign angle attribute" option is used, the points symbols can be rotated and in such a way can represent the direction of the polylines. See the example below

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolylinesToPoints
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Option>	<p>A String - the conversion option. Valid values.</p> <ul style="list-style-type: none">• "Vertex"• "Node"• "Middle"
{Calculate Angle}	A Boolean indicating whether the angle of the polyline at the points location to be calculated and stored in the attribute table. Used only for "Vertex" option.
{Remove Duplicates}	A Boolean indicating whether the duplicate points are to be removed from the output. Used only for "Vertex" option.
{Calculate Position}	A Boolean indicating whether the position of the points along the polygon boundary to be calculated. Used only for "Vertex" option.
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.
{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program

Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolylinesToPoints", "input dataset", "output dataset", "Option", "Calculate Angle", "Remove Duplicates", "Calculate Position", "Keep Z", "Keep M"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolylinesToPoints" "input dataset" "output dataset" "Option" "Calculate Angle" "Remove Duplicates" "Calculate Position" "Keep Z" "Keep M"</code>
.NET using ETGWOutX.dll	<code>PolylinesToPoints(input dataset, output dataset, Option, Calculate Angle, Remove Duplicates, Calculate Position, Keep Z, Keep M)</code>
ArcPy	<code>arcpy.PolylinesToPoints(input dataset, output dataset, "Option", "Calculate Angle", "Remove Duplicates", "Calculate Position", "Keep Z", "Keep M")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polylines To Polygons

[Running programmatically](#)

Converts closed polylines (and polyline chains) to polygons

Inputs:

- A polyline feature layer

Outputs:

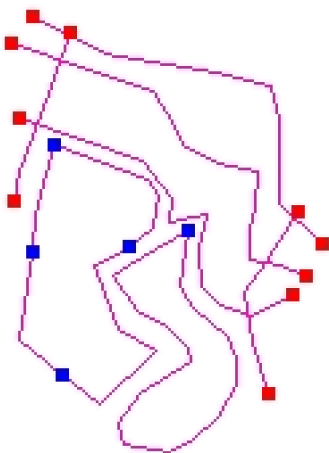
- New polygon layer

Notes :

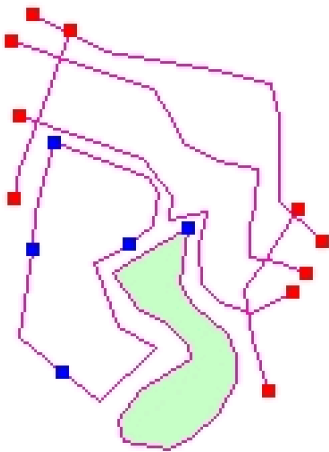
- All closed polylines will be converted to polygons.
- The attributes of the polylines will be transferred to the corresponding polygon features.
- If the "Use closed polyline chains" option is used, the polylines that form closed chains will be used to create polygons. The attributes of the first polyline of the chain will be transferred to the polygon feature
- For advanced polygon creation use the [Build Polygon function](#)

Example:

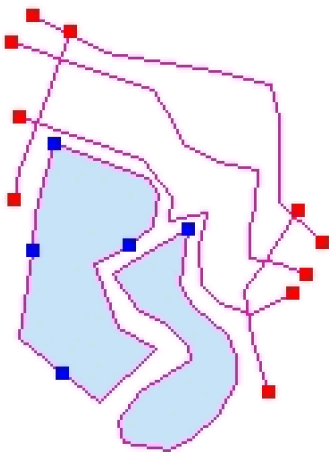
Source Polyline Layer



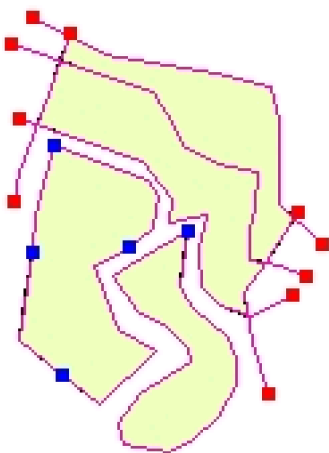
Result Polygon Layer



Result Polygon Layer(closed polyline chains used)



Result Polygon Layer (Build Polygons Function)



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolylinesToPolygons
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Force Closure}	A Boolean indicating whether polylines that are not closed and the distance between the from and to nodes is smaller than the tolerance specified to be closed.
{Close Tolerance}	A Double representing the distance to be used for closing the non closed polylines.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolylinesToPolygons", "input dataset", "output dataset", "Force Closure", "Close Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolylinesToPolygons" "input dataset" "output dataset" "Force Closure" "Close Tolerance"</code>
.NET using ETGWOuX.dll	<code>PolylinesToPolygons(input dataset, output dataset, Force Closure, Close Tolerance)</code>
ArcPy	<code>arcpy.PolylinesToPolygons(input dataset, output dataset, "Force Closure", "Close Tolerance")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polylines To Multipoints

[Running programmatically](#)

Converts a polyline data set to a Multipoint dataset

Inputs:

- A polyline feature layer

Outputs:

- New Multipoint feature class

Notes :

- The attributes of the original polylines are transferred to the resulting multipoints.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolylinesToMultipoints
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.
{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolylinesToMultipoints", "input dataset", "output dataset", "Keep Z", "Keep M"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolylinesToMultipoints" "input dataset" "output dataset" "Keep Z" "Keep M"</code>
.NET using ETGWOuX.dll	<code>PolylinesToMultipoints(input dataset, output dataset, Keep Z, Keep M)</code>
ArcPy	<code>arcpy.PolylinesToMultipoints(input dataset, output dataset, "Keep Z", "Keep M")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Polylines

[Running programmatically](#)

Converts a point data set to a polyline layer. Attaches to the polyline attribute table the values of the attributes for the first and last point that form a single polyline. If your point data does not have Polyline ID and Order attributes, you can try the [Connect Unstructured Points](#) function.

Inputs:

- A point feature layer
 - REQUIRED: an ID field which value defines the points to be used for creation of each polyline
 - OPTIONAL: an Order field that defines in what sequence the points describe the polyline. If no Order field is used the order is defined by the record number of the points
 - OPTIONAL: a Link field. The values for the first and last point that will form a single polyline will be added to the polyline attribute table.
 - OPTIONAL: Z Value field. If specified, a PolylineZ feature class will be created. The values in this field will be set as Z values for the vertices. If the input points have Z values, the user can specify the Z values of the input points to be used by selecting "Features" for Z Value field.
 - OPTIONAL: M Value field. If specified, a PolylineM feature class will be created. The values in this field will be set as M values for the vertices. If the input points have M values, the user can specify the M values of the input points to be used by selecting "Features" for M Value field.

Outputs:

- New polyline layer
- Fields to be added to the polyline attribute table
 - [ET_ID] - the field used as Polyline ID
 - [ET_FromAtt] - the values of the start point of the polyline in the Link field (if link

field is used)

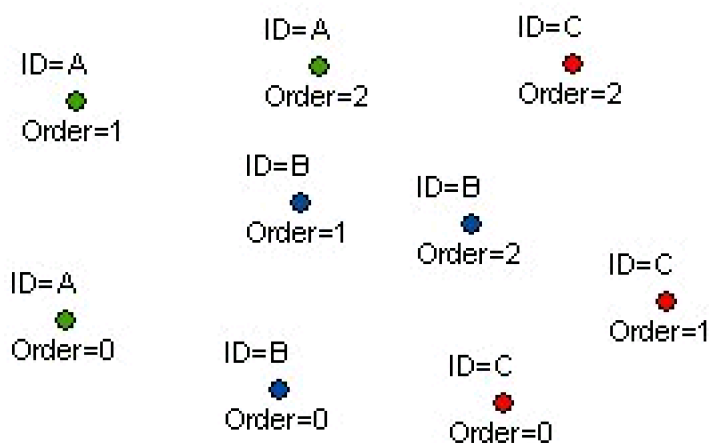
- [ET_ToAtt] - the values of the end point of the polyline in the Link field (if link field is used)

Notes:

- There should be at least two points with the same value in the ID field in order polylines to be created.

Example:

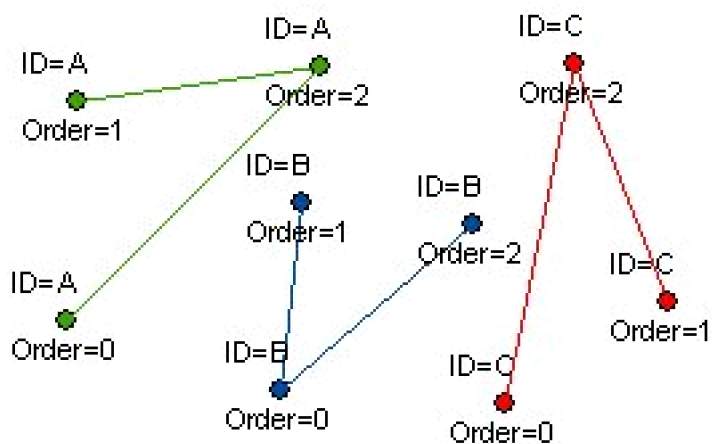
Source points



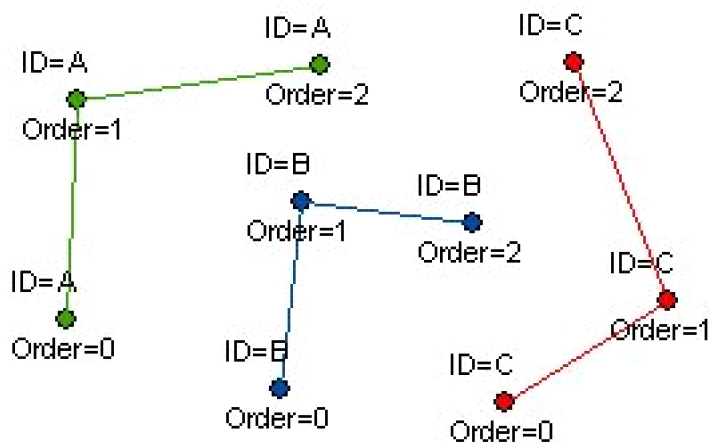
Source points attribute table

Shape*	Id	PolylineID	PointOrder
Point	0	A	0
Point	1	A	2
Point	2	A	1
Point	3	B	2
Point	4	B	0
Point	5	B	1
Point	6	C	0
Point	7	C	2
Point	8	C	1

Resulting polylines: ID Field = [PolylineID], No Order Used



Resulting polylines: ID Field = [PolylineID], Order field = [PointOrder]



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToPolylines
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.

<ID Field>	A String - the name of the field which values will indicate the points used to form a single Polyline feature.
{Order Field}	A String - the name of a Numeric (integer or double) field which values will indicate the order in which the points describe the polylines.If no Order field is used the order is defined by the record number of the points.
{Z Field}	A String - the name of the field which values will be used for Z values of the vertices. "Features" can be used if the input points have Z.
{M Field}	A String - the name of the field which values will be used for M values of the vertices. "Features" can be used if the input points have M.
{Link Field}	A String - the name of a field to be used as a link between the input points and the output. The values for the first and last point that will form a single polyline will be added to the polyline attribute table.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToPolylines", "input dataset", "output dataset", "ID Field", "Order Field", "Z Field", "M Field", "Link Field"])</code>
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToPolylines" "input dataset" "output dataset" "ID Field" "Order Field" "Z Field" "M Field" "Link Field"
.NET using ETGWOutX.dll	<code>PointsToPolylines(input dataset, output dataset, ID Field, Order Field, Z Field, M Field, Link Field)</code>
ArcPy	<code>arcpy.PointsToPolylines(input dataset, output dataset, ID Field, "Order Field", "Z Field", M Field, Link Field)</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Polygons

[Running programmatically](#)

Converts a point data set to a polygon dataset. Attaches to the polygon attribute table the values of the attributes for the first and last point that form a single polygon

Inputs:

- A point feature layer
 - REQUIRED: an ID field which value defines the points to be used for creation of each polygon
 - OPTIONAL: an Order field that defines in what sequence the points describe the polygon. If no Order field is used the order is defined by the record number of the points
 - OPTIONAL: a Link field. The values for the first and last point that will form a single polygon will be added to the polygon attribute table.
 - OPTIONAL: Z Value field. If specified a PolygonZ feature class will be created. The values in this field will be set as Z values for the vertices. If the input points have Z values, the user can specify the Z values of the input points to be used by selecting "Features" for Z Value field.
 - OPTIONAL: M Value field. If specified a PolygonM feature class will be created. The values in this field will be set as M values for the vertices. If the input points have M values, the user can specify the M values of the input points to be used by selecting "Features" for M Value field.

Outputs:

- New polygon feature class
- Fields to be added to the polygon attribute table
 - [ET_ID] - the field used as Polygon ID
 - [ET_FromAtt] - the values of the start point of the polygon in the Link field (if link field is used)

- [ET_ToAtt] - the values of the end point of the polygon in the Link field (if link field is used)

Notes:

- There should be at least three points with the same value in the ID field in order polygons to be created.

Example: [See the example for Points To Polylines function](#)

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToPolygons
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<ID Field>	A String - the name of the field which values will indicate the points used to form a single Polygon feature.
{Order Field}	A String - the name of a Numeric (integer or double) field which values will indicate the order in which the points describe the polygons.If no Order field is used the order is defined by the record number of the points.
{Z Field}	A String - the name of the field which values will be used for Z values of the vertices. "Features" can be used if the input points have Z.
{M Field}	A String - the name of the field which values will be used for M values of the vertices. "Features" can be used if the input points have M.
{Link Field}	A String - the name of a field to be used as a link between the input points and the output. The values for the first and last point that will form a single polyline

will be added to the polyline attribute table.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToPolygons", "input dataset", "output dataset", "ID Field", "Order Field", "Z Field", "M Field", "Link Field"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToPolygons" "input dataset" "output dataset" "ID Field" "Order Field" "Z Field" "M Field" "Link Field"</code>
.NET using ETGWOuX.dll	<code>PointsToPolygons(input dataset, output dataset, ID Field, Order Field, Z Field, M Field, Link Field)</code>
ArcPy	<code>arcpy.PointsToPolygons(input dataset, output dataset, ID Field, "Order Field", "Z Field", M Field, Link Field)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Multipoints

[Running programmatically](#)

Converts a point data set to a Multipoint dataset

Inputs:

- A point feature layer
 - REQUIRED: an ID field which value defines the points to be used for creation of each multipoint.
 - OPTIONAL: an Order field that defines in what sequence the points describe the multi-point. If no Order field is used the order is defined by the record number of the points
 - OPTIONAL: a Link field. The values for the first and last point that will form a single multi-point will be added to the polyline attribute table.
 - OPTIONAL: Z Value field. If specified, a MultipointZ feature class will be created. The values in this field will be set as Z values for the vertices. If the input points have Z values, the user can specify the Z values of the input points to be used by selecting "Features" for Z Value field.
 - OPTIONAL: M Value field. If specified, a MultipointM feature class will be created. The values in this field will be set as M values for the vertices. If the input points have M values, the user can specify the M values of the input points to be used by selecting "Features" for M Value field.

Outputs:

- New multipoint layer
- Fields to be added to the attribute table
 - [ET_ID] - the field used as Multipoint ID

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToMultipoints
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<ID Field>	A String - the name of the field which values will indicate the points used to form a single multipoint feature.
{Order Field}	A String - the name of a Numeric (integer or double) field which values will indicate the order in which the points describe the multipoints.If no Order field is used the order is defined by the record number of the points.
{Z Field}	A String - the name of the field which values will be used for Z values of the vertices. "Features" can be used if the input points have Z.
{M Field}	A String - the name of the field which values will be used for M values of the vertices. "Features" can be used if the input points have M.
{Link Field}	A String - the name of a field to be used as a link between the input points and the output. The values for the first and last point that will form a single polyline will be added to the polyline attribute table.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToMultipoints", "input dataset", "output dataset", "ID Field", "Order Field", "Z Field", "M Field","Link Field"])</code>
.NET using	<code>StartInfo.FileName = ETGWPath</code>

ETGWRun.exe	StartInfo.Arguments = "PointsToMultipoints" "input dataset" "output dataset" "ID Field" "Order Field" "Z Field" "M Field" "Link Field"
.NET using ETGWOutX.dll	PointsToMultipoints(input dataset, output dataset, ID Field, Order Field, Z Field, M Field,Link Field)
ArcPy	arcpy.PointsToMultipoints(input dataset, output dataset, ID Field, "Order Field", "Z Field", M Field, Link Field)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points To Points Z(M)

[Running programmatically](#)

Converts a point dataset to a point Z (M) dataset

Inputs:

- A point feature layer
 - REQUIRED: at least one numeric field with Z or M values that will be applied to the newly created Points Z (M)

Outputs:

- New point Z(M) feature class. All the original attributes are transferred.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToPointsZM
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
{Z Field}	A String - the name of the field which values will be used for Z values of the vertices. "Features" can be used if the input points have Z.
{M Field}	A String - the name of the field which values will be used for M values of the vertices. "Features" can be used if the input points have M.

Running the function

ETGWPPath used in the table below is the full path to ETGWRUN.exe (E.G. "C:\Program

Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PointsToPointsZM", "input dataset", "output dataset", "Z Field", "M Field"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToPointsZM" "input dataset" "output dataset" "Z Field" "M Field"</code>
.NET using ETGWOuX.dll	<code>PointsToPointsZM(input dataset, output dataset, Z Field, M Field)</code>
ArcPy	<code>arcpy.PointsToPointsZM(input dataset, output dataset, Z Field, M Field)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Multipoint To Point

[Running programmatically](#)

Converts a multipoint dataset to a point layer

Inputs:

- A multipoint feature layer

Outputs:

- New point layer
 - New fields are added to the point attribute table
 - [ET_ID] - the FID of original multipoints. The values can be used to link the points back to the multipoints.
 - [ET_Z] - is added and populated with Z values if the multipoint is Z aware
 - [ET_M] - is added and populated with M values if the multipoint is M aware

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	MultipointsToPoints
<input dataset>	A String representing the input layer. Must be of Multipoint type.
<output	A String - the full name of the output layer.

dataset>	
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.
{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "MultipointsToPoints", "input dataset", "output dataset", "Keep Z", "Keep M"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "MultipointsToPoints" "input dataset" "output dataset" "Keep Z" "Keep M"</pre>
.NET using ETGWOuX.dll	<code>MultipointsToPoints(input dataset, output dataset, Keep Z, Keep M)</code>
ArcPy	<code>arcpy.MultipointsToPoints(input dataset, output dataset, "Keep Z", "Keep M")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Multipoint To Polyline

[Running programmatically](#)

Converts a Multipoint dataset to a Polyline feature class

Inputs:

- A Multipoint feature layer

Outputs:

- New Polyline feature class

Notes :

- The attributes of the original multipoints are transferred to the resulting polylines.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	MultipointsToPolylines
<input dataset>	A String representing the input layer. Must be of Multipoint type.
<output dataset>	A String - the full name of the output layer.
{Keep Z}	A Boolean indicating whether if the input has Z values to keep them in the output.
{Keep M}	A Boolean indicating whether if the input has M values to keep them in the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "MultipointsToPolylines", "input dataset", "output dataset", "Keep Z", "Keep M"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "MultipointsToPolylines" "input dataset" "output dataset" "Keep Z" "Keep M"</code>
.NET using ETGWOuX.dll	<code>MultipointsToPolylines(input dataset, output dataset, Keep Z, Keep M)</code>
ArcPy	<code>arcpy.MultipointsToPolylines(input dataset, output dataset, "Keep Z", "Keep M")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Shape Z (M) To Shape

[Running programmatically](#)

Converts a Z (M) data set to a plain (no Z or M) dataset

Inputs:

- A Z aware or M aware feature layer
 - Point
 - Multipoint
 - Polyline
 - Polygon

Outputs:

- New layer
 - The original attributes are preserved

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ShapeZMToShape
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ShapeZMToShape", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "ShapeZMToShape" "input dataset" "output dataset"</code>
.NET using ETGWOuX.dll	<code>ShapeZMToShape(input dataset, output dataset)</code>
ArcPy	<code>arcpy.ShapeZMToShape(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Shape To ShapeZ

[Running programmatically](#)

Converts the features of a data set to 3D features with constant Z value

Inputs:

- A point, multipoint, polyline or polygon feature layer
 - REQUIRED: a numeric field with Z values that will be applied to the newly created 3D geometries.

Outputs:

- New PointZ, MultipointZ, PolylineZ or PolygonZ (depending on the input) layer. All the original attributes are transferred.
- If the input is Polyline and Add M is selected, the M values for the polyline vertices will be calculated based for each vertex on it's distance from the beginning of the polyline.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ShapeToShapeZ
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
{Z Field}	A String - the name of the field which values will be used for Z values of the vertices. "Features" can be used if the input points have Z.

{Add M}	A Boolean indicating whether to add M values to the output
---------	--

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ShapeToShapeZ", "input dataset", "output dataset", "Z Field", "Add M"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ShapeToShapeZ" "input dataset" "output dataset" "Z Field" "Add M"</pre>
.NET using ETGWOuX.dll	<code>ShapeToShapeZ(input dataset, output dataset, Z Field, Add M)</code>
ArcPy	<code>arcpy.ShapeToShapeZ(input dataset, output dataset, Z Field, Add M)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clip

[Running programmatically](#)

Clips a feature layer with the features of a polygon layer

Inputs:

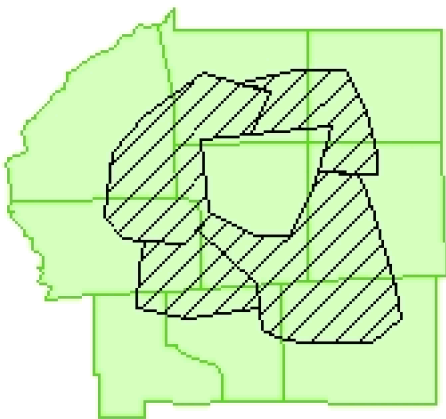
- Layer to be clipped - a Point, Multipoint, Polyline or Polygon layer
- Clip layer - a polygon layer which features will be used for clipping

Outputs:

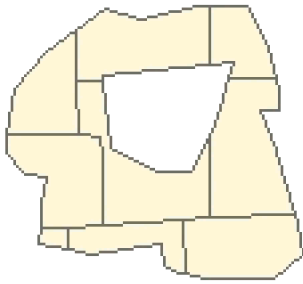
- New layer (Point, Multipoint, Polyline or Polygon depending on the type of the original layer)
 - The attributes are preserved
 - The spatial reference of the input data set is preserved

Examples:

Input Layers



Result



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ClipSingle
<input dataset>	A String representing the input layer.
<Clip Dataset>	A String representing the layer to be merged. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ClipSingle", "input dataset", "Clip Dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "ClipSingle" "input dataset" "Clip Dataset" "output dataset"</code>

	dataset"
.NET using ETGWOutX.dll	ClipSingle(input dataset, Clip Dataset, output dataset)
ArcPy	arcpy.ClipSingle(input dataset, Clip Dataset, "output dataset")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Batch Clip

[Running programmatically](#)

Clips a batch of feature layers with the features of a polygon layer

Inputs:

- Layers to be clipped - a Point, Multipoint, Polyline or Polygon layers
- Clip layer - a polygon layer which features will be used for clipping
- Workspace where the clipped datasets will be stored

Outputs:

- New layers (Point, Multipoint, Polyline or Polygon depending on the types of the original layers)
 - The attributes are preserved
 - The spatial reference of the input data set is preserved
 - The new datasets will be named after the original layers with the user specified suffix

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	BatchClip
<input datasets>	A String - a list separated with ";" indicating the full paths to the datasets to be clipped.

<clip dataset>	A String representing the clip layer. Must be of Polygon type.
<output workspace>	A String - the full name of the output folder (for shapefiles) or File GDB.
{Suffix}	A String representing the suffix added to the output datasets.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "BatchClip", "input datasets", "clip dataset", "output workspace", "Suffix"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "BatchClip" "input datasets" "clip dataset" "output workspace" "Suffix"
.NET using ETGWOuX.dll	BatchClip(input datasets, clip dataset, output workspace, Suffix)
ArcPy	arcpy.BatchClip(input datasets, clip dataset, "output workspace", "Suffix")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Clip

[Running programmatically](#)

Erases a feature layer with the features of a polygon layer

Inputs:

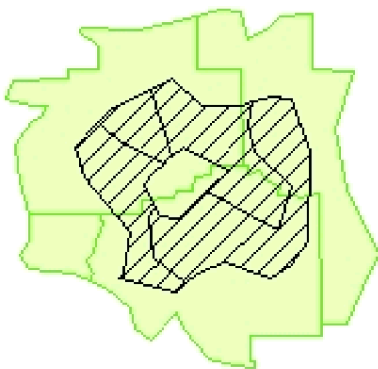
- Layer to be erased - a Point, Multipoint, Polyline or Polygon layer
- Erase layer - a polygon layer which features will be used for clipping

Outputs:

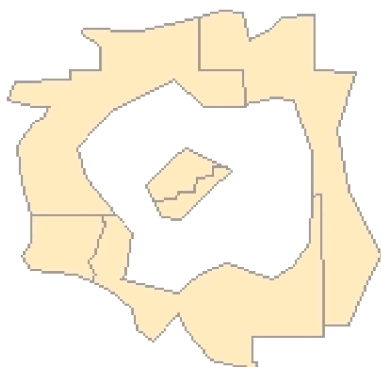
- New layer (Point, Multipoint, Polyline or Polygon depending on the type of the original layer)
 - The attributes are preserved
 - The spatial reference of the input data set is preserved

Examples:

Input Layers



Result



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	EraseSingle
<input dataset>	A String representing the input layer.
<Erase Dataset>	A String representing the layer to be merged. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "EraseSingle", "input dataset", "Erase Dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "EraseSingle" "input dataset" "Erase Dataset" "output dataset"</code>

.NET using ETGWOtX.dll	EraseSingle(input dataset, Erase Dataset, output dataset)
ArcPy	arcpy.EraseSingle(input dataset, Erase Dataset, "output dataset")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Batch Erase

[Running programmatically](#)

Clips a batch of feature layers with the features of a polygon layer

Inputs:

- Layers to be erased - a Point, Multipoint, Polyline or Polygon layers
- Erase layer - a polygon layer which features will be used for clipping
- Workspace where the erased datasets will be stored

Outputs:

- New layers (Point, Multipoint, Polyline or Polygon depending on the types of the original layers)
 - The attributes are preserved
 - The spatial reference of the input data set is preserved
 - The new datasets will be named after the original layers with the user specified suffix

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	BatchErase
<input datasets>	A String - a list separated with ";" indicating the full paths to the datasets to be clipped.

<clip dataset>	A String representing the Erase layer. Must be of Polygon type.
<output workspace>	A String - the full name of the output folder (for shapefiles) or File GDB.
{Suffix}	A String representing the suffix added to the erase datasets.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "BatchErase", "input datasets", "erase dataset", "output workspace", "Suffix"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "BatchErase" "input dataset" "erase dataset" "output workspace" "Suffix"
.NET using ETGWOutX.dll	BatchErase(input dataset, erase dataset, output workspace, Suffix)
ArcPy	arcpy.BatchErase(input dataset, erase dataset, "output workspace", "Suffix")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Merge Layers

[Running programmatically](#)

Merges two layers with the same geometry type into a single output layer.

Inputs:

- Two layers with the same geometry type
- Add all attributes - if the option is selected the attributes from both layers will be added to the output.

Outputs:

- New layer

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	MergeLayers
<input dataset>	A String representing the input layer.
<merge dataset>	A String representing the layer to be merged. Must be of the same geometry type as the input layer.
<output dataset>	A String - the full name of the output layer.
{All Attributes}	Optional. A Boolean indicating whether the attributes of the Merge layer to be transferred to the output.

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "MergeLayers", "input dataset", "merge dataset", "output dataset", "All Attributes"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "MergeLayers" "input dataset" "merge dataset" "output dataset" "All Attributes"</pre>
.NET using ETGWOuX.dll	<code>MergeLayers(input dataset, merge dataset, output dataset, All Attributes)</code>
ArcPy	<code>arcpy.MergeLayers(input dataset, merge dataset, "output dataset" , "All Attributes")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Merge Feature Layers

[Running programmatically](#)

Merges layers from the same geometry type together into a single output layer.

Inputs:

- A base layer. This layer defines what will be the type of the output. The fields of this layer will be preserved. If the base layer has Z/M dimension, only layers with Z/M dimension can be merged to it
 - Point, PointZ, PointM
 - Polyline, PolylineZ, PolylineM
 - Polygon, PolygonZ, PolygonM
- Layers to merge. If the a field name has the same name as a field in the base layer, the attributes will be retained.
- Output file name

Outputs:

- A feature class containing all the features from the base layer and the merge layers. If the base layer has Z/M dimension, the output will have Z/M dimension as well. All the attributes are retained (if the fields are present in the base layer)

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	MergeMulti

<Base Dataset>	A String representing the input layer.
<merge datasets>	A String representing a list (separator - ";") of layers to be merged to the base dataset. All layers in this list be of the same geometry type as the base layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "MergeMulti", "Base Dataset", "merge datasets", "output dataset"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "MergeMulti" "Base Dataset" "merge datasets" "output dataset"</pre>
.NET using ETGWOuX.dll	<code>MergeMulti(Base Dataset, merge datasets, output dataset)</code>
ArcPy	<code>arcpy.MergeMulti(Base Dataset, merge datasets, "output dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Advanced Merge

[Running programmatically](#)

Merges two polygon data sets. The result does not contain overlaps

Inputs:

- Base layer - a polygon layer that will keep all attribute fields
- Merge layer - a polygon layer that will be merged to the Base layer.
- Priority of the Merge layer as described below
 - Priority - "Erase" - the polygons from the Base layer are erased with the polygons of the Merge layer
 - Priority - "Low" - only the polygons (or portions of them) from the Merge layer that do not overlap with these of the Base layer are added to the new layer
 - Priority - "Standard" - Creates intersections where the polygons from the Merge layer intersect these from the base layer. The intersection polygons carry the attributes of the corresponding polygons from both layers
 - Priority - "High" - The polygons from the Merge layer are entirely preserved. Only these polygons (or portions of them) from the base layer that do not overlap with the polygons from the Merge layer are added to the output.

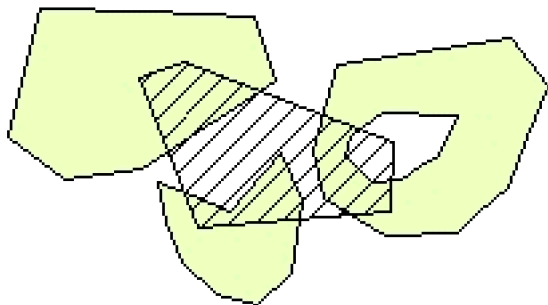
Outputs:

- New Polygon layer
 - No overlaps present. The polygons from the Base and the Merge layers are overlaid depending the priority of the Merge layer
 - All the attributes of the Base layer are preserved
 - Only the attributes in fields with the same name and type as these from the Base layer are preserved for the features from the Merge layer. Exception makes priority "Standard" which copies the attributes from the merge layer as well.

- Base layer always have a Priority "Standard"

Examples:

Input Layers



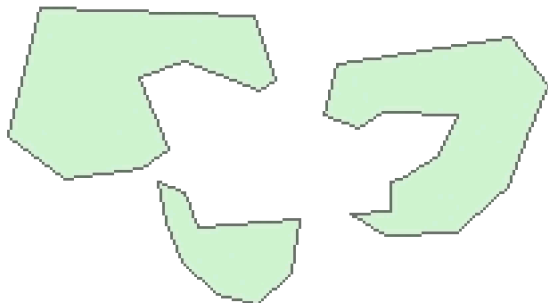
Base layer table

Id	a	b
1	aaa	ddd
2	bbb	eee
3	ccc	fff

Merge layer table

Id	y	a
1	yyy	xxx

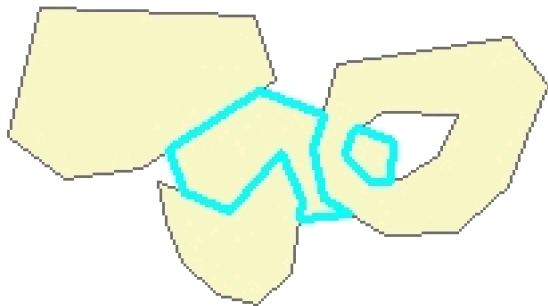
Result: Priority of "-1" (Erase)



Result table

Id	a	b
2	bbb	eee
3	ccc	fff
1	aaa	ddd

Result: Priority of "0" (Low)



Result table

Id	a	b
1	aaa	ddd
2	bbb	eee
3	ccc	fff
1	xxx	
1	xxx	

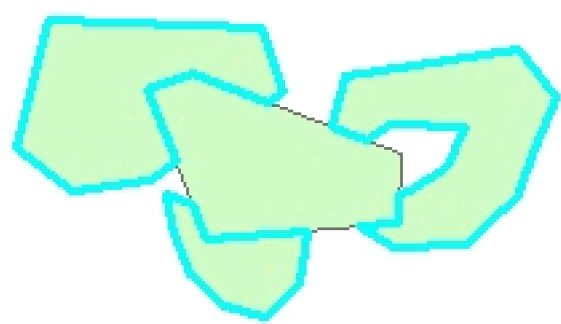
Result: Priority of "1" (Standard)



Result table

Id	a	b	Id_1	y	a_1
2	bbb	eee	1	yyy	xxx
3	ccc	fff	1	yyy	xxx
1	aaa	ddd	1	yyy	xxx
1	xxx		0	yyy	
1	xxx		0	yyy	
2	bbb	eee	0		
3	ccc	fff	0		
1	aaa	ddd	0		

Result: Priority of "2" (High)



Id	a	b
2	bbb	eee
3	ccc	fff
1	aaa	ddd
1	xxx	

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	AdvancedMerge
<input dataset>	A String representing the input layer. Must be of Polygon type.
<merge dataset>	A String representing the layer to be merged. Must be of Polygon type.

<output dataset>	A String - the full name of the output layer.
<Merge Priority>	<p>Required. A string representing the priority of the polygons to be merged</p> <ul style="list-style-type: none"> • Erase - the polygons from the Base dataset are erased with the polygons of the Merge dataset. • Low - only the polygons (or portions of them) from the Merge dataset that do not overlap with these of the Base layer are added to the new dataset. • Standard - Creates intersections where the polygons from the Merge dataset intersect these from the base layer. The intersection polygons carry the attributes of the corresponding polygons from both datasets. • High - The polygons from the Merge dataset are entirely preserved. Only these polygons (or portions of them) from the base dataset that do not overlap with the polygons from the Merge dataset are added to the output.
{All Attributes}	Optional. A Boolean indicating whether the attributes of the Merge layer to be transferred to the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "AdvancedMerge", "input dataset", "merge dataset", "output dataset", "Merge Priority", "All Attributes"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "AdvancedMerge" "input dataset" "merge dataset" "output dataset" "Merge Priority" "All Attributes"
.NET using ETGWOuX.dll	AdvancedMerge(input dataset, merge dataset, output dataset, Merge Priority, All Attributes)

ArcPy

```
arcpy.AdvancedMerge(input dataset, merge dataset, "output dataset" ,  
"Merge Priority", "All Attributes")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Split By Location

[Running programmatically](#)

Clips the features of the input layer with the polygons of the split layer. Creates a new dataset for the features of the input layer (or portions of them) contained by each polygon from the split layer

Inputs:

- Layer to be clipped - a Point, Multipoint, Polyline or Polygon layer
- Clip layer - a polygon layer which features will be used for clipping
- Output folder
- Name field - a field from the Clip layer that will be used for naming of the output datasets
- Prefix - a text that will be used together with the Name field for generating names of the output datasets

Outputs:

- New layers
 - The attributes are preserved
 - The spatial reference of the input data set is preserved

Notes:

- If there are no features from the input layer that are fully or partially within a polygon from the clip layer - no dataset will be created for this polygon

Examples:

The naming of the output dataset is based on the Name field selected and the prefix.

- If the Prefix = "Rivers" and the Name field = "StateNames" the resulting feature classes will be named
 - "Rivers_Nevada.shp"
 - "Rivers_Texas.shp"

- "Rivers_Arizona.shp"
-

- If the prefix box is left empty and the Name field = "StateNames" the resulting feature classes will be named
 - "Nevada.shp"
 - "Texas.shp"
 - "Arizona.shp"
 -
- If the Prefix = "Roads" and the Name field = "StateNames" and in the output folder there is an existing feature class named "Roads_Nevada.shp", the new feature class will be named "Roads_Nevada1.shp"

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SplitByLocation
<input dataset>	A String representing the input layer.
<clip dataset>	A String representing the clip layer.
<output workspace>	A String - the full name of the output folder (for shapefiles) or File GDB.
<Name Field>	A String - the name of the field which values are to be used for naming the output datasets.
{Prefix}	A String representing the prefix added to the names of the output datasets.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "SplitByLocation", "input dataset", "clip dataset", "output workspace", "Name Field", "Prefix"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "SplitByLocation" "input dataset" "clip dataset" "output workspace" "Name Field" "Prefix"</pre>
.NET using ETGWOutX.dll	<code>SplitByLocation(input dataset, clip dataset, output workspace, Name Field, Prefix)</code>
ArcPy	<code>arcpy.SplitByLocation(input dataset, clip dataset, output workspace, "Name Field", "Prefix")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Split By Attributes

[Running programmatically](#)

Splits a layer into separate datasets based on an the attribute values in the selected field

Inputs:

- Layer to be split - a Point, Multipoint, Polyline or Polygon layer
- Output workspace (folder or File GDB)
- Name field - a field from the input layer that will be used for splitting
- Prefix - a text that will be used together with the Name field for generating names of the output workspaces

Outputs:

- New feature layers
 - The attributes are preserved
 - The spatial reference of the input data set is preserved

Examples:

The naming of the output workspace is based on the Name field selected and the prefix.

- If the Prefix = "Rivers" and the Name field = "StateNames" the resulting feature classes will be named
 - "Rivers_Nevada.shp"
 - "Rivers_Texas.shp"
 - "Rivers_Arizona.shp"
 -
- If the prefix box is left empty and the Name field = "StateNames" the resulting feature classes will be named
 - "Nevada.shp"
 - "Texas.shp"
 - "Arizona.shp"

- If the Prefix = "Roads" and the Name field = "StateNames" and in the output folder there is an existing feature class named "Roads_Nevada.shp", the new feature class will be named "Roads_Nevada1.shp"

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SplitByAttributes
<input dataset>	A String representing the input layer.
<output workspace>	A String - the full name of the output folder (for shapefiles) or File GDB.
<Split Field>	A String - the name of the field which values are to be used for splitting.
{Prefix}	A String representing the prefix added to the names of the output datasets.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "SplitByAttributes", "input dataset", "output workspace", "Split Field", "Prefix"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "SplitByAttributes" "input dataset" "output workspace" "Split Field" "Prefix"
.NET using ETGWOutX.dll	SplitByAttributes(input dataset, output workspace, Split Field, Prefix)

ArcPy

```
arcpy.SplitByAttributes(input dataset, output workspace, "Split Field", "Prefix")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Transfer Polygon Attributes

[Running programmatically](#)

Transfers the attributes from one polygon layer (source) to another (target) based on their spatial location (overlay). The user specifies the method for transferring the attributes of each field of the source polygon attribute table.

Inputs:

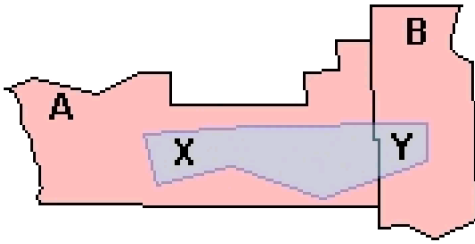
- Target layer - a polygon layer that will receive the attributes
- Source layer - a polygon layer which attributes will be transferred to the target layer
- Fields which values will be transferred
- The method that will be used to transfer the values for each field. The methods are discussed below
 - Count (only for numeric fields)
 - Value (only for numeric fields)
 - Type

Outputs:

- New Polygon layer
 - All the attributes of the Target layer are preserved
 - The fields of the Source layer selected for transfer will be added to the attribute table and their values will be calculated based on the transfer method specified

Transfer Methods:

The Source dataset has two polygons A and B. The Target dataset has a single polygon - Z. The portion of the Target polygon that intersects with polygon "A" of the Source layer is polygon X, and the portion that intersects with polygon B is polygon Y.



- Count (sum proportion) - Typical application - transferring census data.

$$\text{population_Z} = \text{population_A} * \text{area_X} / \text{area_A} + \text{population_B} * \text{area_Y} / \text{area_B}$$

- Value (weighted average) - Typical application - transferring rainfall data

$$\text{rainfall_Z} = (\text{rainfall_A} * \text{area_X} + \text{rainfall_B} * \text{area_Y}) / \text{area_Z}$$

- Type (majority) - Typical application - transferring text data (soil type etc.)

IF $\text{area_X} / \text{area_Z} > \text{area_Y} / \text{area_Z}$ THEN $\text{soiltype_Z} = \text{soiltype_A}$

IF $\text{area_X} / \text{area_Z} < \text{area_Y} / \text{area_Z}$ THEN $\text{soiltype_Z} = \text{soiltype_B}$

Notes:

- In order correct results to be obtained both Source and Target datasets should be clean from overlaps
- The procedure performs cleaning of both Source and Target datasets to avoid incorrect results. The cleaning is performed on temporary datasets. No changes are applied to the input data. If the target dataset has overlapping polygons, a new polygon representing the overlap will be created in the output polygon dataset
- The spatial references of the Source and the Target datasets must have the same geographic coordinate system

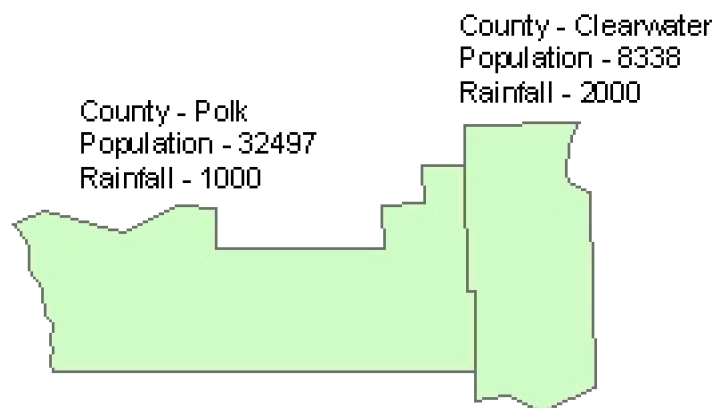
Example:

Source Data

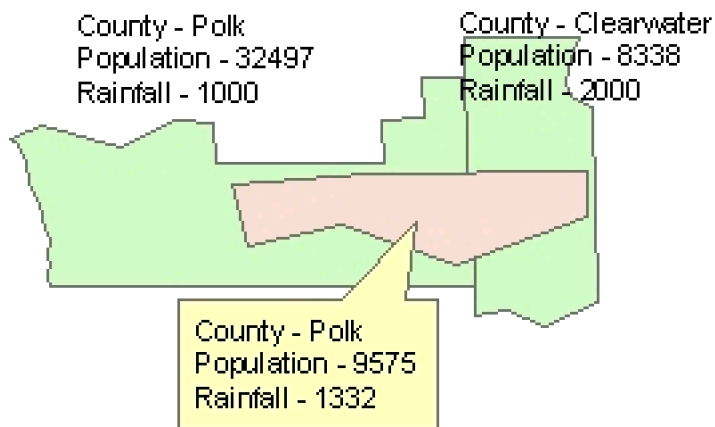
Transfer Methods

- County - Type
- Population - Count

- Rainfall - Value



Results



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	TransferAttributes
<Source Dataset>	A String representing the input layer. Must be of Polygon type.
<Target Dataset>	A String representing the layer to be merged. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.

<Transfer Fields<	A String representing a list (separator ";") of the fields to transfer together with the method for each field. Valid values - "Count", "Value", "Type". Example: "Field1 Value; Field2 Type; Field3 Count"
-------------------	---

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "TransferAttributes", "Source Dataset", "Target Dataset", "output dataset", "Transfer Fields"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "TransferAttributes" "Source Dataset" "Target Dataset" "output dataset" "Transfer Fields"</pre>
.NET using ETGWOutX.dll	<code>TransferAttributes(Source Dataset, Target Dataset, output dataset, Transfer Fields)</code>
ArcPy	<code>arcpy.TransferAttributes(Source Dataset, Target Dataset, "output dataset" , "Transfer Fields")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Remove Exact Duplicates

[Running programmatically](#)

Removes duplicates with exactly the same shape from a Point, Multipoint, Polyline or Polygon dataset.

Inputs:

- A feature layer (Point, Multipoint, Polygon, Polyline)
- OPTIONAL: A reference layer (Should have the same shape type as the input layer)

Outputs:

- New Point, Multipoint, Polyline or Polygon dataset (depending on the input) with no features with duplicate shapes present. If a reference dataset is specified, all the features from the input dataset that have exactly the same shapes with a feature from the reference dataset will be excluded from the output.

Notes:

- If a reference dataset is used, the features from the input dataset that have exactly the same geometry as features from the reference dataset will be removed.
- If features with exactly the same shapes are found in the input dataset, only the first feature will be saved in the output.
- If you want to remove overlaps from partially overlapping geometries use Clean Polygon or Clean Polyline functions

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RemoveDuplicates

<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
{Reference Dataset}	A String representing the reference layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "RemoveDuplicates", "input dataset", "output dataset", "Reference Dataset"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "RemoveDuplicates" "input dataset" "output dataset" "Reference Dataset"
.NET using ETGWOutX.dll	RemoveDuplicates(input dataset, output dataset,Reference Dataset)
ArcPy	arcpy.RemoveDuplicates(input dataset, output dataset,Reference Dataset)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Symmetrical Difference

[Running programmatically](#)

Calculates the geometric intersection of the input polygon feature classes. Creates a polygon feature class that contains the areas of both input datasets that do not overlap.

Inputs:

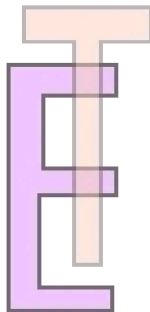
- Two polygon feature classes
- Add all attributes - if the option is selected the attributes from both layers will be added to the output.

Outputs:

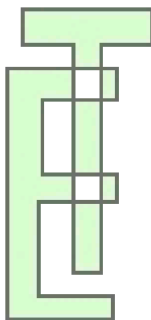
- New polygon layer

Examples:

Input Layers



Result



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SymmetricalDifference
<input dataset>	A String representing the input layer. Must be of Polygon type.
<merge dataset>	A String representing the layer to be merged. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.
{All Attributes}	Optional. A Boolean indicating whether the attributes of the Merge layer to be transferred to the output.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SymmetricalDifference", "input dataset", "merge dataset", "output dataset", "All Attributes"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SymmetricalDifference" "input dataset" "merge dataset" "output dataset" "All Attributes"</code>
.NET using ETGWOutX.dll	<code>SymmetricalDifference(input dataset, merge dataset, output dataset, All Attributes)</code>
ArcPy	<code>arcpy.SymmetricalDifference(input dataset, merge dataset, "output dataset", "All Attributes")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Intersect Polygons

[Running programmatically](#)

Produces a new layer which is the geometric intersection of the two input polygon layers.

Inputs:

- Two polygon layers to be intersected

Outputs:

- New Polygon layer
 - The attributes of both input layers are preserved
 - The spatial reference of the input dataset is preserved

Notes:

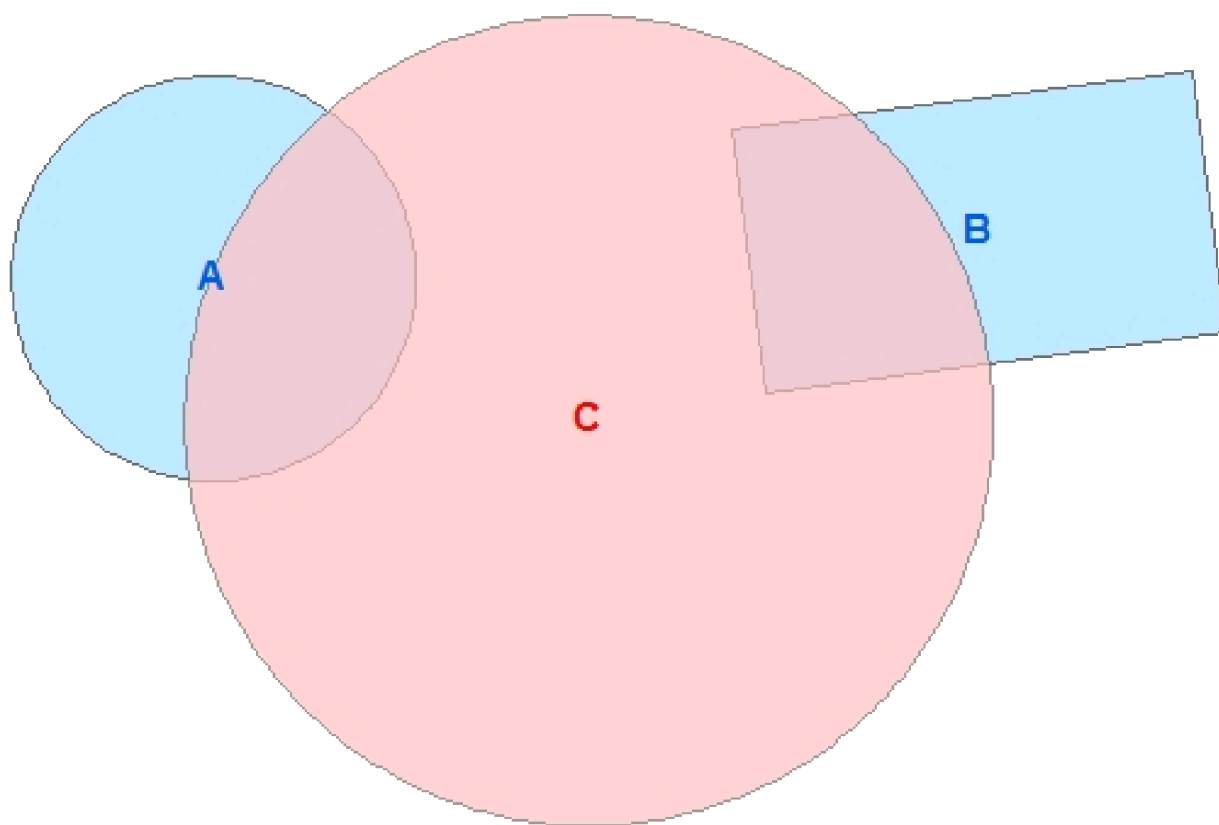
The function works only for polygon with polygon intersections.

For intersection of layers with different geometries use

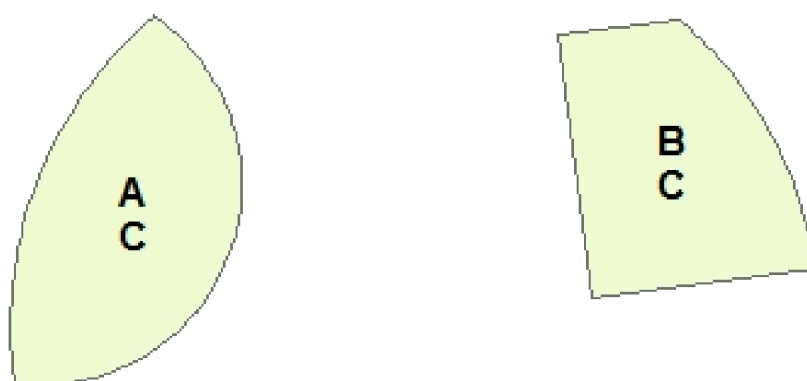
- Polygons with Polylines
 - [Clip](#) and [Spatial Join](#) for polyline output
 - [Point Intersection](#) for Point output
- Polylines with Polylines - [Point Intersection](#)
- Polygons with Points - [Spatial Join](#)

Examples:

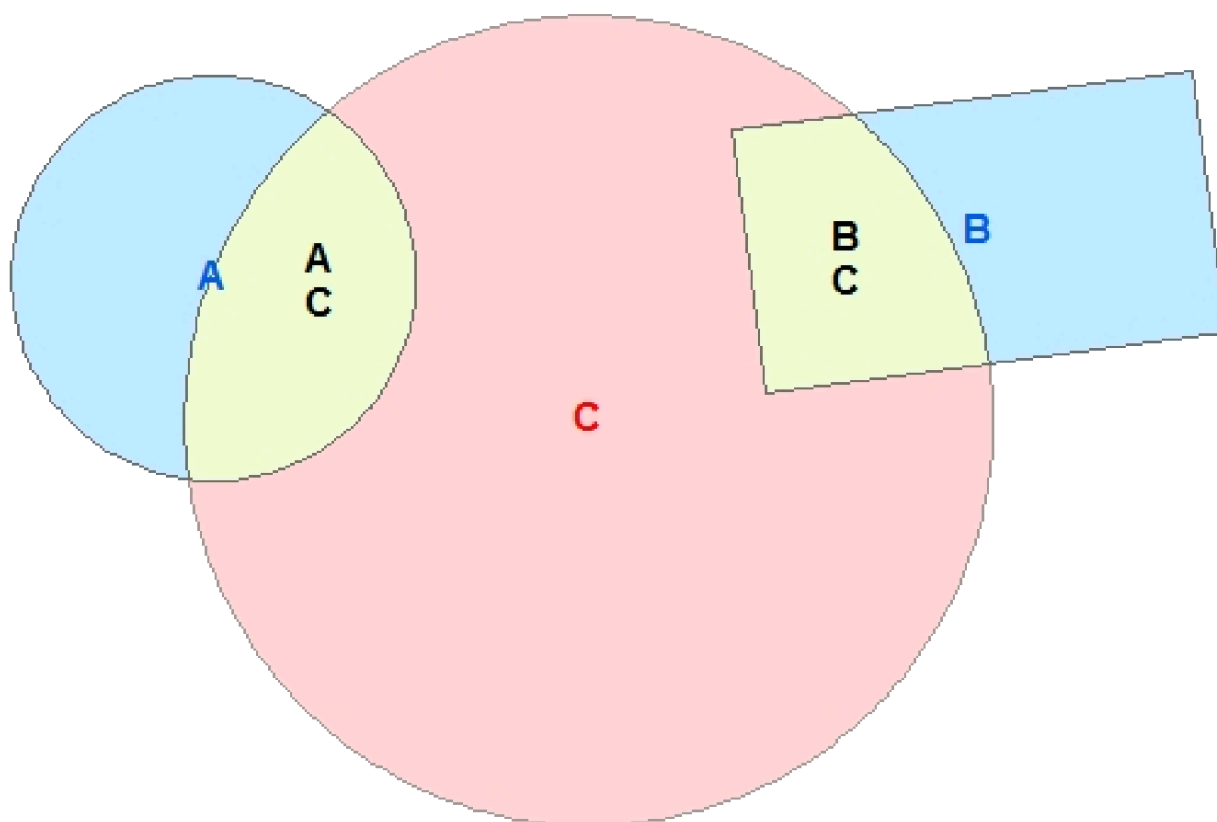
Input Layers



Result



Result overlayed with the inputs



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	IntersectPolygons
<input dataset>	A String representing the input layer. Must be of Polygon type.
<Intersect Dataset>	A String representing the layer to be used for intersect. Must be of Polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPPath used in the table below is the full path to ETGWRUN.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRUN.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "IntersectPolygons", "input dataset", "Intersect Dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "IntersectPolygons" "input dataset" "Intersect Dataset" "output dataset"</pre>
.NET using ETGWOuX.dll	<code>IntersectPolygons(input dataset, Intersect Dataset, output dataset)</code>
ArcPy	<code>arcpy.IntersectPolygons(input dataset, Intersect Dataset, "output dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create new File GDB

[Running programmatically](#)

Creates a new File Geodatabase in the user specified location

Inputs:

- Output Folder
- Name for the new File GDB

Outputs:

- A new File GDB

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CreateFileGDB
<Output Folder>	A String representing the folder where the output File GDB will be created.
<File GDB Name>	A String representing the name of the File GDB to be created.

Running the function

ETGWPPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPPath, "CreateFileGDB", "Output Folder", "File GDB Name"])</code>

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateFileGDB" "Output Folder" "File GDB Name"
.NET using ETGWOutX.dll	CreateFileGDB(Output Folder, File GDB Name)
ArcPy	arcpy.CreateFileGDB(Output Folder, File GDB Name)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Buffer

[Running programmatically](#)

Creates buffer polygons from the features of the input dataset. The buffer distance (in the units of the spatial reference of the input dataset) can be entered as a number (equal for all input features) or a numeric field.

Inputs:

- A point, multipoint, polyline or polygon feature layer
- Buffer distance - a number (the same buffer distance will be used for all input polylines) or the name of a numeric field in the polyline attribute table that has the buffer distance for each input polyline.
- Dissolve option - the boundaries of the intersecting buffers will be dissolved. The original attributes will not be preserved if the dissolve option is used.

Outputs:

- New polygon feature class

Notes :

- The attributes of the input features will be transferred to the resulting polygons only if the Dissolve option is NOT used.
- For advanced buffering of polylines see the [Buffer Polylines](#) function.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	Buffer

<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
{Buffer Distance}	A Double representing the buffer distance in the units of the Spatial Reference of the input layer.
{Buffer Field}	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the buffer distance.
{Dissolve Buffers}	A Boolean. If True - the boundaries of the intersecting buffers will be dissolved.
{Dissolve Fields}	A String representing a list (separated with ";") with the field names to be used for dissolving.
{Statistic Fields}	A String representing a list (separated with ";") with the field names for which statistics will be created. Example: "Field1 Sum;Field2 Max;Field3 Min"
{Create Multiparts}	A Boolean indicating whether the function will create multipart polygons.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "Buffer", "input dataset", "output dataset", "Buffer Distance", "", "Dissolve Buffers", "Dissolve Fields", "Statistic Fields", "Create Multiparts"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "Buffer" "input dataset" "output dataset" "Buffer Distance" "" "Dissolve Buffers" "Dissolve Fields" "Statistic Fields" "Create Multiparts"
.NET using ETGWOutX.dll	Buffer(input dataset, output dataset, Buffer Distance, "", Dissolve Buffers, Dissolve Fields, Statistic Fields, Create Multiparts)

ArcPy

```
arcpy.Buffer(input dataset, output dataset, "Buffer Distance", "", "Dissolve  
Buffers", "Dissolve Fields", "Statistic Fields", "Create Multiparts")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Explode Multipart

[Running programmatically](#)

Explodes the multi-part features from a polygon or polyline layer. The resulting data set will not contain multi-part features

Inputs

- A polygon or polyline feature layer

Outputs

- A layer with no multipart shapes present.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ExplodeMultipart
<input dataset>	A String representing the input layer. Must be Polyline or Polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ExplodeMultipart", "input dataset", "output dataset"])</code>
.NET using	<code>StartInfo.FileName = ETGWPath</code>

ETGWRun.exe	StartInfo.Arguments = "ExplodeMultipart" "input dataset" "output dataset"
.NET using ETGWOutX.dll	ExplodeMultipart(input dataset, output dataset)
ArcPy	arcpy.ExplodeMultipart(input dataset, output dataset)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Sort Shapes

[Running programmatically](#)

Sorts the features of a feature layer according to user specified fields and order methods.

Inputs:

- A feature layer
 - Point
 - Multipoint
 - Polyline
 - Polygon
- Fields to be used for sorting
- Sort order for each field

Outputs:

- New feature class sorted according the selected fields

How to use:

- Select layer to be sorted and location for the new feature class
- A list of all the fields in the layer is presented in a list box. Using the arrow buttons move the fields to be used for sorting to the sort fields list box
- Use the Up and Down buttons to arrange the fields in the order they will be used in the sorting process. Click the Next button
- For each field select sort order (Ascending or Descending). Clicking on the cell with the sort order toggles the method

- Click the Finish button

Notes:

- The fields are used for sorting in the order they have in the selected fields list box
- The function might be very useful:
 - if there are small polygons hidden beneath larger ones. In this case sorting descending by the area will show all the polygons
 - if point data has to be displayed using Pie charts. If the points are close to each other some of the pies might be hidden by the adjacent ones with larger values in classification field. If the shapes are sorted in descending order using the classification field the small pies will be visible on top of the big ones

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SortShapes
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Sort Fields>	A String representing a list (separator ";") of the fields to be sorted together with the sort method for each field. Example: "Field1 Ascending;Field2 Descending;Field3 Descending"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SortShapes", "input dataset", "output dataset", "Sort Fields"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SortShapes" "input dataset" "output dataset" "Sort Fields"</pre>
.NET using ETGWOuX.dll	<code>SortShapes(input dataset, output dataset, Sort Fields)</code>
ArcPy	<code>arcpy.SortShapes(input dataset, output dataset, "Sort Fields")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Move Shapes

[Running programmatically](#)

Moves the features of a feature layer according to user specified translation vector .

Inputs:

- A feature layer
 - Point
 - Multipointoint
 - Polyline
 - Polygon
- Delta X - movement in X direction
- Delta Y - movement in Y direction

Outputs:

- New layer
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function	MoveShapes

Name	
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Delta X>	A Double representing the X component of the translation vector.
<Delta Y>	A Double representing the Y component of the translation vector.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "MoveShapes", "input dataset", "output dataset", "Delta X", "Delta Y"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "MoveShapes" "input dataset" "output dataset" "Delta X" "Delta Y"</pre>
.NET using ETGWOutX.dll	<code>MoveShapes(input dataset, output dataset, Delta X, Delta Y)</code>
ArcPy	<code>arcpy.MoveShapes(input dataset, output dataset, "Delta X", "Delta Y")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Copyright © Ianko Tchoukanski

Rotate Shapes

[Running programmatically](#)

Rotates the features of a feature layer around user specified rotation point and rotation angle .

Inputs:

- A feature layer
 - Point
 - Multioint
 - Polyline
 - Polygon
- Rotation angle - angle in degrees staring from North counterclockwise.
- Origin point for the rotation might be:
 - input X,Y
 - the first point of a reference layer

Outputs:

- New feature class
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RotateShapes
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Rotation Angle>	A Double representing the rotation angle.
{Reference Dataset}	A String representing the input layer. Must be of Point type. The first point of this dataset will be used as origin point.
{Origin X}	A Double representing the X coordinate of the origin point.
{Origin Y}	A Double representing the Y coordinate of the origin point.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "RotateShapes", "input dataset", "output dataset", "Rotation Angle", "Reference Dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "RotateShapes" "input dataset" "output dataset" "Rotation Angle" "Reference Dataset"</code>
.NET using ETGWOutX.dll	<code>RotateShapes(input dataset, output dataset, Rotation Angle, Reference Dataset)</code>
ArcPy	<code>arcpy.RotateShapes(input dataset, output dataset, "Rotation Angle", "Reference Dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Scale Shapes

[Running programmatically](#)

Scales the features of a feature layer according to user specified anchor point X and Y scale factors .

Inputs:

- A feature layer
 - Point
 - Multipoint
 - Polyline
 - Polygon
- X scale factor
- Y scale factor
- Origin point for the scaling might be:
 - input X,Y
 - the first point of a reference layer

Outputs:

- New layer
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ScaleShapes
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Scale Factor X>	A Double representing the Scale Factor in X direction.
<Scale Factor Y>	A Double representing the Scale Factor in Y direction.
{Reference Dataset}	A String representing the input layer. Must be of Point type. The first point of this dataset will be used as origin point.
{Origin X}	A Double representing the X coordinate of the origin point.
{Origin Y}	A Double representing the Y coordinate of the origin point.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ScaleShapes", "input dataset", "output dataset", "Scale Factor X", "Scale Factor Y", "Reference Dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ScaleShapes" "input dataset" "output dataset" "Scale Factor X" "Scale Factor Y" "Reference Dataset"</code>
.NET using ETGWOuX.dll	<code>ScaleShapes(input dataset, output dataset, Scale Factor X, Scale Factor Y, Reference Dataset)</code>

ArcPy

```
arcpy.ScaleShapes(input dataset, output dataset, "Scale Factor X", "Scale  
Factor Y", "Reference Dataset")
```

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Closest Feature Distance

[Running programmatically](#)

Calculates the distance for each feature of a dataset to the closest feature from the same dataset (Point, Polyline or Polygon). The function is slow and can hang on large datasets. If the input dataset is a Point one, we recommend using the [Find Closest Point](#) function added in version 11.1

Inputs:

- A feature layer
- Search tolerance - the maximum distance to search for neighboring features.

Outputs:

- A new layer. The attribute table of the resulting layer will have three new fields
 - [ET_ID] - the ID of the feature
 - [ET_Dist] - the distance from the feature to the closest feature.
 - [ET_Closest] - the ID of the closest feature.

Notes:

- If the distance from a feature to the closest feature is larger than the Search Tolerance then the [ET_Dist] and [ET_Closest] will have a value of -1
- If the layer is of polygon type all the polygons that are within another polygon will have a distance of 0. If the distance to the polygons boundaries has to be calculated, convert first the polygon layer to a polyline one using Polygon To Polyline Wizard.
- The larger the search tolerance is, the slower the process will be
- The distance is calculated in the Spatial Reference of the input dataset.
- If there are more than one feature with the same distance to a feature (for example intersecting polylines) only one of the ID's will be recorded in the [ET_Closest] field.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ClosestFeatureDistance
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer
<Search Tolerance>	A Double representing the Search Tolerance (in the units of the spatial reference of the input layer)

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ClosestFeatureDistance", "input dataset", "output dataset", "Search Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ClosestFeatureDistance" "input dataset" "output dataset" "Search Tolerance"</code>
.NET using ETGWOuX.dll	<code>ClosestFeatureDistance(input dataset, output dataset, Search Tolerance)</code>
ArcPy	<code>arcpy.ClosestFeatureDistance(input dataset, output dataset, "Search Tolerance")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Select and Export

[Running programmatically](#)

Exports the features of a dataset to a new layer based on an attribute query.

Inputs:

- A layer
- SQL Expression. The expression builder can be used if executed from the ET GeoWizards interface.

Outputs:

- New layer
 - The output layer will contain only the selected features of the input dataset
 - The attributes of the input data set are preserved.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	SelectAndExport
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<SQL Expression>	A String representing the selection expression. Example: Shape_Area < 200 AND Name = 'a'

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "SelectAndExport", "input dataset", "output dataset", "SQL Expression"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "SelectAndExport" "input dataset" "output dataset" "SQL Expression"</code>
.NET using ETGWOuX.dll	<code>SelectAndExport(input dataset, output dataset, SQL Expression)</code>
ArcPy	<code>arcpy.SelectAndExport(input dataset, output dataset, "SQL Expression")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Project Layer

[Running programmatically](#)

Project the input layer to the projection of the reference layer

Inputs:

- Input Layer - a Point, Multipoint, Polyline or Polygon layer
- Reference Layer - a polygon layer which features will be used for clipping

Outputs:

- New layer (Point, Multipoint, Polyline or Polygon depending on the type of the input layer)
 - The attributes are preserved
 - The features of the input layer projected to the spatial reference of the reference layer

Notes:

- It is highly recommended the spatial reference of the input layer to have the same Geographic Coordinate System as the spatial reference of the reference layer.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	ProjectLayer
<input dataset>	A String representing the input layer.

<Reference Layer>	A String representing the reference layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "ProjectLayer", "input dataset", "Reference Layer", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "ProjectLayer" "input dataset" "Reference Layer" "output dataset"</code>
.NET using ETGWOutX.dll	<code>ProjectLayer(input dataset, Reference Layer, output dataset)</code>
ArcPy	<code>arcpy.ProjectLayer(input dataset, Reference Layer, "output dataset")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Delete Multiple Fields

[Running programmatically](#)

Deletes multiple fields from a layer.

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
 - Multipoint
- Fields to be deleted

Outputs:

- A new dataset.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	DeleteMultipleFields
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

<Field List>	A String representing a list of field names (separated by ";") to be deleted.
--------------	---

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "DeleteMultipleFields", "input dataset", "output dataset", "Field List"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "DeleteMultipleFields" "input dataset" "output dataset" "Field List"</pre>
.NET using ETGWOutX.dll	<code>DeleteMultipleFields(input dataset, output dataset, Field List)</code>
ArcPy	<code>arcpy.DeleteMultipleFields(input dataset, output dataset, "Field List")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Order Fields

[Running programmatically](#)

Exports a feature layer to a new feature class. The user selects the fields to be exported and the order in which they will appear in the attribute table.

Inputs:

- A feature layer
 - Point
 - Multipoint
 - Polyline
 - Polygon
- Fields to be exported
- The order in which the fields will be added to the attribute table of the new feature class.

Outputs:

- A new feature class. The fields in the attribute table are permanently ordered.

How to use:

- Select a layer to be exported and a location for the new feature class
- A list of all the fields in the layer is presented in a list box. Using the arrow buttons move the fields to be exported to the order fields list box
- Use the Up and Down buttons to arrange the fields in the order you want them to appear in the attribute table.
- Click the Finish button

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	OrderFields
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Field List>	A String representing an ordered list of field names (separated by ";") to be copied in the output dataset.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "OrderFields", "input dataset", "output dataset", "Field List"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "OrderFields" "input dataset" "output dataset" "Field List"</code>
.NET using ETGWOutX.dll	<code>OrderFields(input dataset, output dataset, Field List)</code>
ArcPy	<code>arcpy.OrderFields(input dataset, output dataset, "Field List")</code>

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Redefine Fields

[Running programmatically](#)

Change field names and definitions.

Inputs:

- A feature layer
 - Point
 - Multipoint
 - Polyline
 - Polygon
- New field names, length, precision, scale

Outputs:

- A new layer.

How to use:

- Select a layer to be exported and a location for the new feature class
- A list of all the fields in the layer is presented in a grid where the user can change the names and definitions of the fields

Notes:

- The type of the fields can be changed, but if the values in the original field cannot be automatically converted to the type for the redefined field, the values will not be populated in the output.
- The new field names should be max 10 characters long if the output is a shapefile

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	RedefineFields
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Field List>	<p>A String representing a list of field names (separated by ";") together with their definition {OldName NewName NewType NewWidth NewPrecision}.</p> <p>Valid field types - STRING, INTEGER, LONG, SHORT, REAL, FLOAT, DOUBLE, DATE</p> <p>Example: "a b String 50;c d Integer 6;e f Double 8 3"</p>

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "RedefineFields", "input dataset", "output dataset", "Field List"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "RedefineFields" "input dataset" "output dataset" "Field List"</code>
.NET using ETGWOutX.dll	<code>RedefineFields(input dataset, output dataset, Field List)</code>
ArcPy	<code>arcpy.RedefineFields(input dataset, output dataset, "Field List")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Copy Fields

[Running programmatically](#)

Copies fields from one dataset to another.

Inputs:

- Input feature layer
- Source feature layer - the layer that will be used as a source for the field definitions

Outputs:

- A new layer. The attributes of the input layer will be preserved. The selected fields from the source layer will be copied to the attribute table of the output layer

How to use:

- Specify input and source layers
- Specify output layers
- Select fields from the source layer to be copied over and the order in which the field will be added to the attribute table of the target layer

Notes:

- The new fields will be added after the fields of the input layer
- If the input attribute table has a field with the same name as a field selected to be copied, the field name will be changed (suffix added) and added to the output

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	CopyFields
<input dataset>	A String representing the input layer.
<Source Dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
<Field List>	A String representing a list of the fields (separated with ";") to be copied.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CopyFields", "input dataset", "Source Dataset", "output dataset", "Field List"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CopyFields" "input dataset" "Source Dataset" "output dataset" "Field List"
.NET using ETGWOuX.dll	CopyFields(input dataset, Source Dataset, output dataset,Field List)
ArcPy	arcpy.CopyFields(input dataset, Source Dataset, output dataset,"Field List")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)

- The argument separator for `StartInfo.Arguments` is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create routes from existing polylines

[Running programmatically](#)

Creates routes by merging existing polylines that have the same common identifier.

Inputs

- A Polyline feature layer
- Route Identifier field
- Method for route creation
- Output Spatial Reference

Output:

- A PolylineM feature class. The polylines are measured depending on the method selected - based on the length of the polylines, the values of a single field or two fields (From Measure and To Measure)

Measuring Methods

- Using the lengths of the source polylines.
 - The user controls the direction of the routes by specifying the coordinate priority of the starting measure (see notes)
 - If there are spatial gaps between the polylines to be joined, the user specifies whether these gaps to be taken into account when assigning the measures (see notes)
- Using the values in a single numeric field
 - The user controls the direction of the routes by specifying the coordinate priority of the starting measure (see notes)
 - If there are spatial gaps between the polylines to be joined, the user specifies whether these gaps to be taken into account when assigning the measures (see

notes)

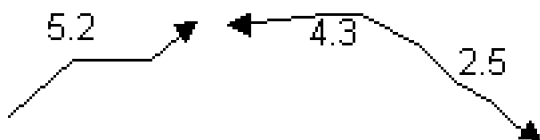
- Using known measures in two numeric fields. From Measure and To Measure.
 - Very important factor in this case is the orientation of the original polylines. The polylines must be oriented in the direction of increasing measure
 - Since known measures are used for each polyline, the Spatial Gaps parameter is not used when using this method

Notes:

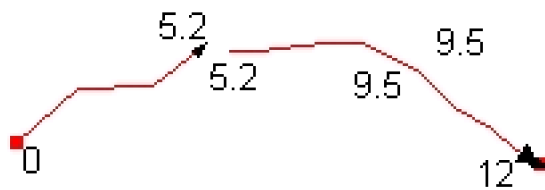
- Coordinate Priority (not used if the third method above is used). This parameter defines the direction of the output routes and the order in which the original polylines will participate in the route. The available options are
 - Lower Left ("ll")
 - Lower Right ("lr")
 - Upper Right ("ur")
 - Upper Left ("ul")

The options are determined by the minimum bounding rectangle for each route. If the "Lower Left" option is used the routes will start from South-West All original polylines will be oriented to go in North-East direction and the measures will increase in this direction.

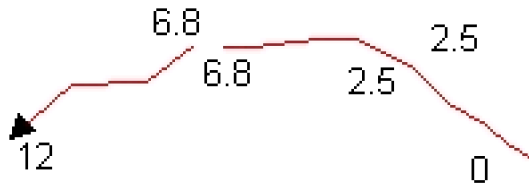
Original Polylines



Route - Coordinate Priority = "Lower Left"

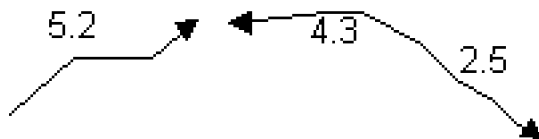


Route - Coordinate Priority = "Lower Right"

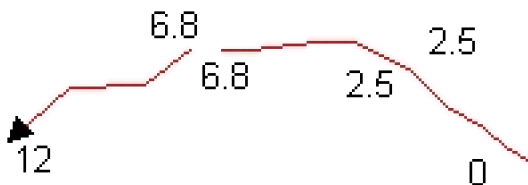


- Spatial Gaps: In many cases a route consists of disjointed parts A road for example that have the same name on both sides of a river might be represented by a single route. For such cases the user has to specify how the spatial gaps between the disjointed parts of the route will be handled when calculating the measures.

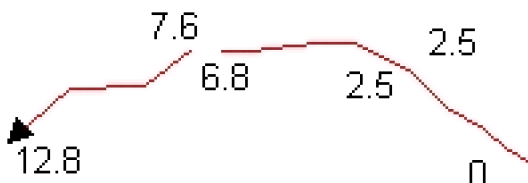
Original Polylines



Ignore Spatial Gaps option selected - Continuous measurements



Ignore Spatial Gaps option not selected - The gap distance incorporated into the measures. The straight-line distance between the disjointed nodes added to the measures



- The user can specify output spatial reference that is different from the projection of the input dataset. The Output Spatial Reference must have the same geographic coordinate system as the input dataset

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CreateRoutes
<input dataset>	A feature layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<ID Field>	A String - the name of the field which values will be used as Route Identifier.
<Measure From>	<p>A String indicating where the measures will be taken from (see above). Valid values:</p> <ul style="list-style-type: none">• "Length"• "SingleField"• "TwoFields"
{Measure Field1}	A String - a field name to be used for source of the measures (SingleField option) and From Measure (TwoFields option)
{Measure Field2}	A String - a field name to be used for source of the To Measure (TwoFields option only)
{Coordinate Priority}	<p>A String indicating the coordinate priority to be used (see above).Valid values:</p> <ul style="list-style-type: none">• "ll", "lowerleft", "sw", "south-west"• "lr", "lowerright", "se", "south-east"• "ul", "upperleft", "nw", "north-west"

	<ul style="list-style-type: none"> • "ur", "upperright", "ne", "north-east" • "b", "bottom", "s", "south" • "t", "top", "n", "north" • "l", "left", "w", "west" • "r", "right", "e", "east"
{Ignore Gaps}	A Boolean indicating whether the spatial gaps to be ignored(see above).
{Max Segments}	An integer indicating the maximum number of polylines to be included in a single route.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CreateRoutes", "input dataset", "output dataset", "ID Field", "Measure From", "Measure Field1", "Measure Field2", "Coordinate Priority", "Ignore Gaps", "Max Segments"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateRoutes" "input dataset" "output dataset" "ID Field" "Measure From" "Measure Field1" "Measure Field2" "Coordinate Priority" "Ignore Gaps" "Max Segments"
.NET using ETGWOuX.dll	CreateRoutes(input dataset, output dataset, ID Field, Measure From, Measure Field1, Measure Field2, Coordinate Priority, Ignore Gaps, Max Segments)
ArcPy	arcpy.CreateRoutes(input dataset, output dataset, "ID Field", "Measure From", "Measure Field1", "Measure Field2", "Coordinate Priority", "Ignore Gaps", "Max Segments")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Calibrate routes with points

[Running programmatically](#)

Adjusts route measures with existing points using measure information stored as attributes in the Point Attribute Table or the M values of pointM dataset. The calibration process inserts new vertices to the routes in the places where the calibration points intersect the routes. The measure value of these vertices is set to the measure value of the corresponding point. The measures of the existing vertices is adjusted according to the interpolation/extrapolation option selected and the adjustment method selected.

Inputs

- A PolylineM feature layer - to be calibrated
- Route Identifier field
- A Point or PointM feature layer - to be used for calibration
- Point Route Identifier field
- Point Measure field (only if the measures are to be taken from a field)
- Search tolerance - only the points that are closer to the route than this tolerance will be used for calibration
- Interpolation/Extrapolation options
- Adjustment method
- Output Spatial Reference

Output:

- A PolylineM feature class. The measures are adjusted based on the point dataset and the options selected by the user

Interpolation options:

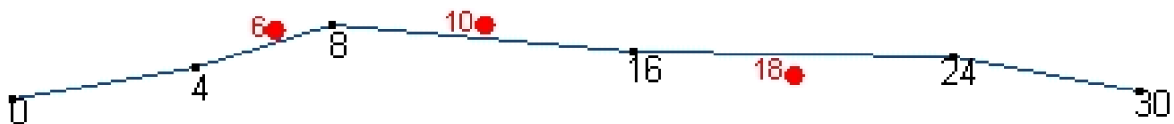
Three options are available. They can be used in any combination

- Extrapolate before calibration points - the measures of the preexisting vertices before the

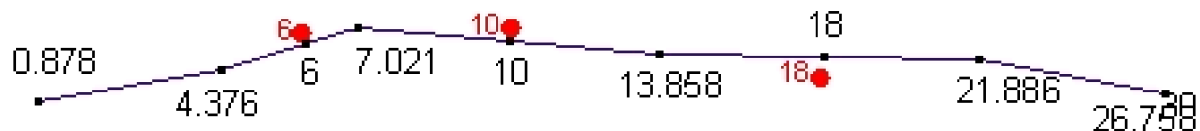
first calibration point will be adjusted

- Interpolate between calibration points - the measures of the preexisting vertices between the first and last calibration points will be adjusted
- Extrapolate after calibration points - the measures of the preexisting vertices after the last calibration point will be adjusted

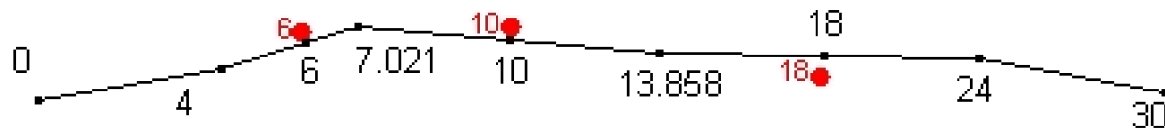
Original route and Calibration points



All options used for calibration. The measures of the vertices before, between and after the calibration points are adjusted



Only "Interpolate between" option used. The vertices before and after calibration points preserve their original measures



Adjustment methods:

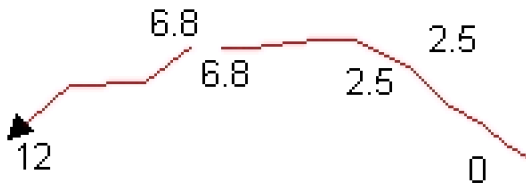
- Shortest path distance - the distance between the measure points is used to establish the calibration ratio. Then this ratio is applied to the preexisting vertices based on their distance to the calibration points.
- The existing measure distance - The measures of the calibration points are calculated based on the existing measures. These measures are compared to the new measures to establish the calibration ratio. Then this ratio is applied to the preexisting vertices based on their M values points.

Notes:

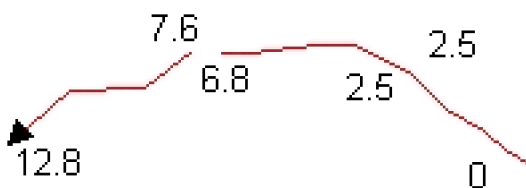
- Spatial Gaps: In many cases a route consists of disjointed parts. A road for example that has the same name on both sides of a river might be represented by a single route. For

such cases the user has to specify how the spatial gaps between the disjointed parts of the route will be handled when calculating the measures.

Ignore Spatial Gaps option selected - Continuous measurements



Ignore Spatial Gaps option not selected - The gap distance incorporated into the measures. The straight-line distance between the disjointed nodes added to the measures



- The user can specify output spatial reference that is different from the projection of the Routes Dataset. The Output Spatial Reference must have the same geographic coordinate system as the Routes Dataset and the calibration points dataset

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CalibrateRoutes
<Routes Dataset>	A String representing the input layer. Must be of PolylineM type.
<Calibration Dataset>	A String representing the calibration layer. Must be of Point type..
<output dataset>	A String - the full name of the output layer.

<Route Key Field>	A String representing a the Route Identifier field name in the Routes Dataset.
<Points Key Field>	A String representing a the Route Identifier field name in the Calibration Dataset.
<Calibration Method>	A String indicating the Calibration Method to be used. Valid strings: <ul style="list-style-type: none"> • "Distance" • "Measure"
<Search Tolerance>	A Double representing the search tolerance to be used (in the units of the spatial reference of the Route Dataset).
{Use M}	A Boolean indicating whether the M values of the Calibration Points to be used (only if the calibration dataset has M values
{M Field}	A String - the name of the field in the Calibration Dataset to be used for M calibration values (if Use M = FALSE)
{Extrapolate Before}	A Boolean indicating wheter to extrapolate before measure points (see explanation above).
{Interpolate Between}	A Boolean indicating wheter to interpolate between measure points (see explanation above).
{Extrapolate After}	A Boolean indicating wheter to extrapolate after measure points (see explanation above).
{Ignore Gaps}	A Boolean indicating whether the spatial gaps to be ignored(see above).

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CalibrateRoutes", "Routes Dataset",

	"Calibration Dataset", "output dataset", "Route Key Field", "Points Key Field", "Calibration Method", "Search Tolearance", "Use M", "M Field", "Extrapolate Before", "Interpolate Between", "Extrapolate After", "Ignore Gaps"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CalibrateRoutes" "Routes Dataset" "Calibration Dataset" "output dataset" "Route Key Field" "Points Key Field" "Calibration Method" "Search Tolearance" "Use M" "M Field" "Extrapolate Before" "Interpolate Between" "Extrapolate After" "Ignore Gaps"
.NET using ETGWOutX.dll	CalibrateRoutes(Routes Dataset, Calibration Dataset, output dataset, Route Key Field, Points Key Field, Calibration Method, Search Tolearance, Use M, M Field, Extrapolate Before, Interpolate Between, Extrapolate After, Ignore Gaps)
ArcPy	arcpy.CalibrateRoutes(Routes Dataset, Calibration Dataset, "output dataset", "Route Key Field", "Points Key Field", "Calibration Method", "Search Tolearance", "Use M", "M Field", "Extrapolate Before", "Interpolate Between", "Extrapolate After", "Ignore Gaps")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Google and Google Earth are trademarks of Google Inc

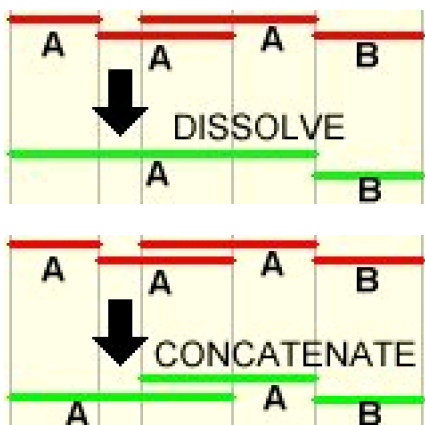
Copyright © lanko Tchoukanski

Dissolve/Concatenate Event Tables

[Running programmatically](#)

Both Dissolve and Concatenate functions combine records in an event table if the events are on the same route and have the same value in a specified field. The functions are available for line event layers only. The wizard allows the resulting data to be added to the map as a line event layer or a standalone table.

- Dissolve will combine the events if their measures overlap
- Concatenate will combine the events if the TO measure of one event is equal to the FROM measure of the next event



Input

- An line event layer
- A dissolve/concatenate field - the values of the records in this field will be used for dissolving/concatenating of the events

Output:

- A new event table (DBF or within File GDB) with the aggregated events
- The new table will contain all the original fields of the table of the input event layer

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	DissolveRouteEvents
Function Name	ConcatenateRouteEvents
<Event Table>	A String - the full path to the event table (DBF or table in File GDB).
<Output Table>	A String - the full path to the output table (DBF or table in File GDB).
<Key Field>	A String - the name of the route ID field in the event table .
<Dissolve Field>	A String - the name of the field in the input table to be used for Dissolve/Concatenate
<From Measure Field>	A String - the name of the From Measure field in the event table .
<To Measure Field>	A String - the name of the To Measure field in the event table.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Dissolve Events

Language	Syntax
Python	subprocess.call([ETGWPath, "DissolveRouteEvents", "Event Table", "Output Table", "Key Field", "Dissolve Field", "From Measure Field", "To Measure Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "DissolveRouteEvents" "Event Table" "Output Table" "Key Field" "Dissolve Field" "From Measure Field" "To Measure Field"

.NET using ETGWOtX.dll	DissolveRouteEvents(Event Table, Output Table, Key Field, Dissolve Field, From Measure Field, To Measure Field)
ArcPy	arcpy.DissolveRouteEvents(Event Table, Output Table, "Key Field", "Dissolve Field", "From Measure Field", "To Measure Field")

Concatenate Events

Language	Syntax
Python	subprocess.call([ETGWPath, "ConcatenateRouteEvents", "Event Table", "Output Table", "Key Field", "Dissolve Field", "From Measure Field", "To Measure Field"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "ConcatenateRouteEvents" "Event Table" "Output Table" "Key Field" "Dissolve Field" "From Measure Field" "To Measure Field"
.NET using ETGWOtX.dll	ConcatenateRouteEvents(Event Table, Output Table, Key Field, Dissolve Field, From Measure Field, To Measure Field)
ArcPy	arcpy.ConcatenateRouteEvents(Event Table, Output Table, "Key Field", "Dissolve Field", "From Measure Field", "To Measure Field")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

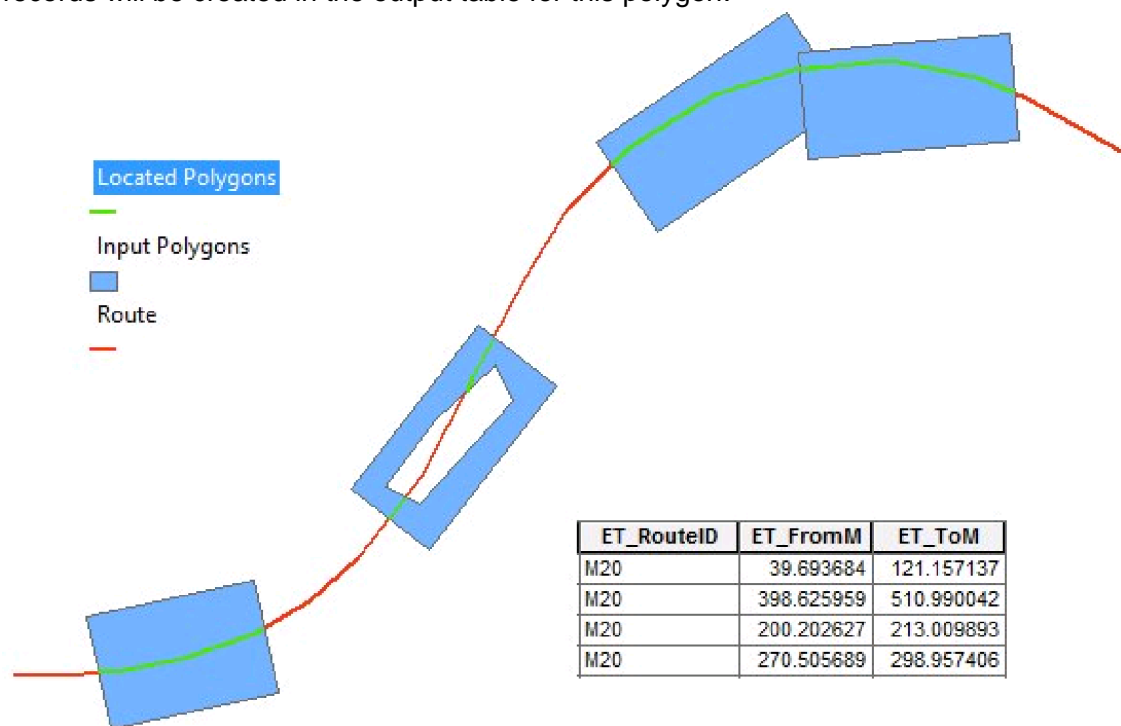
[\(Go to TOP\)](#)

Locate features features along routes

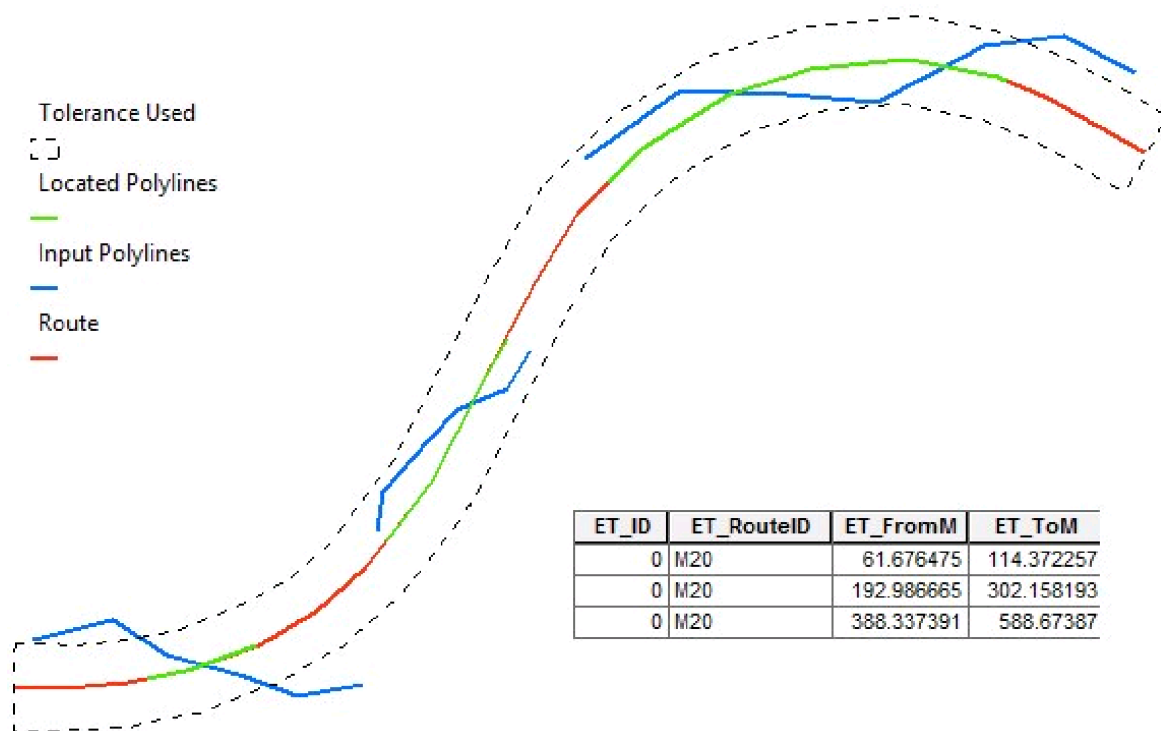
[Running programmatically](#)

Creates an event layer (table) from the input route dataset and a feature layer.

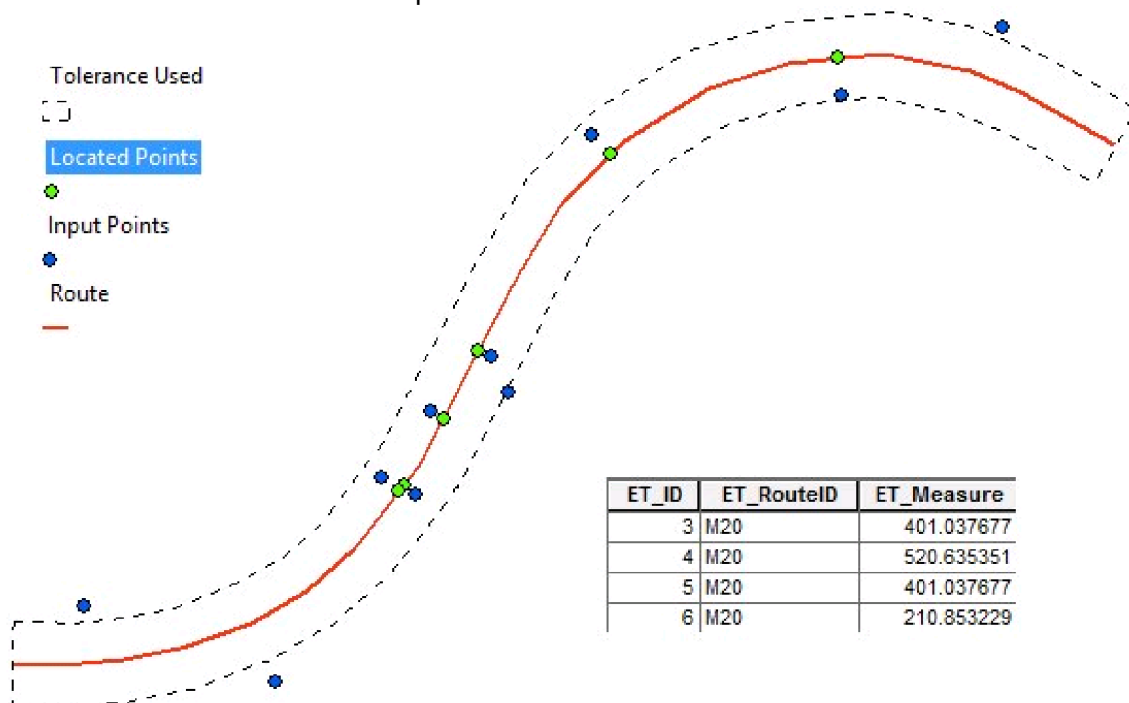
- Polygon - Finds the route and measure information at the geometric intersection of the input polygon layer and the route layer and creates a line event polyline layer(or event table). The output table will contain a route identifier, the FROM and TO measures of the route on which each polygon was located. If a polygon intersects more than one route multiple records will be created in the output table for this polygon.



- Polyline - Finds the route and measure information for the polylines or parts of them that are within the search tolerance from the routes and creates a polyline layer (or event table). The output table will contain a route identifier, the FROM and TO measures of the route on which each polyline was located. If a polyline or parts of it are within the search tolerance to more than one route multiple records will be created in the output table for this polyline.



- **Point** - Finds the route and measure information for the points from a Point layer and creates a point layer (or event table). Only the points that are within specified search tolerance of routes will be recorded in the output table. The output table will contain a route identifier and a measure for each point.



Inputs:

- A PolylineM feature layer - the routes to be used
- Route Identifier field - the values in this field will be recorded in the output event table
- A Point, Polyline or Polygon feature layer which features will be located on the routes
- Search Tolerance if Points or Polylines are to be located.

Output:

- A Point or Polyline layer or an event table(DBF or FileGdb) with event record for each event located on the route
- If the Add Attributes option is selected the output table will contain all the original fields of the input dataset
- Three/Two new fields will be added
 - [ET_RouteID] - the route identifier field. The values will correspond to the route on which each polygon was located
 - [ET_FromM] - the FROM measure of each event(Polygon and Polylines)
 - [ET_ToM] - the TO measure of each event (Polygon and Polylines)
 - [ET_Measure] - the measure of each event (Points)

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	LocateFeaturesAlongRoutes
<input	A feature layer. Must be of Point, Polyline or Polygon type.

dataset>	
<Route Layer>	A String - feature layer. Must be of PolylineM type.
<output dataset>	A String - the full name of the output layer.
<Route Key Field>	A String - the name of the route ID field in the route dataset
{Search Tolerance}	A Double - the search tolerance (for Point or Polyline input datasets). For Polygons use 0
{Otput Table}	A Boolean - if TRUE only event table will be stored, if FALSE a feature layer will be saved.
{Add Atributes}	A Boolean - if TRUE the attributes of the input dataset will be copied to the result.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "LocateFeaturesAlongRoutes", "input dataset", "Route Layer", "output dataset", "Route Key Field", "Search Tolerance", "Otput Table", "Add Atributes"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "LocateFeaturesAlongRoutes" "input dataset" "Route Layer" "output dataset" "Route Key Field" "Search Tolerance" "Otput Table" "Add Atributes"
.NET using ETGWOuX.dll	LocateFeaturesAlongRoutes(input dataset, Route Layer, output dataset, Route Key Field, Search Tolerance, Otput Table, Add Atributes)
ArcPy	arcpy.LocateFeaturesAlongRoutes(input dataset, Route Layer, "output dataset", "Route Key Field", "Search Tolerance", "Otput Table", "Add Atributes")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

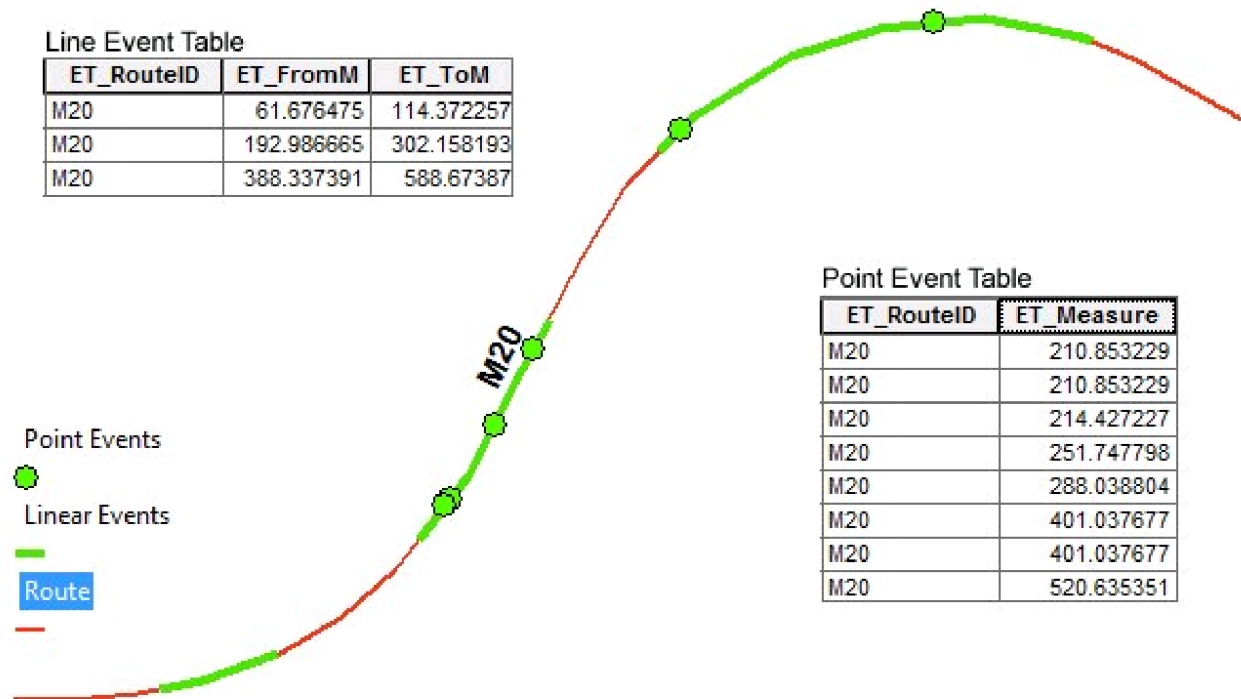
Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Create Features from Events

[Running programmatically](#)

Creates a feature layer from an event table (point or line) and a Input Table.



Inputs:

- A Point or Line event table

A point event table needs to have two fields

- [Route ID] - corresponding to the route ID in the rout dataset
- [Measure] - the measure of each point event

A line event table needs to have three fields

- [Route ID] - corresponding to the route ID in the rout dataset
- [From Measure] - the start measure of a linear event
- [To Measure] - the end measur of a linear event

- Route Dataset
- Event Type - Line or Point

Output:

- A Point or Polyline layer depending on the type of the events

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FeaturesFromRouteEvents
<Route Layer>	A String - feature layer. Must be of PolylineM type.
<Input Table>	A String - the full path to the event table (DBF or table in File GDB).
<output dataset>	A String - the full name of the output layer.
<Route Key Field>	A String - the name of the route ID field in the route dataset
<Event Type>	A String - the Event Type. Valid values: "Line" or "Point"
<Event Key>	A String - the name of the route ID field in the event table .
<From Measure Field>	A String - the name of the From Measure(for Line events) or Measure (for Point events) field in the event table .
{To Measure Field}	A String - the name of the To Measure(for Line events) field in the event table.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FeaturesFromRouteEvents", "Route Layer", "Input Table", "output dataset", "Route Key Field", "Event Type", "Event Key", "From Measure Field", "To Measure Field"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "FeaturesFromRouteEvents" "Route Layer" "Input Table" "output dataset" "Route Key Field" "Event Type" "Event Key" "From Measure Field" "To Measure Field"</code>
.NET using ETGWOuX.dll	<code>FeaturesFromRouteEvents(Route Layer, Input Table, output dataset, Route Key Field, Event Type, Event Key, From Measure Field, To Measure Field)</code>
ArcPy	<code>arcpy.FeaturesFromRouteEvents(Route Layer, Input Table, "output dataset", "Route Key Field", "Event Type", "Event Key", "From Measure Field", "To Measure Field")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Cogo Inverse

[Running programmatically](#)

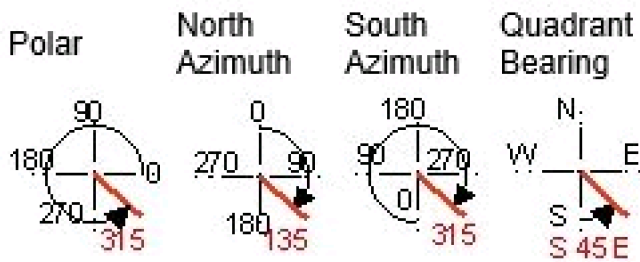
Converts a polyline or a polygon data set to a feature class containing only single segmented polylines. For each segment the COGO attributes are calculated and added to the resulting attribute table. The attributes of the original features are copied to the output features.

Inputs:

- A Polyline or Polygon feature layer

Options:

- Direction Angle Type - the type of the output angle for the direction of the segments



- Direction Angle Units
 - DD - Decimal Degrees
 - DMS - Degrees Minutes Seconds
 - Rad - Radians
 - Grad - Gradians (One gradian is equal to 1/400 circle)
 - Gon - Gons - used in some European countries (One gon is equal to 1/400 circle)
- Precision
 - Linear - integer indicating the number of places after the decimal point for the output linear measurements

- Angular - integer indicating the number of places after the decimal point for the output angular measurements

Outputs:

- New polyline feature class
- Attribute fields added to the attribute table of the output feature class
 - Always
 - [Direction] - the direction of the segment. The angle can be measured in arithmetic or geographic notation depending on the user choice
 - [Distance] - the length of the segment measured in the units of the original dataset
 - [Delta] - for circular arcs only. The central angle of the circular arc in degrees
 - [Radius] - for circular arcs only. The radius of the circular arc
 - [Tangent] - for circular arcs only. The distance from the Start/End points of the circular arc to the intersection point of the tangents
 - [ArcLength] - for circular arcs only. The length of the circular arc
 - [Delta] - for circular arcs only. The central angle of the circular arc
 - [Side] - the side of the circular arc compared with the tangent in the start point
 - User choice
 - [XStart] - X coordinate of the start point of the segment
 - [YStart] - Y coordinate of the start point of the segment

- [XEnd] - X coordinate of the end point of the segment
- [YEnd] -Y coordinate of the end point of the segment
- User choice - shapes with Z values only
 - [ZStart] - Z coordinate of the start point of the segment
 - [ZEnd] - Z coordinate of the end point of the segment
 - [Slope] - the slope of the segment in degrees (from -90 to 90)
- User choice - shapes with M values only
 - [MStart] - M coordinate of the start point of the segment
 - [MEnd] - M coordinate of the end point of the segment

Notes :

- The Z or M fields will be added only if the source dataset has Z/M values
- ET GeoWizards 12.0 cannot handle true arcs. If the source is File GDB make sure that there are no true arcs in the input layer.

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CogolInverse

<input dataset>	A String representing the input layer. Must be of Polyline or Polygon type.
<output dataset>	A String - the full name of the output layer.
<Point Coordinates>	A Boolean - indicates whether the coordinates of the Start and End point for each segment are to be added to the attribute table.
<Z Attributes>	A Boolean - indicates whether the Z/M coordinates of the Start and End point for each segment are to be added to the attribute table.
<Direction Type>	<p>Required. A String indicating the direction type to be used. Valid values:</p> <ul style="list-style-type: none"> • "NorthAzimuth", "NA" • "SouthAzimuth", "SA" • "Polar" • "QuadrantBearing", "QB"
<Direction Units>	<p>Required. A String indicating the direction units to be used. Valid values:</p> <ul style="list-style-type: none"> • "DecimalDegrees", "DD" • "DegreesMinutesSeconds", "DMS" • "Radians" • "Gradians" • "Gons", "QB"
{Linear Precision}	An Integer representing the number of places after the decimal point for the output linear measurements
{Angular	An Integer representing the number of places after the decimal point for the

Precision}	output angular measurements.
------------	------------------------------

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "CogolInverse", "input dataset", "output dataset", "Point Coordinates", "Z Attributes", "Direction Type", "Direction Units", "Linear Precision", "Angular Precision"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "CogolInverse" "input dataset" "output dataset" "Point Coordinates" "Z Attributes" "Direction Type" "Direction Units" "Linear Precision" "Angular Precision"</code>
.NET using ETGWOutX.dll	<code>CogolInverse(input dataset, output dataset, Point Coordinates, Z Attributes, "Direction Type", Direction Units, "Linear Precision", Angular Precision)</code>
ArcPy	<code>arcpy.CogolInverse(input dataset, output dataset, "Point Coordinates", "Z Attributes", "Direction Type", "Direction Units", "Linear Precision", "Angular Precision")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

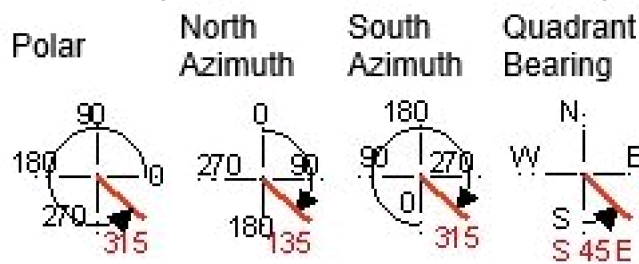
Lines from Points Direction and Distance

[Running programmatically](#)

Creates single segmented polylines from a point dataset that has in the attribute table fields which values represent direction and distance from each point to the target point.

Inputs:

- A Point dataset
- Direction Field - a field in the attribute table that has the values for the directions of the lines to be created
- Distance Field - a field in the attribute table that has the values for the distances (length) of the lines to be created
- Direction Angle Type - the type of the output angle for the direction of the segments



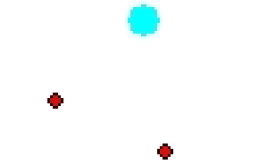
- Direction Angle Units
 - DD - Decimal Degrees
 - DMS - Degrees Minutes Seconds
 - Rad - Radians
 - Grad - Gradians (One gradian is equal to 1/400 circle)
 - Gon - Gons - used in some European countries (One gon is equal to 1/400 circle)

Outputs:

- New polyline layer
- The attributes of the input point features are preserved

Example :

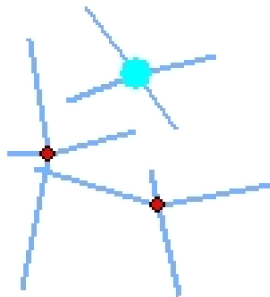
Input Points



Point Attribute Table

FI	Shap	Direction	Distance
0	Point	N 77-45-13.20 E	1064.4766
1	Point	S 35-46-12.41 E	924.303
2	Point	S 68-17-9.83 W	980.8273
3	Point	N 37-2-10.37 W	1111.2219
4	Point	S 12-34-11.25 E	1222.7778
5	Point	N 73-32-2.71 W	1706.9954

Resulting Polylines



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	LinesFromPointDirDist
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.
<Direction Field>	Required. A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the directions of the lines to be created.
<Distance Field>	A String representing the name of a field in the in the attribute table of the input dataset field name. The field has the values for the distances of the lines to be created.
<Direction Type>	<p>Required. A String indicating the direction type to be used. Valid values:</p> <ul style="list-style-type: none"> • "NorthAzimuth", "NA" • "SouthAzimuth", "SA" • "Polar" • "QuadrantBearing", "QB"
<Direction Units>	<p>Required. A String indicating the direction units to be used. Valid values:</p> <ul style="list-style-type: none"> • "DecimalDegrees", "DD" • "DegreesMinutesSeconds", "DMS" • "Radians" • "Gradians" • "Gons", "QB"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "LinesFromPointDirDist", "input dataset", "output dataset", "Direction Field", "Distance Field", "Direction Type", "Direction Units"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "LinesFromPointDirDist" "input dataset" "output dataset" "Direction Field" "Distance Field" "Direction Type" "Direction Units"</code>
.NET using ETGWOutX.dll	<code>LinesFromPointDirDist(input dataset, output dataset, Direction Field, Distance Field, "Direction Type", Direction Units)</code>
ArcPy	<code>arcpy.LinesFromPointDirDist(input dataset, output dataset, "Direction Field", "Distance Field", "Direction Type", "Direction Units")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Points Along Polylines

[Running programmatically](#)

Creates points along the polylines of the input dataset.

- The points are located on user specified relative distance from the start point of the polylines.
- The user can specify an offset distance and on which side of the polylines the offset points will be created.
- If "Both" option is selected for each polyline will be created 2 points (one on the left and one on the right side), otherwise one point per polyline will be created.

Inputs:

- A polyline feature layer
- Relative distance along polylines. A value between 0 and 1 indicating the distance from the from point as a ratio.
 - 0 indicates the start point
 - 0.5 indicates a point in the middle of the polyline
 - 1 indicates the end of the polyline
- Side of the points - three options are available
 - Both - 2 points will be created on both sides of the polylines
 - Left - one point per polyline will be created and will be located on the left side of the polylines
 - Right - one point per polyline will be created and will be located on the right side of the polylines
- Offset -a distance from the polyline for the points to be created. If not specified, the points

will be on the polylines

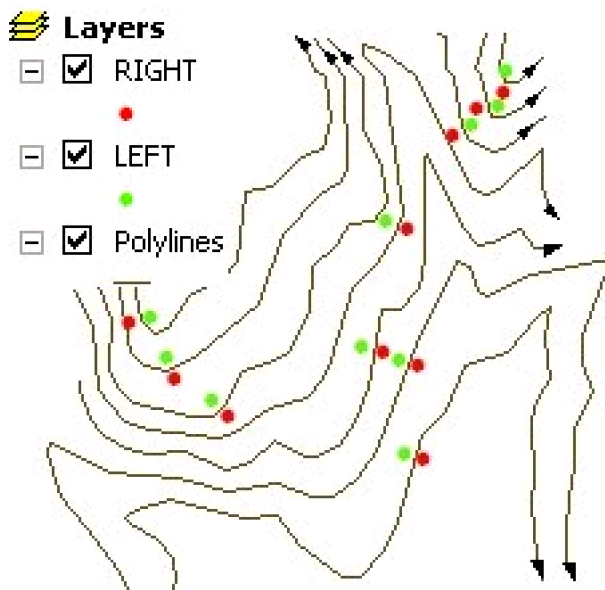
Outputs:

- New Point feature class with one or two (depending on the Side option) points per polyline
- The attributes of the original polylines are preserved
- The following fields are added to the point attribute table
 - [ET_ID] - the FID of original polylines.
 - [ET_Along] - the distance from the start point of the polyline to the point created.
 - [ET_Offset] - the distance of the point created to the corresponding polyline.

Notes:

- The offset is measured in the units of the spatial reference of the input dataset
- The output spatial reference is the one of the input polyline dataset

Examples:



Two Point datasets created

RIGHT:

- Side = "Right"
- Distance Along = 0.5
- Offset Distance = 50 feet

LEFT:

- Side = "Left"
- Distance Along = 0.5
- Offset Distance = 50 feet

Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsAlongPolylines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Relative Distance>	A Double representing distance between the station lines.

<Side>	<p>Required. A String -This parameter defines on which side of the polyline the station lines will be created:</p> <ul style="list-style-type: none"> • "Both" • "Left". • "Right"
{Offset}	A Double representing the Offset of the points created from the input polylines.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PointsAlongPolylines", "input dataset", "output dataset", "Relative Distance", "Side", "Offset From"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsAlongPolylines" "input dataset" "output dataset" "Relative Distance" "Side" "Offset"
.NET using ETGWOuX.dll	PointsAlongPolylines(input dataset, output dataset, Relative Distance, Side, Offset)
ArcPy	arcpy.PointsAlongPolylines(input dataset, output dataset , "Relative Distance", "Side", "Offset")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space,

you need to double quote it.

[\(Go to TOP\)](#)

Copyright © Ianko Tchoukanski

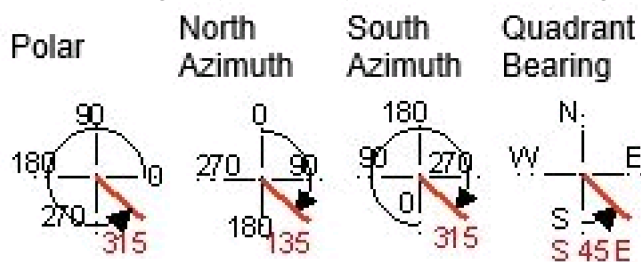
Points To Pie Segments

[Running programmatically](#)

Creates Pie segment polygons from points, pie direction, radius and central angle.

Inputs:

- A Point dataset
- Direction Field - a field in the attribute table that has the values for the directions of the circular segments to be created. The direction defines the central radius of the segment.
- Distance Field - a field in the attribute table that has the values for the radius of the circular segments to be created
- Central Angle Field - a field in the attribute table that has the values for the central angle of the pie circular segments to be created
- Direction Angle Type - the type of the output angle for the direction of the segments



- Direction Angle Units
 - DD - Decimal Degrees
 - DMS - Degrees Minutes Seconds
 - Rad - Radians
 - Grad - Gradians (One gradian is equal to 1/400 circle)
 - Gon - Gons - used in some European countries (One gon is equal to 1/400 circle)

Outputs:

- New polygon layer
- The attributes of the input point features are preserved

Example :

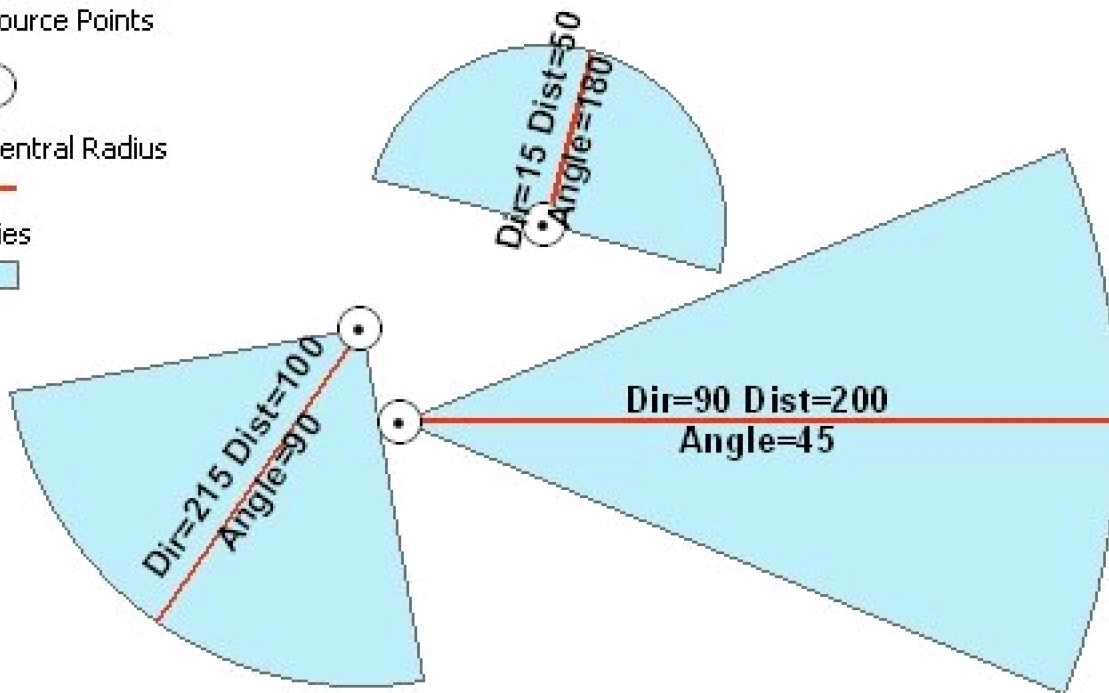
Source Points



Central Radius



Pies



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PointsToPieSegments
<input dataset>	A String representing the input layer. Must be of Point type.
<output dataset>	A String - the full name of the output layer.

<Direction Field>	Required. A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the directions of the pies to be created.
<Distance Field>	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the distances of the pies to be created.
<Angle Field>	A String representing the name of a field in the in the attribute table of the input dataset. The field has the values for the central angle of the pies to be created.
<Direction Type>	<p>Required. A String indicating the direction type to be used. Valid values:</p> <ul style="list-style-type: none"> • "NorthAzimuth", "NA" • "SouthAzimuth", "SA" • "Polar" • "QuadrantBearing", "QB"
<Direction Units>	<p>Required. A String indicating the direction units to be used. Valid values:</p> <ul style="list-style-type: none"> • "DecimalDegrees", "DD" • "DegreesMinutesSeconds", "DMS" • "Radians" • "Gradians" • "Gons", "QB"

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPath, "PointsToPieSegments", "input dataset", "output dataset", "Direction Field", "Distance Field", "Angle Field", "Direction Type", "Direction Units"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PointsToPieSegments" "input dataset" "output dataset" "Direction Field" "Distance Field" "Angle Field" "Direction Type" "Direction Units"</code>
.NET using ETGWOutX.dll	<code>PointsToPieSegments(input dataset, output dataset, Direction Field, Distance Field, Angle Field, "Direction Type", Direction Units)</code>
ArcPy	<code>arcpy.PointsToPieSegments(input dataset, output dataset, "Direction Field", "Distance Field", "Angle Field", "Direction Type", "Direction Units")</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Features To Convex Polygons

[Running programmatically](#)

Creates a convex polygon from each feature in the input feature class. Attributes of the original features are transferred to the convex polygons.

Inputs:

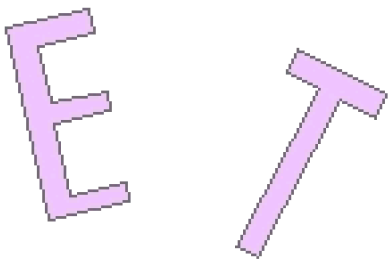
- A Polyline, Polygon or Multipoint feature class

Outputs:

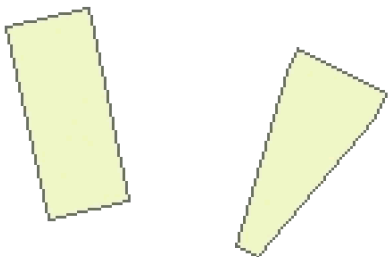
- A polygon feature class. All attributes of the original features are preserved

Examples:

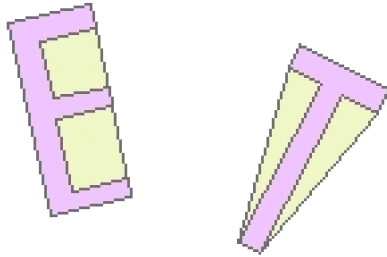
Input Dataset



Result Dataset



Overlay



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FeaturesToConvexPolygons
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FeaturesToConvexPolygons", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "FeaturesToConvexPolygons" "input dataset" "output dataset"</pre>
.NET using ETGWOuX.dll	<code>FeaturesToConvexPolygons(input dataset, output dataset)</code>
ArcPy	<code>arcpy.FeaturesToConvexPolygons(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Features To Envelopes

[Running programmatically](#)

Creates a polygon from the envelope of each feature in the input feature class. Attributes of the original features are transferred to the envelope polygons.

Inputs:

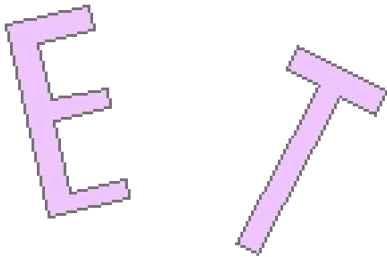
- A Polyline, Polygon or Multipoint feature class
- Expand distance. A distance in the units of the spatial reference of the input dataset with which the envelope of each feature will be expanded. The parameter is optional. The default value is 0.

Outputs:

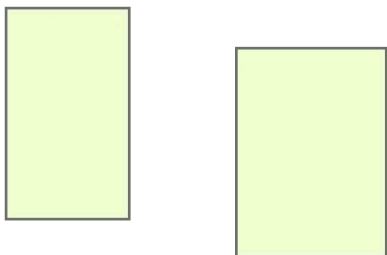
- A polygon feature class. All attributes of the original features are preserved

Examples:

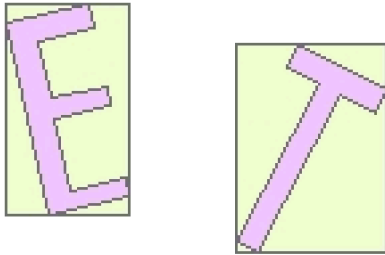
Input Dataset



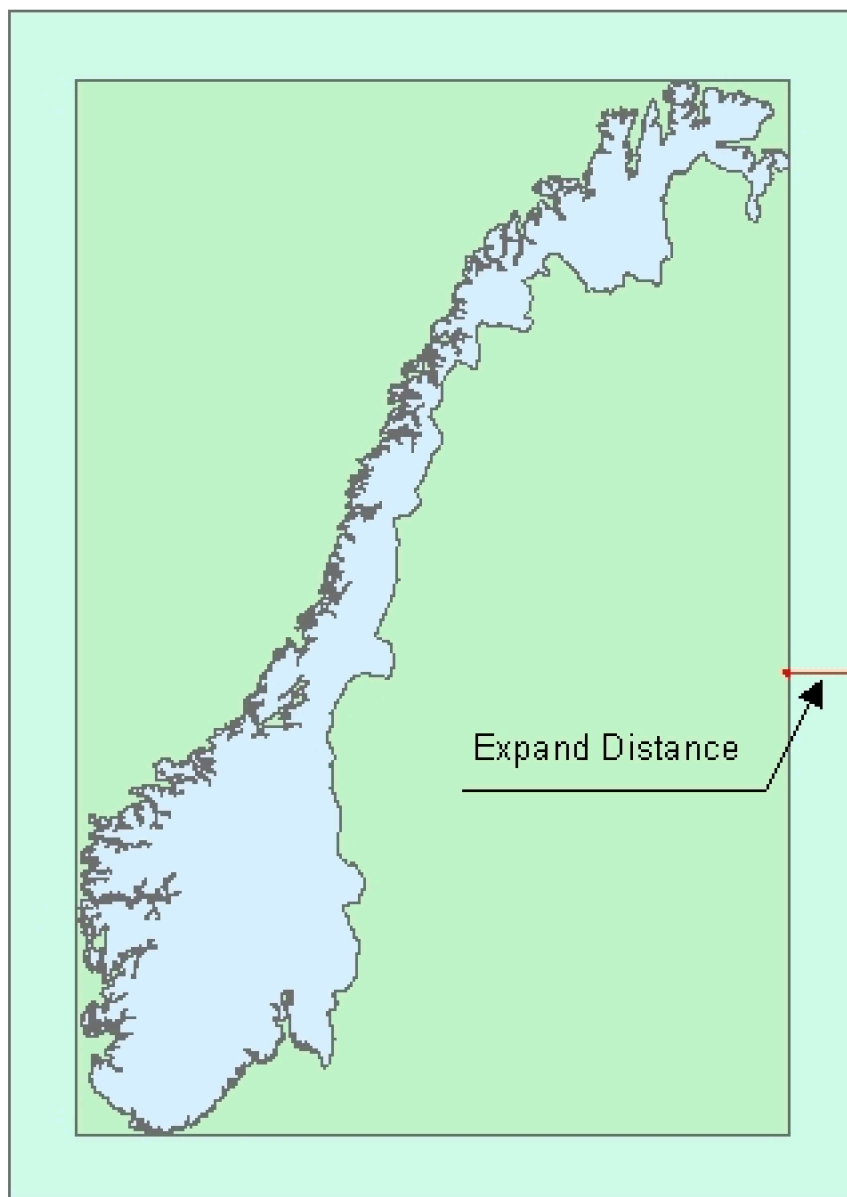
Result Dataset



Overlay



Expand parameter



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FeaturesToEnvelopes
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
{Expand Tolerance}	A double - distance in the units of the spatial reference of the input dataset with which the envelope of each feature will be expanded. The parameter is optional. The default value is 0 - no expand.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FeaturesToEnvelopes", "input dataset", "output dataset", "Expand Tolerance"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath StartInfo.Arguments = "FeaturesToEnvelopes" "input dataset" "output dataset" "Expand Tolerance"</code>
.NET using ETGWOuX.dll	<code>FeaturesToEnvelopes(input dataset, output dataset,Expand Tolerance)</code>
ArcPy	<code>arcpy.FeaturesToEnvelopes(input dataset, output dataset,Expand Tolerance)</code>

Notes:

- <> - required parameter
- {} - optional parameter

- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Features To Minimum Bounding Circles

[Running programmatically](#)

Creates a circular bounding polygon from each feature in the input feature class. Attributes of the original features are transferred to the resulting polygons.

Inputs:

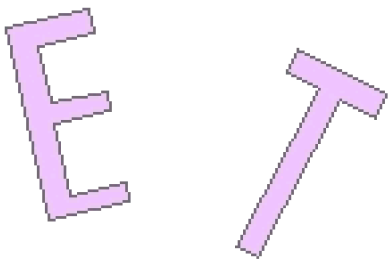
- A Polyline, Polygon or Multipoint dataset

Outputs:

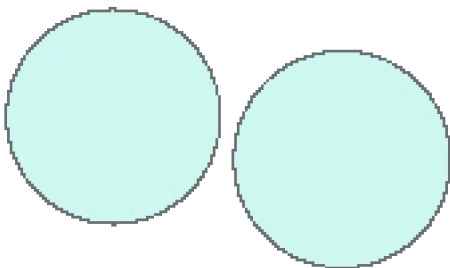
- A polygon feature class. All attributes of the original features are preserved

Examples:

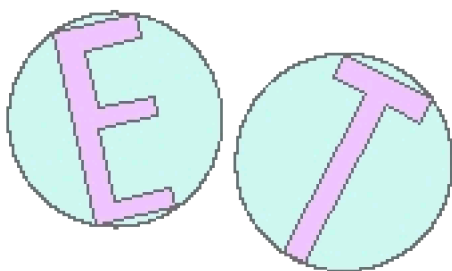
Input Dataset



Result Dataset



Overlay



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	FeaturesToCircles
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "FeaturesToCircles", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPath</code> <code>StartInfo.Arguments = "FeaturesToCircles" "input dataset" "output dataset"</code>
.NET using ETGWOutX.dll	<code>FeaturesToCircles(input dataset, output dataset)</code>
ArcPy	<code>arcpy.FeaturesToCircles(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Features To Minimum Bounding Rectangles

[Running programmatically](#)

Creates a bounding rectangle from each feature in the input feature class. Three ways to align the rectangles are available. Attributes of the original features are transferred to the resulting polygons.

Inputs:

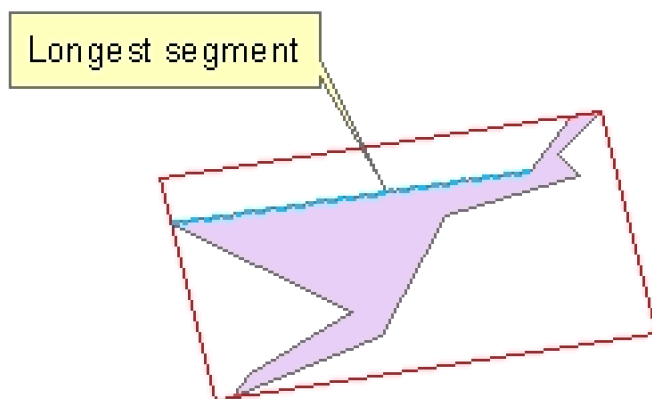
- A Polyline or Polygon feature class
- The orientation of the rectangles to be created
 - Along the longest segment of the polygon boundary
 - Along the longest axis of the original polygons
 - Minimum area rectangle

Outputs:

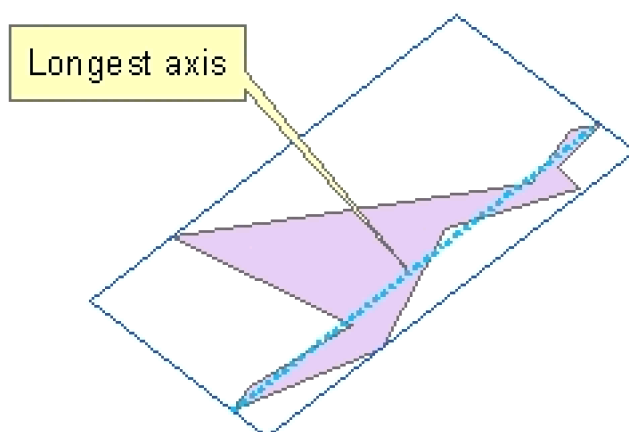
- A polygon feature class. All attributes of the original features are preserved
- New fields added to the attribute table
 - ET_Length - the length longest side of the bounding rectangle in the units of the Spatial Reference of the input feature class
 - ET Width - the length shortest side of the bounding rectangle in the units of the Spatial Reference of the input feature class

Examples:

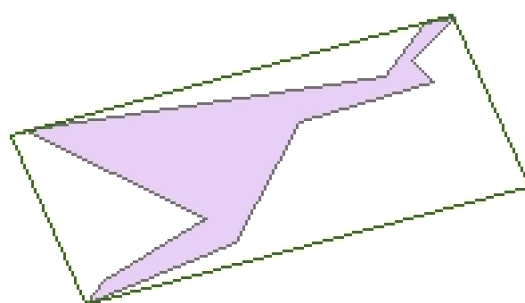
Bounding rectangle aligned with the longest segment of the boundary of the input polygon



Bounding rectangle aligned with the longest axis of the boundary of the input polygon



Minimum area bounding rectangle



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
------------	-------------

Function Name	FeaturesToRectangles
<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.
{Alignment}	<p>A String defining the orientation of the rectangles to be created</p> <ul style="list-style-type: none"> • "LongestSegment" - aligns the rectangle along the longest segment of the polygon boundary • "LongestAxis" - aligns the rectangle along the longest axis of the original polygons • "MinArea" - Minimum area rectangle

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "FeaturesToRectangles", "input dataset", "output dataset", "Alignment"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "FeaturesToRectangles" "input dataset" "output dataset" "Alignment"
.NET using ETGWOuX.dll	FeaturesToRectangles(input dataset, output dataset,Alignment)
ArcPy	arcpy.FeaturesToRectangles(input dataset, output dataset,Alignment)

Notes:

- <> - required parameter

- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygons to Equal Area Circles

[Running programmatically](#)

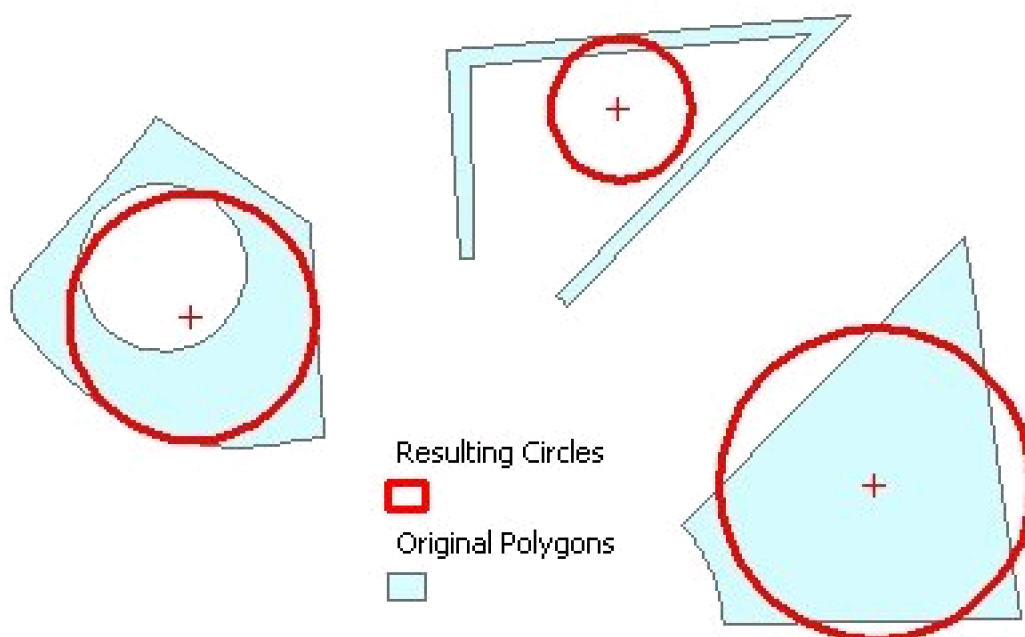
Creates a circular polygon with equal area for each polygon from the input feature class. The center of the circle is located in the centroid of the original polygon. The attributes of the original features are transferred to the resulting polygons.

Inputs:

- A Polygon feature layer.

Outputs:

- New polygon layer.



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToEqualAreaCircles

<input dataset>	A String representing the input layer.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	<code>subprocess.call([ETGWPath, "PolygonsToEqualAreaCircles", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<pre>StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonsToEqualAreaCircles" "input dataset" "output dataset"</pre>
.NET using ETGWOutX.dll	<code>PolygonsToEqualAreaCircles(input dataset, output dataset)</code>
ArcPy	<code>arcpy.PolygonsToEqualAreaCircles(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygons to Maximum Inscribed Circles

[Running programmatically](#)

Creates from each polygon in the input feature class a circular polygon representing the circle with maximum radius that can be inscribed in the input polygon. The center of the is located in the "deepest" point of each polygon. Attributes of the original features are transferred to the resulting polygons.

Inputs:

- A Polygon feature layer

Outputs:

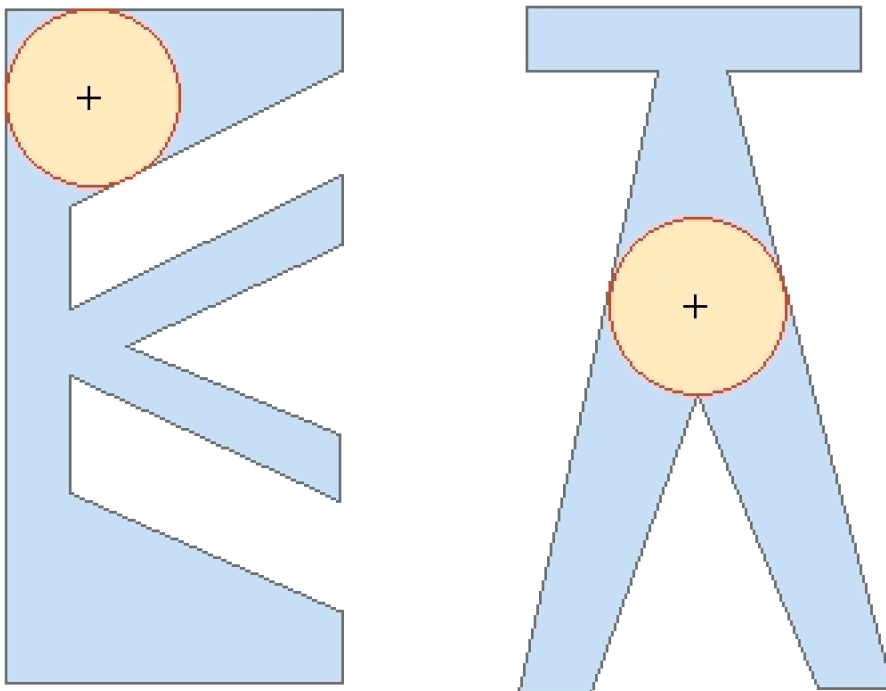
- A polygon layer. All attributes of the original features are preserved
- A new field [ET_Radius] is added and populated for each polygon.

Note:

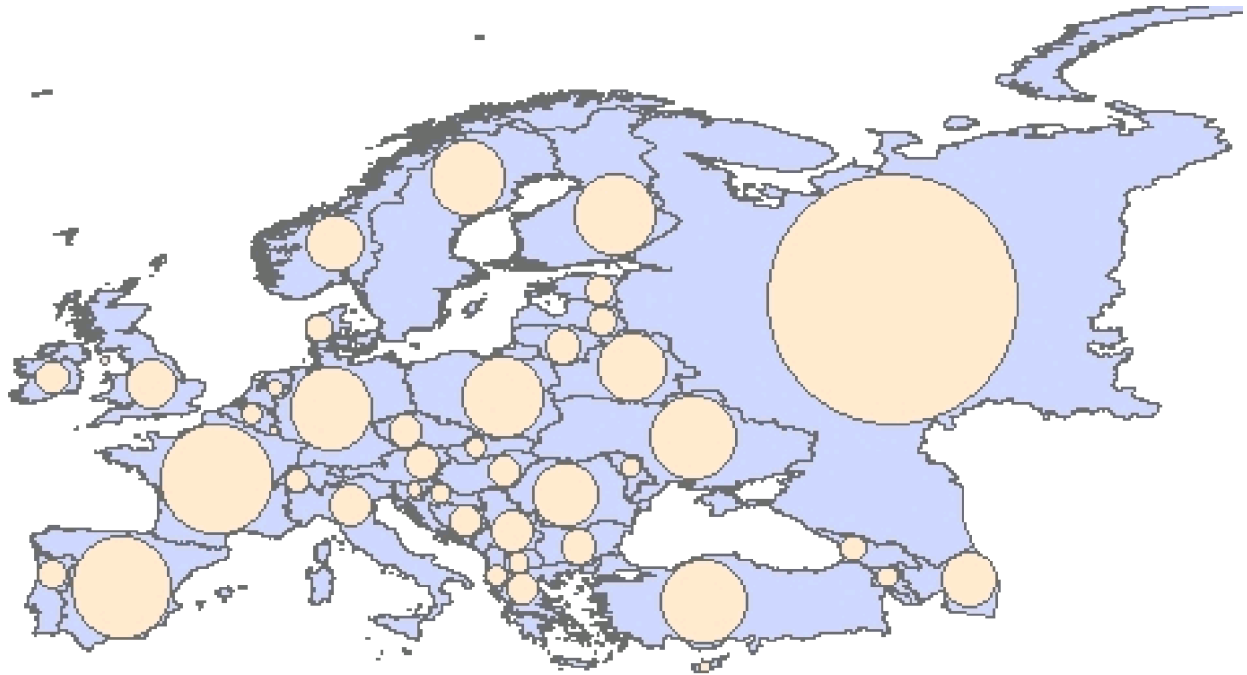
The function uses an interpolation algorithm and the precision of the calculation might not be 100%

Examples:

Example 1



Example 2 - Europe



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToMaxInscribedCircles
<input dataset>	A String representing the input layer. Must be of polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
----------	--------

Python	<code>subprocess.call([ETGWPPath, "PolygonsToMaxInscribedCircles", "input dataset", "output dataset"])</code>
.NET using ETGWRun.exe	<code>StartInfo.FileName = ETGWPPath StartInfo.Arguments = "PolygonsToMaxInscribedCircles" "input dataset" "output dataset"</code>
.NET using ETGWOutX.dll	<code>PolygonsToMaxInscribedCircles(input dataset, output dataset)</code>
ArcPy	<code>arcpy.PolygonsToMaxInscribedCircles(input dataset, output dataset)</code>

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Polygons to Deepest Points

[Running programmatically](#)

Creates from each polygon in the input feature class an inside point located farthest from the polygons boundary. Attributes of the original features are transferred to the resulting polygons.

Inputs:

- A Polygon feature layer

Outputs:

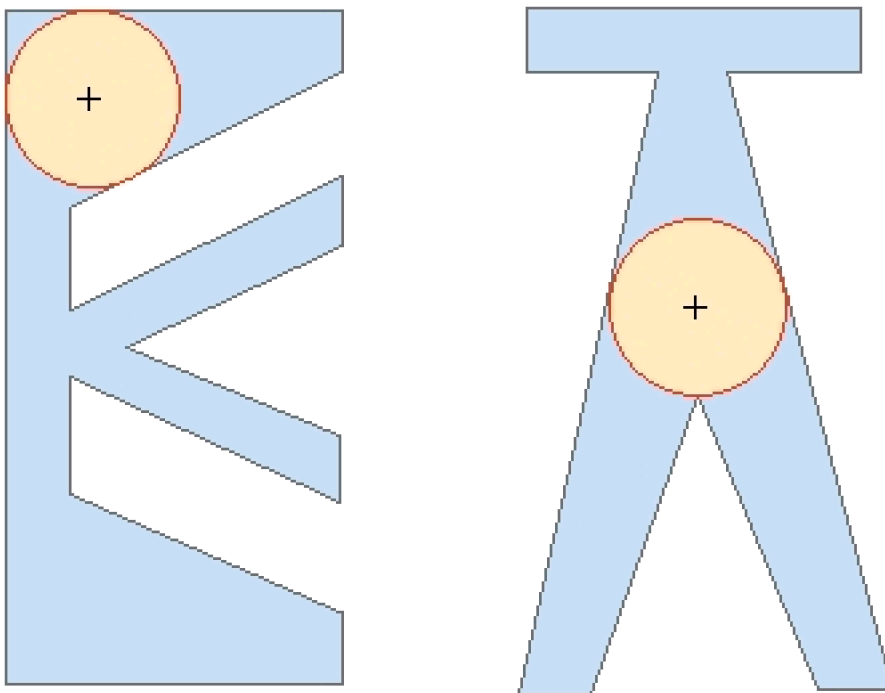
- A point layer. All attributes of the original features are preserved
- A new field [ET_Depth] is added and populated for each polygon.

Note:

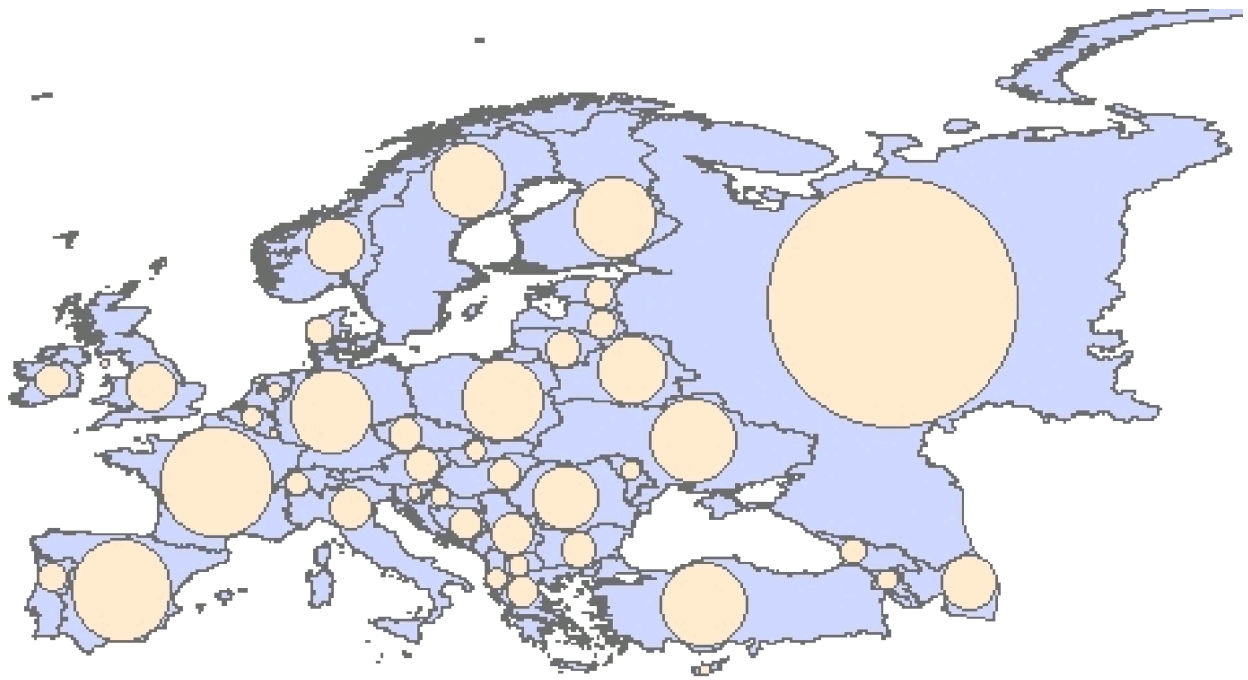
The function uses an interpolation algorithm and the precision of the calculation might not be 100%

Examples (the center of the circle in the images below are the "deepest" points for each polygon):

Example 1



Example 2 - Europe



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	PolygonsToDeepestPoints
<input dataset>	A String representing the input layer. Must be of polygon type.
<output dataset>	A String - the full name of the output layer.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "PolygonsToDeepestPoints", "input dataset", "output dataset"])

.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "PolygonsToDeepestPoints" "input dataset" "output dataset"
.NET using ETGWOtX.dll	PolygonsToDeepestPoints(input dataset, output dataset)
ArcPy	arcpy.PolygonsToDeepestPoints(input dataset, output dataset)

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Create Station Lines

[Running programmatically](#)

Creates equally spaced lines along the polylines from the input dataset. The station lines are single segmented polylines perpendicular to the input polylines (at the location of the station).

Inputs:

- A polyline feature layer
- Distance between stations
- Side of the station lines - three options are available
 - Both - the middle of the station lines will intersect the original polylines
 - Left - station lines will be located on the left side of the polylines
 - Right - station lines will be located on the right side of the polylines
- Length of the station lines can be specified
 - Constant - all the station lines will have the same user specified length
 - M Values - the M value of the input polylines (at the location of the station) will be used for length of the station lines. The input polylines must have M values.
 - Z Values - the Z value of the input polylines (at the location of the station) will be used for length of the station lines. The input polylines must have Z values.

Outputs:

- New Polyline layer with single segmented polylines perpendicular to the input polylines, distributed along the input polylines based on the user selected options.
- The attributes of the original polylines are preserved

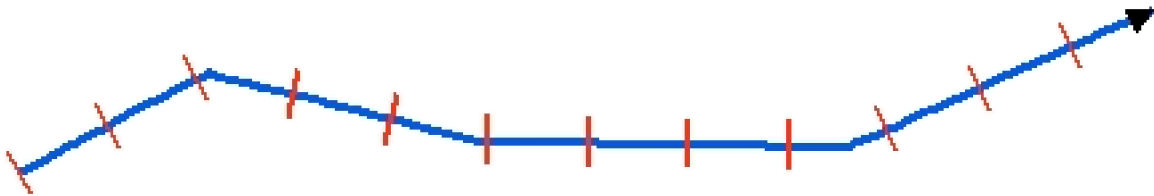
- The following fields are added to the output attribute table
 - [ET_ID] - the FID of original polylines.
 - [ET_Angle] - the angle of the polyline at the station.
 - [ET_Station] - the distance from the start point of the polyline to the station line
 - [ET_Length] - the length of the station line

Notes:

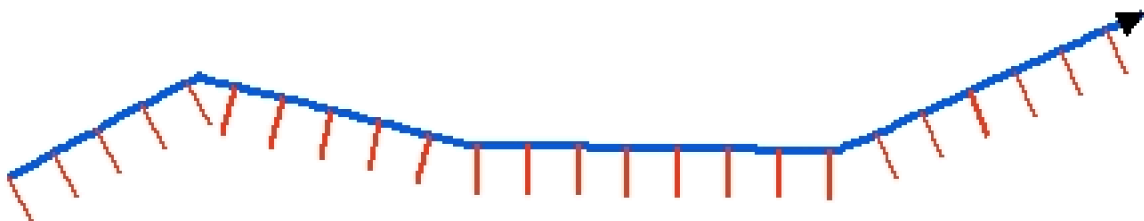
- The distance is measured in the units of the spatial reference of the input dataset
- The output spatial reference is the one of the input polyline dataset

Examples:

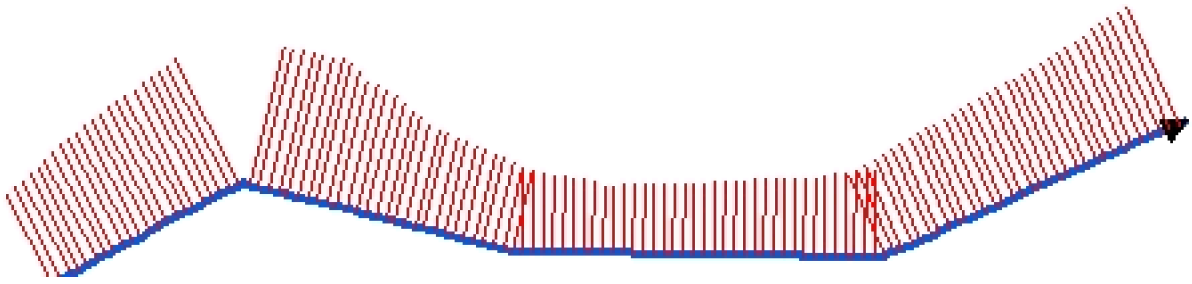
Side = "Both", Step = 200 meters, Constant Length = 100 meters



Side = "Right", Step = 100 meters, Constant Length = 100 meters



Side = "Left", Step = 20 meters, Length from Z values



Running Programmatically

[\(Go to TOP\)](#)

Parameters

Expression	Explanation
Function Name	CreateStationLines
<input dataset>	A String representing the input layer. Must be of Polyline type.
<output dataset>	A String - the full name of the output layer.
<Station Distance>	A Double representing distance between the station lines.
<Side>	<p>Required. A String -This parameter defines on which side of the polyline the station lines will be created:</p> <ul style="list-style-type: none"> • "Both" • "Left". • "Right"
<Length From>	<p>Required. A String - the source for the length of the station lines:</p> <ul style="list-style-type: none"> • "C", "Constant" - any of the two values can be used • "Z", "ZValues" - any of the two values can be used

	<ul style="list-style-type: none"> • "M", "MValues" - any of the two values can be used
{Length}	A Double representing the length of the station lines if "Constant" length option is selected.

Running the function

ETGWPath used in the table below is the full path to ETGWRun.exe (E.G. "C:\Program Files\ETSpatial Techniques\ETGeo Wizards\ETGWRun.exe")

Language	Syntax
Python	subprocess.call([ETGWPath, "CreateStationLines", "input dataset", "output dataset", "Station Distance", "Length From", "Length"])
.NET using ETGWRun.exe	StartInfo.FileName = ETGWPath StartInfo.Arguments = "CreateStationLines" "input dataset" "output dataset" "Station Distance" "Length From" "Length"
.NET using ETGWOuX.dll	CreateStationLines(input dataset, output dataset, Station Distance, Length From, Length)
ArcPy	arcpy.CreateStationLines(input dataset, output dataset , "Station Distance", "Length From", "Length")

Notes:

- <> - required parameter
- {} - optional parameter
- See examples for [Python](#) , [.NET](#) or [ArcPy](#)
- The argument separator for StartInfo.Arguments is space. If a string might contain a space, you need to double quote it.

[\(Go to TOP\)](#)

Free functions of ET GeoWizards

ET GeoWizards is not a free program. It has however many functions that are free - can be used with the unregistered version with no limitations.

Note that the free functions are available only when executed from the User Interface. They are not available if the functionality is executed from Python, .NET or ArcToolbox

List of Free Functions

- Basic functions
 - Create New feature class
 - Create new File GDB
 - Sort Shapes
 - Move Shapes
 - Rotate Shapes
 - Scale shapes
 - Generate
 - Ungenerate
 - Explode multi-part features
 - Closest Feature Distance
 - Select and Export
 - Project Layer
- Field Functions
 - Order Fields

- Redefine Fields
- Copy Fields
- Delete Multiple Fields
- Conversion functions
 - Polygon To Polyline
 - Polygon To Point
 - Polyline To Point
 - Polyline To Polygon
 - Polyline To Multipoint
 - Point To Polyline
 - Point To Polygon
 - Point To Multipoint
 - Point To Point Z (M)
 - Multipoint To Point
 - Multipoint To Polyline
 - Shape Z (M) To Shape
 - Shape To Shape Z
- Overlay functions
 - Clip layer

- Erase layer
- Merge Layers
- Sampling Functions
 - Create Point Grid
 - Vector Grid
- Spatial Relations Functions
 - Convex Hull
- Polyline functions
 - Generalize polyline layer
 - Densify polyline layer
 - Get PolylineZ characteristics
 - Flip Polylines
- Point functions
 - Point Distance
 - Station Points