

ET GeoWizards is a set of powerful functions that will help the ArcGIS users to manipulate data with easy. It offers a lot of functionality not available as standard in ArcGIS. It also enables the ArcGIS users with ArcView (ArcGIS Basic) licenses to perform some data processing functions currently available only in ArcEditor (ArcGIS Standard) and ArcInfo (ArcGIS Advanced).

The main target of the software are the ArcView license holders, but it will be an asset for everyone using ArcEditor and even ArcInfo

The functionality of ET GeoWizards is available in two different ways

- Via the user friendly wizard type interface
- Via a set of tools for Arc Toolbox (ArcGIS 9.0 or above) which can be used in the Model Builder, Command Line or in Python scripts.

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
- See [How to Register ET GeoWizards](#) for registration information

Installation Instructions

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

Important note: If you have ET GeoWizards installed and plan to uninstall ArcGIS you MUST first uninstall ET GeoWizards

- Close ArcMap
- If you have a previous version of ET GeoWizards installed, uninstall it first.
- Make sure that you have the sub-version of ET GeoWizards appropriate for your ArcGIS version
- Unzip ETGeoWizardsXX.zip - two files will be extracted from the archive:
 - setup.exe
 - ETGeoWizardsXX_YY_Setup.msi
- Run setup.exe - a simple installation wizard will guide you through the process.
- A new program group with 2 items will be created
 - ET GeoWizards User Guide
 - Readme

If you start ArcMap the ET GeoWizards toolbar should be already loaded.

- For pre ArcGIS 10 - Go to View ==> Toolbars and select the toolbar.
- For ArcGIS 10 - Go to Customize ==>Toolbars and select the toolbar.



Note:

ET GeoWizards runs in DEMO mode until registered.

- The Demo mode has the following limitations
 - Some of the features are free - do not have any restrictions with the DEMO version. See ET GeoWizards - [free features for a list](#)
 - All the Surface functions can work with layers with up to 300 features
 - The rest of the functions have restriction of 100 features in the layer to be processed
- See [How to Register ET GeoWizards](#) for registration information

How to use ET GeoWizards

A. Via the User Interface

1. Clicking on the  button 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.
6. Follow the Wizard

Note:

Read the messages in the message box on the right side of the wizards. There will be a short description of the current function, it will report for incorrect inputs or give some additional instructions

B. In ArcToolbox

The ET GeoWizards tools can be used as any standard geoprocessing tool - from ArcToolbox, in the Model Builder or Python scripts.

Load all ET GeoWizards tools:

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 ETGeoWizards.tbx file
3. Click Open.

Load a tool into your own toolbox or toolset:

1. Right-click the toolbox or toolset where you want to add system tools, point to Add, and click Tool.
2. In the dialog find and expand ET GeoWizards toolbox and toolsets in it. Check the tools you would like to add to your toolbox or toolset. If you check the toolbox, all tools within the toolbox will be added. If you check a toolset within the toolbox, all tools within the toolset will be added.
3. Click OK.

Notes:

- 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.
- The usage of the ToolBox tools is described at the bottom of the topic for each ET GeoWizards function. Many of the help topics have also Python script examples.

C. In .NET customizations

Copyright © Ianko Tchoukanski

How to use ET GeoWizards functionality in .NET

Most of the functions of ET GeoWizards (starting from version 11.2) can be used in custom applications (stand alone, ArcGIS Add - Ins, custom controls). The syntax of each ET GeoWizards function is described in the main topic of the function ==> .NET implementation. See also the [utility functions for the .NET implementation](#).

Quick start - VB.NET example for stand alone application

Prerequisites

- ArcGIS - installed and licensed
- ArcObjects SDK for NET Framework - installed
- Microsoft Visual Studio
- ET GeoWizards 11.2 and above - installed and registered

1. Start Visual Studio
2. Go to File ==> New ==> Project
 - In the dialog go to Installed Templates ==> Visual Basic ==> ArcGIS ==> Extending ArcObjects and select Windows Application (Desktop)
 - Give a name to your project and click OK
 - In the ArcGIS Project Wizards that will open select your product (Basic, Standard or Advanced) and click Finish
3. Go to Project ==> Properties ==> References:
 - Add reference ==> .NET ==> find ESRI.ArcGIS.Geodatabase and select it ==> Click OK
 - Add reference ==> Browse ==> navigate to the installation folder of ET GeoWizards and select ETGeoWizards112.dll. Make sure that "Copy Local" is set to true in the reference properties.
4. Save the Assembly and Build it.
5. Create a button on your form and name it Build Thiessen
6. Double click on the button to start editing the code
7. Paste the code below

```
Imports ETGeoWizards112
Imports ESRI.ArcGIS.Geodatabase
Public Class Form1
Private Sub PolylineToPoints_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PolylineToPoints.Click
Try
Dim et As New ETGWCORE
'set the full name of the input dataset.
Dim sInputName As String =c:\test\polylines.shp"
'set the name for the output dataset. It should not exist
Dim sOutName As String =c:\test\vertices.shp"
'get the feature class from the input name.
Dim pInFC As IFeatureClass = et.FeatureClassFromPath(sInputName)
'run the Polyline To Points function. See the topic in the user guide for description of the parameters and options available
Dim pOutFC As IFeatureClass = et.PolylineToPoints(pInFC, sOutName,Vertex")
If Not pOutFC Is Nothing Then
MsgBox("Vertices created")
Else
MsgBox("Error! See log file for details")
End If
Catch ex As Exception
MsgBox(ex.Message)
End Try
End Sub
End Class
```

Note: Some of the functions will need two additional files located in the EXE folder. Find the following files in the installation folder of ET GeoWizards and copy them in the folder of your EXE

- ETApp.exe
- ETMod.dll

How to register ET GeoWizards

A. Single use (fixed) license

The registration process involves three steps:

1. Visit ET GeoWizards page on ShareIt.com and purchase the software. You will receive a reference number for your order.
2. On the ET GeoWizards Toolbar or ET GeoWizards Main Dialog go to Help ==> Request License Key button. Fill the small form - all the fields are required.
 - User Name
 - Company
 - ShareIt 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 ==> About Tab). 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.

ET GeoWizards and Projections

Some short definitions(compiled from ArcGIS desktop help)

- 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 (for feature classes the projection information is stored in the feature classname.prj file)

- 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 (in the case of feature classes - "feature classname.prj") is missing

Projections of the dataframe (View)

- No projection - the data is displayed as is
- Assigned projection - the data is reprojected on the fly and displayed in the data frame's projection.

ET GeoWizards works with data in any projection and view frame in any projection

Notes:

- All the wizards preserve the Spatial Reference of the input data source. 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 and the data frame does not have projection (or has the same projection as the data), because the tolerance is compared directly with the data.
- The Wizards will not work if the data frame contains one or more layers that have projection with Geographic Coordinate System different from the Geographic Coordinate System of the data frame's projection. ArcMap gives a warning in such a case, but allows it.
- Although possible, it is not recommended to manipulate Unprojected data (data in a Geographic Coordinate System) for reasons mentioned above
- If the projection information is missing, the data can be manipulated only in a data frame with no projection assigned.

ET GeoWizards and Geodatabases

General

ArcGIS supports three types of Geodatabases:

- Personal geodatabase
- SDE
- File geodatabase (ArcGIS 9.2 and above)

ET GeoWizards writes its outputs as stand alone feature classes in Personal Geodatabases (PGDB) and File Geodatabases (FGDB).

Feature Datasets

The feature classes of a feature dataset might participate in Topologies, Geometric networks, etc. The results of 99% of the functions of ET GeoWizards are written into a new dataset (feature class or shapefile). In many cases the result is not final - it goes through several procedures before the final dataset is derived. To avoid placing feature classes into feature datasets that have been structured to do something, we on purpose have disabled ET GeoWizards to output in feature datasets. It is very easy to copy the result to a feature dataset after the user is sure that the desired result is obtained. One can always make a model that will use the ET GeoWizards tools and just add at the end the standard "Feature Class To Feature Class" tool to direct the output to a feature dataset.

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. These temporary datasets are not written in the users geodatabase, but in a special geodatabases. ET GeoWizards creates two temporary geodatabases - a PGDB and a FGDB. Depending on the type of the selected output the intermediate feature classes are stored in one of these temporary databases. The temporary databases are stored in a location specified by the user. The location (temp folder) can be set using the ET GeoWizards Main Dialog ==> About Tab ==> Settings). If the temp folder has not be set Et GeoWizards sets it to "c:\temp\ET_Temp" The names of the temp geodatabases are:

- "et_tempPGDB.mdb" - for storing temporary PGDB feature classes.
- "et_tempFGDB" - for storing temporary FGDB feature classes.

Maintenance of the temp geodatabases

All functions are designed to maintain the temporary geodatabases by

- Removing the intermediate feature classes after completion
- Compacting the databases


In some cases however some of the functions cannot delete the intermediate datasets. This might cause the size of the temporary geodatabases to grow after long use of the software. Large size of the temporary geodatabases might cause decreased performance of some of the functions of ET GeoWizards. The easiest way to avoid such problems is simply to delete the temporary geodatabases on regular basis.

ET GeoWizards Toolbar

The toolbar







contains:

- The button that opens the main dialog of the software - 
- The Help Menu which has the following items depending on the version
 - Fixed Version
 - User Guide - opens the documentation of ET GeoWizards
 - Settings - opens dialog which display the current temp folder and allows the user to change the temp folder.
 - View Log File - opens the log file where are recorded all the activities of the functions of ET GeoWizards. It is recommended that this file is cleaned from time to time.
 - About - opens a dialog that states the version and the build date of ET GeoWizards as well as the registration status of the software
 - Request License Key - Use this after purchasing a license to send registration information to register@ian-ko.com
 - Register - Use this to load the license key which you will receive after sending the registration information
 - Purchase Online - will open the default WEB Browser with the order page of ET GeoWizards.
 - Concurrent Version
 - User Guide - opens the documentation of ET GeoWizards
 - Settings - opens dialog which display the current temp folder and allows the user to change the temp folder.
 - View Log File - opens the log file where are recorded all the activities of the functions of ET GeoWizards. It is recommended that this file is cleaned from time to time.
 - About - opens a dialog that states the version and the build date of ET GeoWizards as well as the registration status of the software
 - Connect To License Server - opens the dialog that allows the user to connect to the license server.
 - Release License - releases the license checked out to make it available for other users.

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.

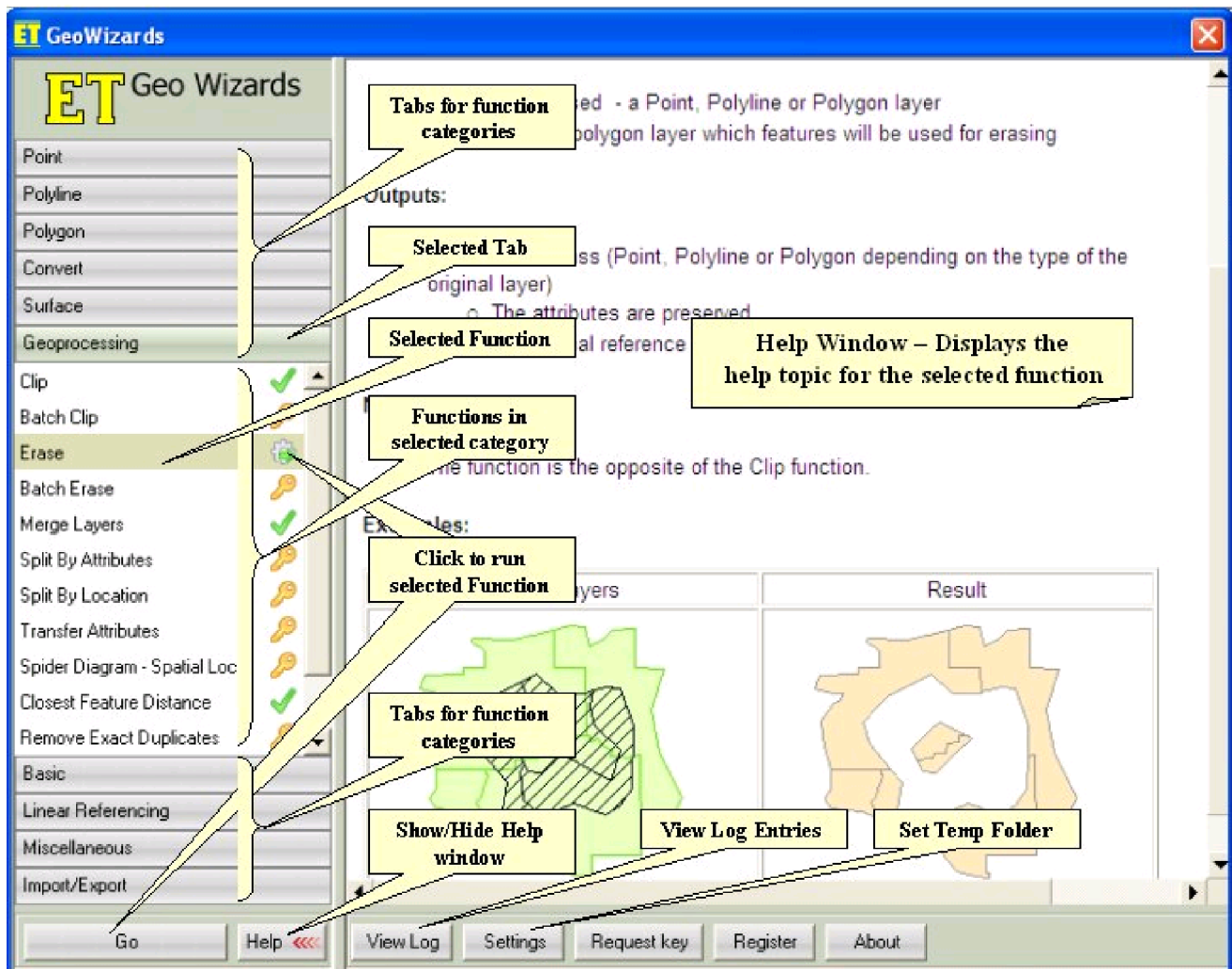
On the registered software only  and  icons should appear.

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.



Near Feature

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

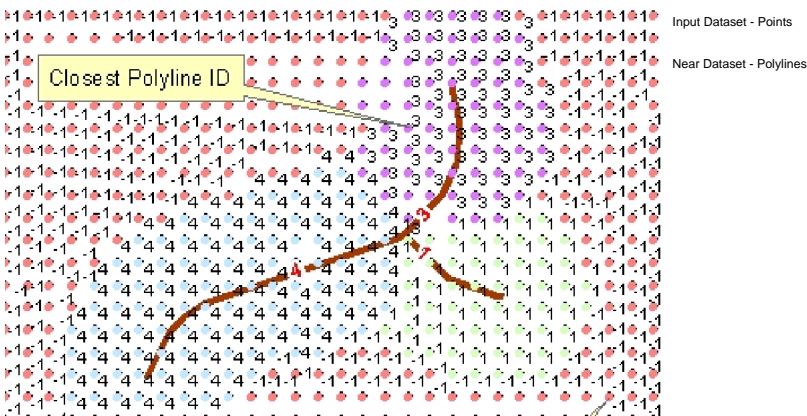
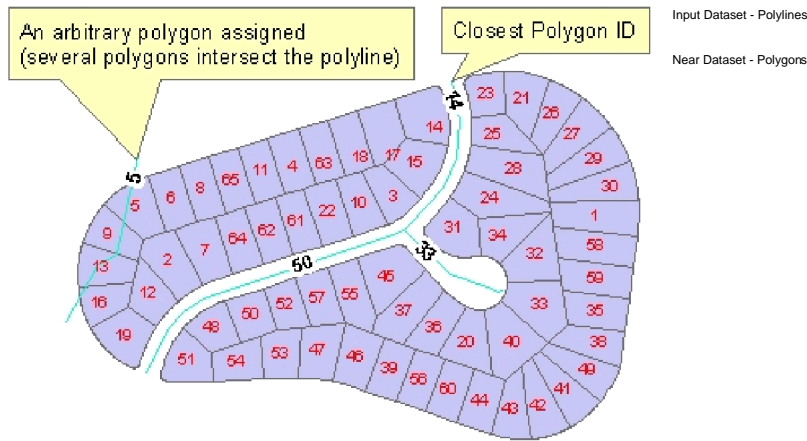
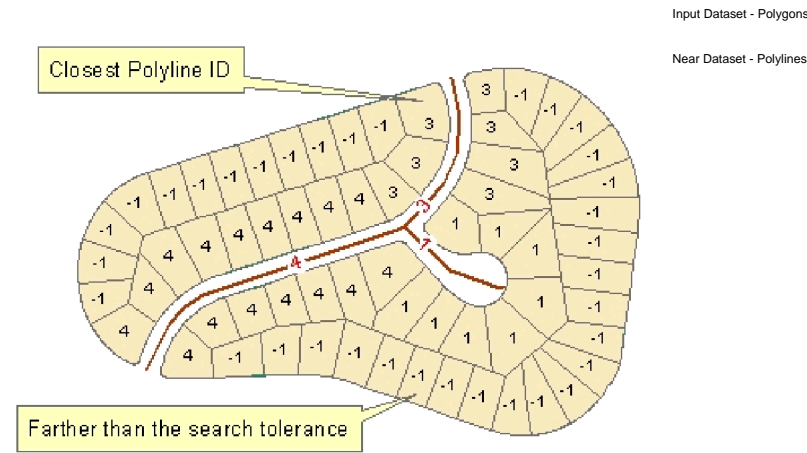
Outputs:

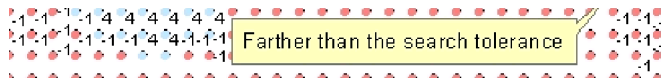
- New feature class. The attribute table of the resulting feature class will have three new fields
 - [ET_Dist] - the distance from the input feature to the closest feature from the near layer
 - [ET_Closest] - the ID of the closest feature from the near layer

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.
- The spatial references of the Input and Near dataset must have the same geographic coordinate system.

Examples:





ToolBox implementation

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

Command line syntax

ET_GPNearFeature<input_dataset> <Near_dataset> <out_feature_class> <search_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Point, Multipoint, Polyline or Polygon feature class or feature layer.
<Near_dataset>	A Point, Multipoint, Polyline or Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<search_tolerance>	A Double representing the Search tolerance (in the units of the spatial reference of the input dataset) to be used

Scripting syntax

ET_GPNearFeature(input_dataset, Near_dataset, out_feature_class, search_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

NearFeature(plnFC As IFeatureClass, pNearFC As IFeatureClass, sOutFName As String, dSearchTol As Double) As IFeatureClass

Allocation

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

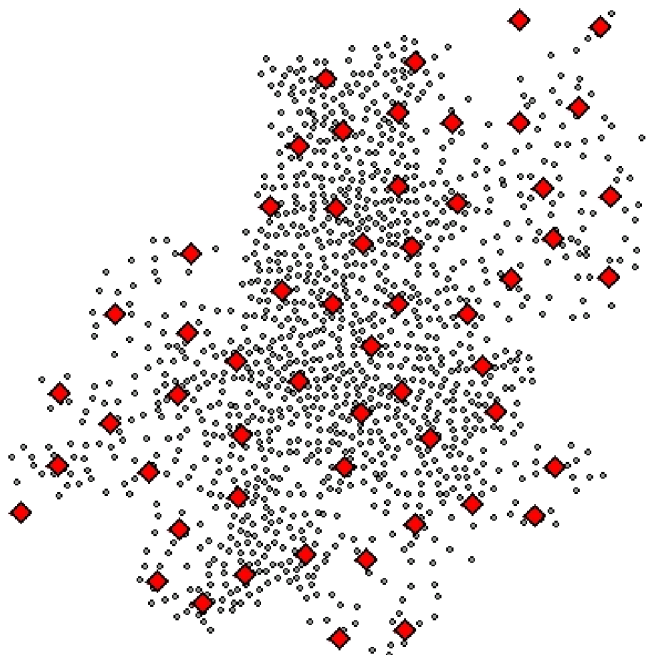
- Point feature layer representing the Facilities (Centers).
- Facility name field (optional) - the values in this field are used to identify the facilities. If the field is not specified the FID will be used as a name
- Facility type field (optional) - the values of this field indicate whether a specific 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:

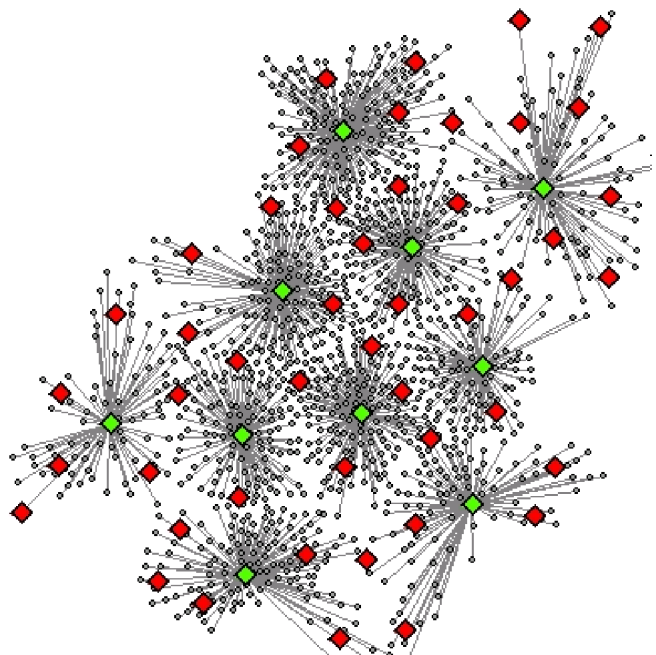
- New Point feature class containing only the selected facilities. The attribute table of the resulting feature class will have the following fields
 - FacilityID - The original FID of the selected facility
 - Facility - The value in the user specified Name field of the selected facility
 - Type - the 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 lines linking 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

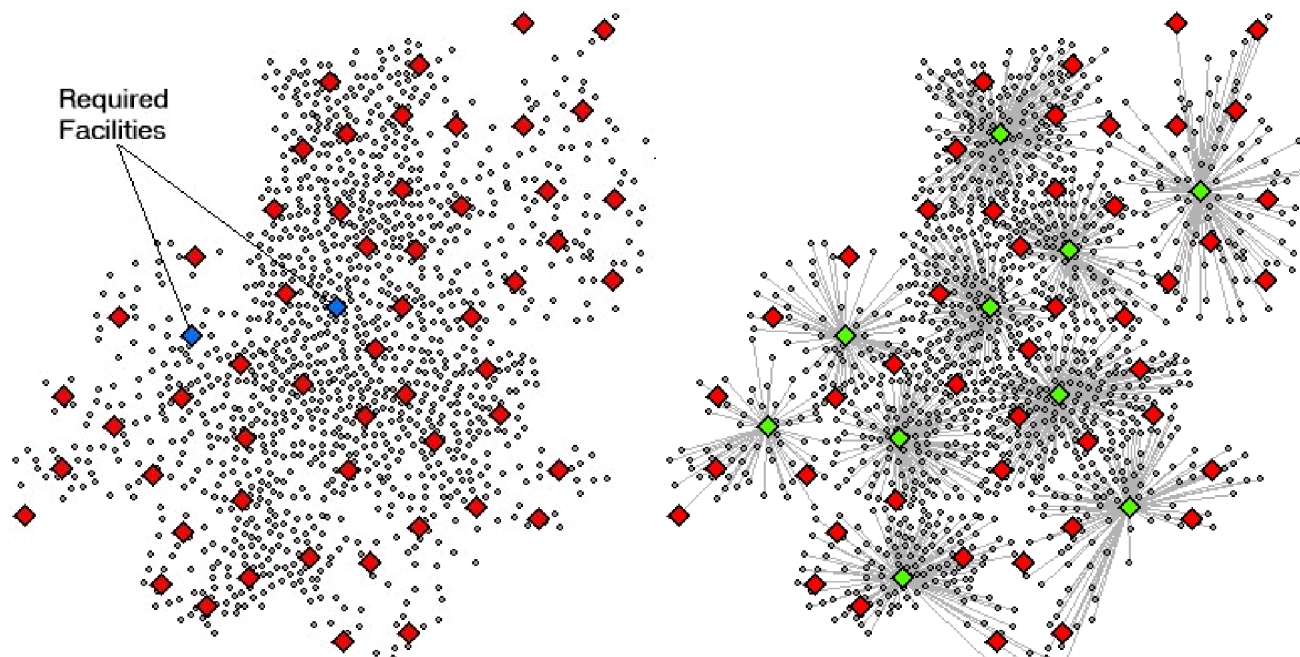


Result (Selected facilities in green)



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).

ToolBox implementation

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

Command line syntax

ET_GPAlocate <facilities_dataset> {facility_name_field} {facility_type_field} <customers_dataset> {customer_name_field} <out_links_feature_class> <out_facilities_feature_class> <number_facilities> {Cutoff_distance}

Parameters

Expression	Explanation
<centers_dataset>	A Point feature class or feature layer - the candidate facilities.
{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 facilities.
<customers_dataset>	A Point feature class or feature layer - the Customers (demand points).
{customer_name_field}	A String representing a field name - the values in this field are used to identify the customers.
<out_links_feature_class>	A String - the full name of the output link feature class (A feature class with the same full name should not exist)
<out_facilities_feature_class>	A String - the full name of the output selected facilities feature class (A feature class with the same full name should not exist)
<number_facilities>	An integer - the number of facilities to be selected
{Cutoff_distance}	A number - the maximum distance between a Facility and a Customer to be used.

Scripting syntax

EET_GPAlocate (facilities_dataset, facility_name_field, facility_type_field, customers_dataset, customer_name_field, out_links_feature_class, out_facilities_feature_class, number_facilities, Cutoff_distance

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

```
Allocate(pFacFC As IFeatureClass, pCustFC As IFeatureClass, sOutLinkFN As String, sOutFacFN As String, iNumFac As Integer, Optional sFacName As String = "",  
Optional sFacType As String = "", Optional sCustName As String = "", Optional dCutOff As Double = 1000000000) As IFeatureClass
```

Copyright © Ianko Tchoukanski

Create Concave Hull

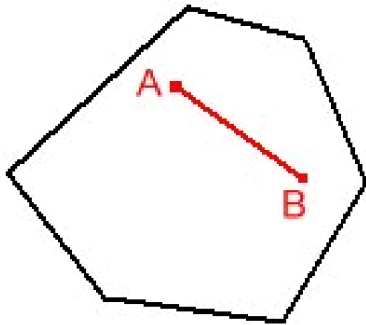
[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

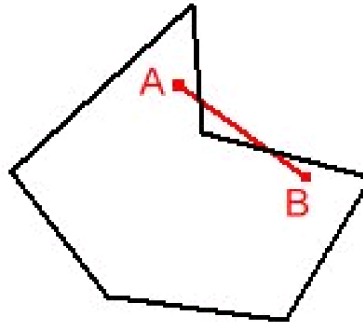
The Concave Hull function creates a polygon that represents the area occupied by a set of data points.

- The resulting polygon might be concave or convex

Convex

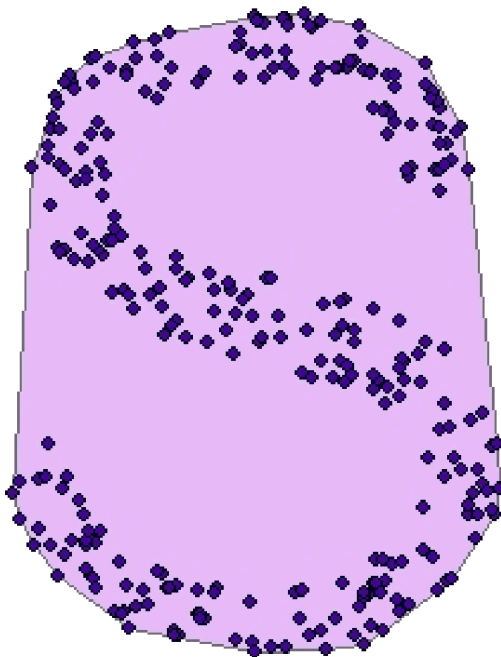


Non Convex (Concave)

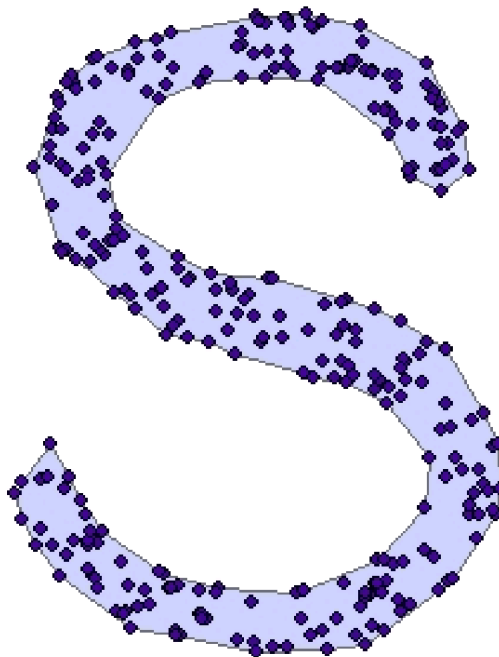


- A Concave hull describes better the shape of the point cloud than the convex hull

Convex Hull



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

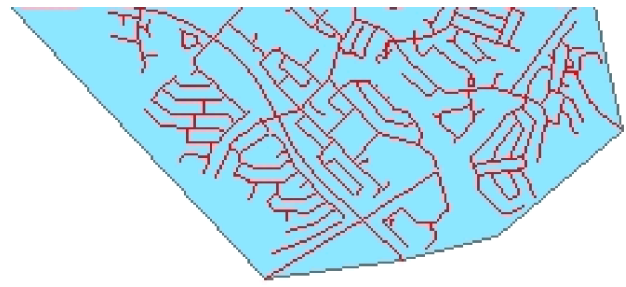


Convex Hull

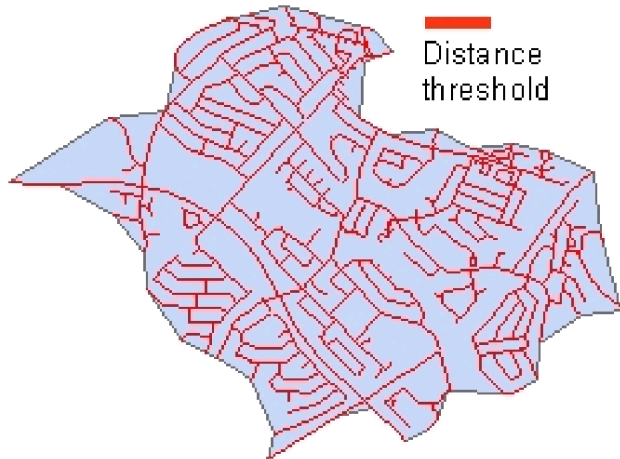




Concave Hull 1



Concave Hull 2



Inputs:

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

Outputs:

- New polygon feature class.

ToolBox implementation

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

Command line syntax

ET_GPCreateConcaveHull <input_dataset> <out_feature class> <distance_threshold>

Parameters

Expression	Explanation
<input_dataset>	A Point, Multipoint, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<distance_threshold>	A Double representing the threshold for creating a concave hull - in the units of the spatial reference of the input dataset

Scripting syntax

ET_GPConvexHull (input_dataset,out_feature class, distance_threshold)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CreateConcaveHull(pInFC As IFeatureClass, sOutFName As String, dDistanceTolerance As Double) As
IFeatureClass

Copyright © Ianko Tchoukanski

Build Convex Hull

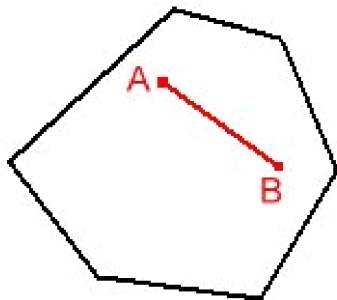
[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

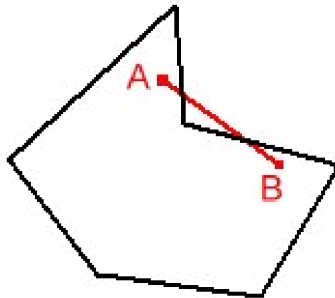
Builds the Convex Hull of the features of a layer

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

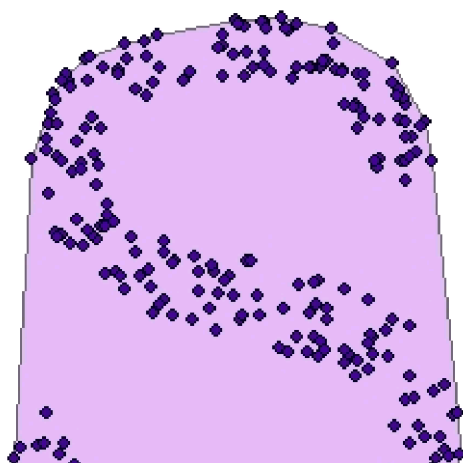


Convex Hull

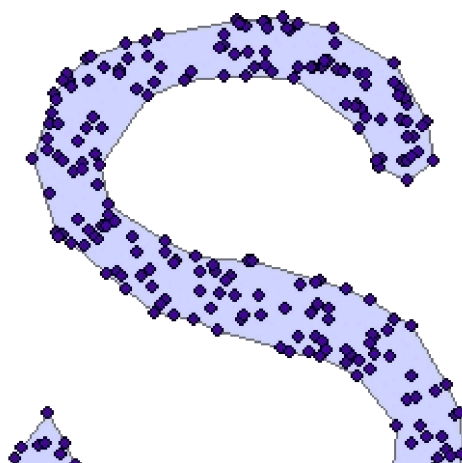


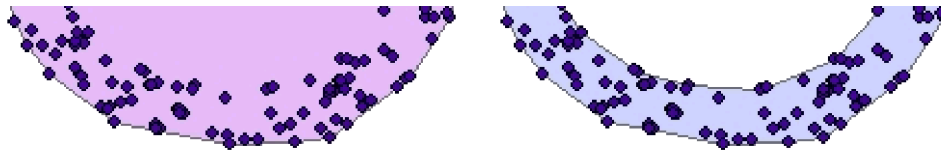
While in general the Convex Hull is good to describe the shape of the input data points, in many cases a polygon that describes better the region occupied by the point cloud is needed. See the Create Concave Hull function

Convex Hull



Concave Hull





Inputs:

- A feature layer (Point, Polyline, Polygon)

Outputs:

- New polygon feature class.

ToolBox implementation

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

Command line syntax

ET_GPConvexHull <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPConvexHull (input_dataset,out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ConvexHull(pInFC As IFeatureClass, sOutFName As String, dMaxGap As Double, Optional bAvoidLoops As Boolean = False)
As IFeatureClass

Create Cluster Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

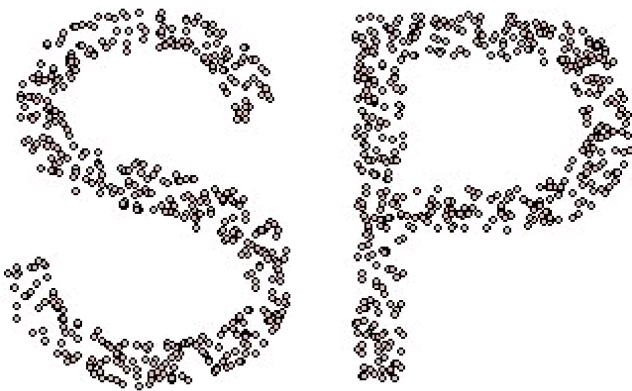
- A feature layer (Point, Polyline, Polygon)
- Cluster Tolerance - in the units of the spatial reference of the input dataset
- Holes/No Holes option

Outputs:

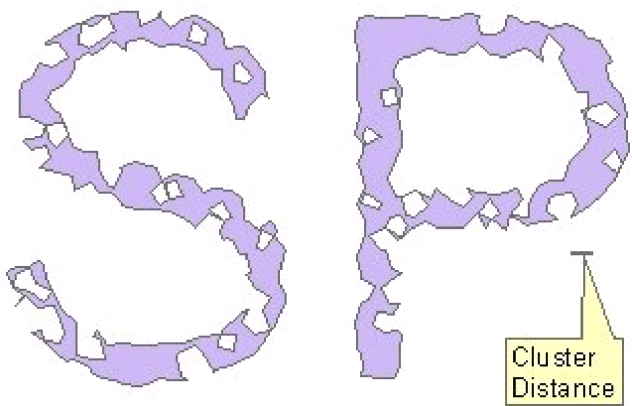
- New polygon feature class.

Examples:

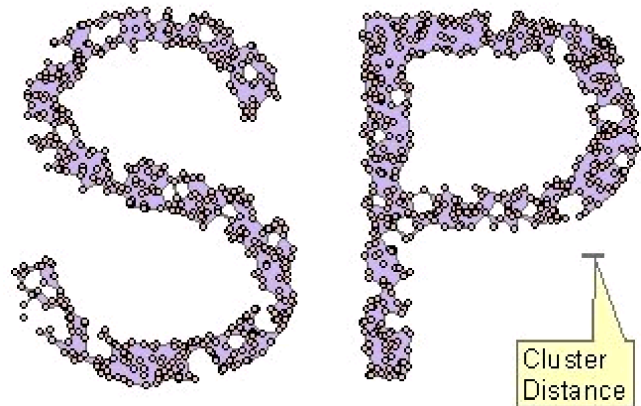
Source Points



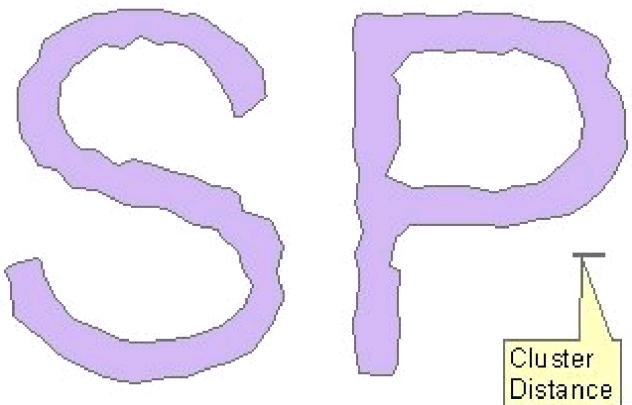
Cluster Polygons 1



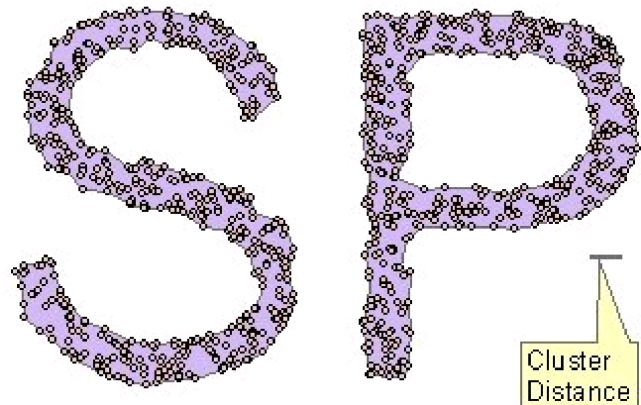
Cluster Polygons 1 overlaid with the source points



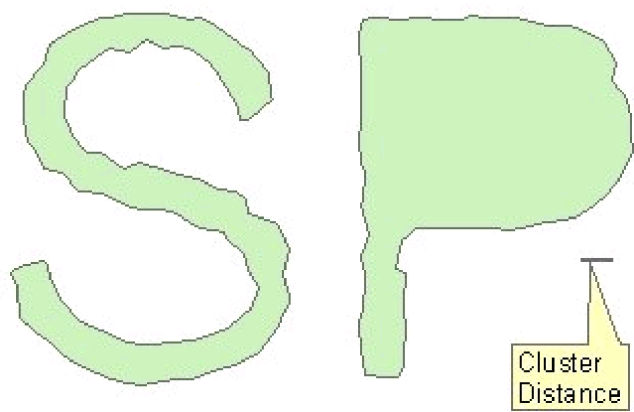
Cluster Polygons 2



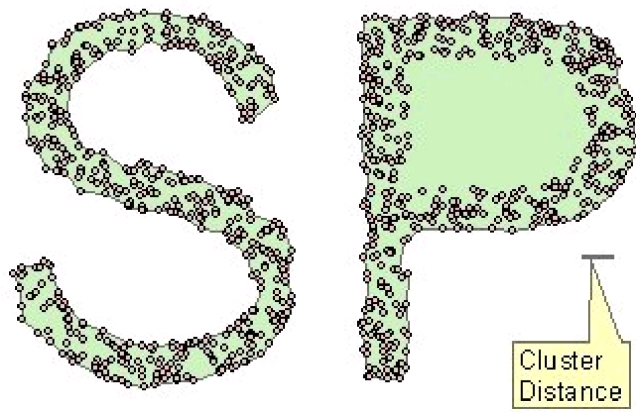
Cluster Polygons 2 overlaid with the source points



Cluster Polygons 3 (No Holes option selected)



Cluster Polygons 3 overlaid with the source points



ToolBox implementation

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

Command line syntax

ET_GPCreateClusterPolygons<input_dataset> <out_feature class> <cluster_distance><remove_holes_from_polygons>

Parameters

Expression	Explanation
<input_dataset>	A Point, Multipoint, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<cluster_distance>	A Double representing the maximum distance between the points within a cluster - in the units of the spatial reference of the input dataset
<remove_holes_from_polygons>	A Boolean - If true, the resulting polygons will not contain holes.

Scripting syntax

ET_GPCreateClusterPolygons(input_dataset,out_feature class, cluster_distance, remove_holes_from_polygons)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\clusters"
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx");
arcpy.gp.ET_GPCreateClusterPolygons(input_dataset, result, 10.00, true)
```

.NET implementation

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

CreateClusterPolygons(pInFC As IFeatureClass, sOutFName As String, dClusterTolerance As Double, Optional bRemoveHoles As Boolean = False) As IFeatureClass

Build Thiessen Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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, Polyline, Polygon)

Outputs:

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

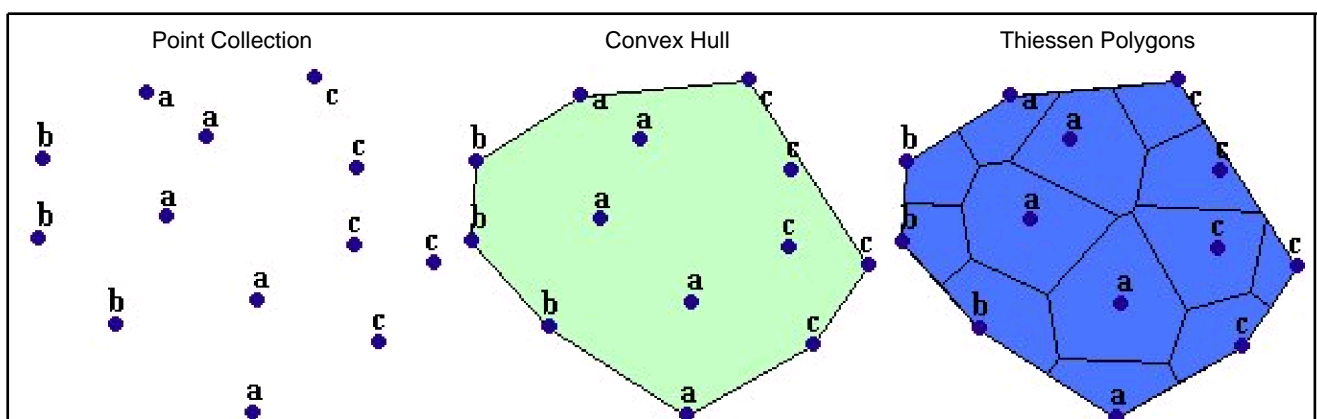
Notes :

- The process goes through several steps
 - Collects the points from a point layer (vertices if the source is a polyline or polygon layer)
 - Clean duplicate points
 - Generates Convex Hull
 - Creates a TIN structure
 - Generates perpendicular bisectors for each tin edge.
 - Builds the Thiessen polygons
 - Clips the Thiessen polygons feature class with the convex hull.
- To achieve best results when creating Thiessen Polygons from a polyline layer use Generalize Polylines Wizard or Densify Polyline Wizard (before running the Thiessen Polygons procedure) in order to remove unnecessary points or add points to the long straight segments
- 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.
- If the source is a polyline or polygon layer, only the attributes of the first feature found inside each Thiessen polygon will be transferred.
- The function should work with no problems on datasets with up to 2 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:



ToolBox implementation

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

Command line syntax

ET_GPBuildThiessen <input_dataset> <out_feature class> {clip_buffered} {buffer_distance} {attach_attributes}

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{clip_buffered}	A Boolean indicating whether the output will be restricted to the extent rectangle of the input features (False) or will be extended with the specified buffer distance (True)
{buffer_distance}	A Double setting the the distance with which the extent rectangle of all the input features will be buffered (in the units of the input dataset) to be used if the {clip_buffered} is True. If {clip_buffered} is False this parameter is ignored
{attach_attributes}	A Boolean indicating whether the attributes of the original features will be transferred to the resulting thiessen polygons

Scripting syntax

ET_GPBuildThiessen (input_dataset, out_feature class, clip_buffered, buffer_distance, attach_attributes)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

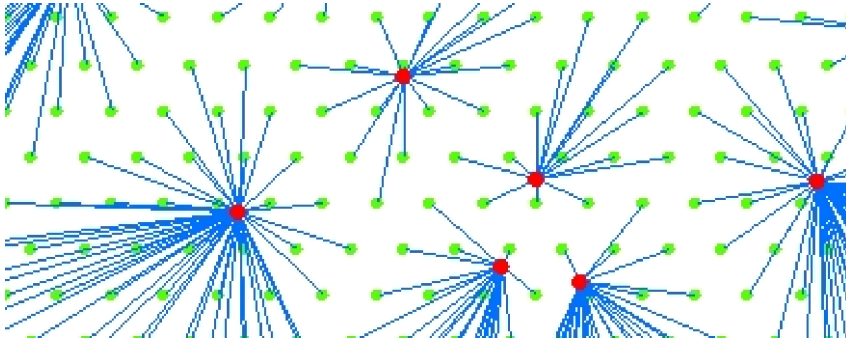
BuildThiessen(pInFC As IFeatureClass, sOutFName As String, Optional dBuffer As Double = 0, Optional bAttach As Boolean = True) As IFeatureClass

Spider Diagram

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 a further than this distance from any Center will not be assigned to a Center
- Output Spatial Reference

Outputs:

- New Point 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

Notes:

- The default output spatial reference is the one of the Centers 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 feature classes

ToolBox implementation

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

Command line syntax

```
ET_GPSSpiderDiagram <centers_dataset> <Destination_dataset> <out_feature class> <Cutoff_distance>
{output_spatial_reference}
```

Parameters

Expression	Explanation
<centers_dataset>	A Point feature class or feature layer
<Destination_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Cutoff_distance>	A Double representing the Search tolerance (in the units of the {output_spatial_reference}) to be used

{output_spatial_reference} The spatial reference in which the calculations will be performed. If not specified the spatial reference of the input dataset will be used.

NOTE: The spatial references of the <centers_dataset>, <Destination_dataset> and {output_spatial_reference} must have the same Geographic Coordinate System

Scripting syntax

ET_GPSpiderDiagram (centers_dataset, Destination_dataset, out_feature class, Cutoff_distance, output_spatial_reference)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SpiderDiagram(pCentersFC As IFeatureClass, pDestFC As IFeatureClass, sOutFName As String, dCutOff As Double, pOutSRef As ISpatialReference) As IFeatureClass

Spider Diagram Attribute Link

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 - connect only closest destination. If not used all destinations that have the same values as a center will be connected to the center.
- Optional - Depending on the type of the destination dataset the following options are available
 - Point - not used
 - Polyline
 - Connect the Center to closest point on the polyline (Default)
 - Connect the Center to the middle point of the polyline
 - Polygon
 - Connect the Center to the closest point on the polygon boundary
 - Connect the Center to the centroid of the polygon.
- Optional - Cutoff distance - the maximum distance between a Center and a Destination to be used. Destinations that a 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

ToolBox implementation

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

Command line syntax

```
ET_GPSSpiderLink <centers_dataset> <centers_link_field> <Destination_dataset> <destination_link_field> <out_feature class> {closest_only}{connect_center} {Cutoff_distance}
```

Parameters

Expression	Explanation
<centers_dataset>	A Point feature class or feature layer
<centers_link_field>	A String representing the link field in the centers feature class.
<Destination_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<destination_link_field>	A String representing the link field in the destinations feature class.
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{closest_only}	A Boolean - if True, only the closest destination feature to the center will be connected

{connect_center}

A Boolean - the value depends on the type of the destination feature class:

- Point - not used
- Polyline
 - False = Connect the Center to closest point on the polyline (Default)
 - True = Connect the Center to the middle point of the polyline
- Polygon
 - False = Connect the Center to the closest point on the polygon boundary(Default)
 - True = Connect the Center to the centroid of the polygon.

{Cutoff_distance}

A Double representing the Search tolerance (in the units of the {centers_dataset} to be used

Scripting syntax

ET_GPSSpiderLink (centers_dataset, centers_link_field Destination_dataset, destination_link_field, out_feature class, closest_only, connect_center, Cutoff_distance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SpiderDiagramAttributeLink(pCentersFC As IFeatureClass, sCentersLink As String, pDestFC As IFeatureClass, sDestLink As String, sOutFName As String, Optional bClosestOnly As Boolean = False, Optional bMiddle As Boolean = False, Optional dCutOff As Double = 1000000000) As IFeatureClass

Find Closest Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 2 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

ToolBox implementation

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

Command line syntax

ET_GPFindClosestPoint<input_dataset> <out_feature class> <cut_off>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<cut_off>	A Double representing the maximum distance between two points to be considered neighbors - in the units of the spatial reference of the input dataset

Scripting syntax

ET_GPFindClosestPoint(input_dataset,out_feature class, cut_off)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FindClosestPoint(pInFC As IFeatureClass, sOutFName As String, dCutOff As Double) As IFeatureClass

Connect To Closest Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates an output polyline feature class with single segmented polylines that connect each point of the input Point feature class to its 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 feature class. The attribute table of the resulting feature class 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.

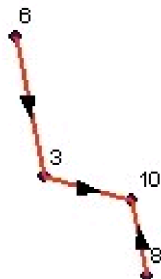
Examples:



Result Dataset - Add Duplicate Links option NOT selected

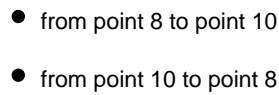
- The closest point to point 8 is point 10.
- The closest point to point 10 is point 8

Only one link is added - from point 8 to point 10





- Two coincident links are added



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

ET_GPConnectToClosestPoint<input_dataset> <out_feature class> <cut_off>{add_duplicate_links}

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<cut_off>	A Double representing the maximum distance between two points to be considered neighbors - in the units of the spatial reference of the input dataset
{add_duplicate_links}	A Boolean indicating whether duplicate links to be added to the output or not.

ET_GPCConnectToClosestPoint(input_dataset,out_feature class, cut_off,add_duplicate_links)

{ } - optional parameter

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

```
ConnectToClosestPoint(plnFC As IFeatureClass, sOutFName As String, dCutOff As Double, Optional bAddDuplicates As Boolean = False) As IFeatureClass
```

Connect Unstructured Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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)

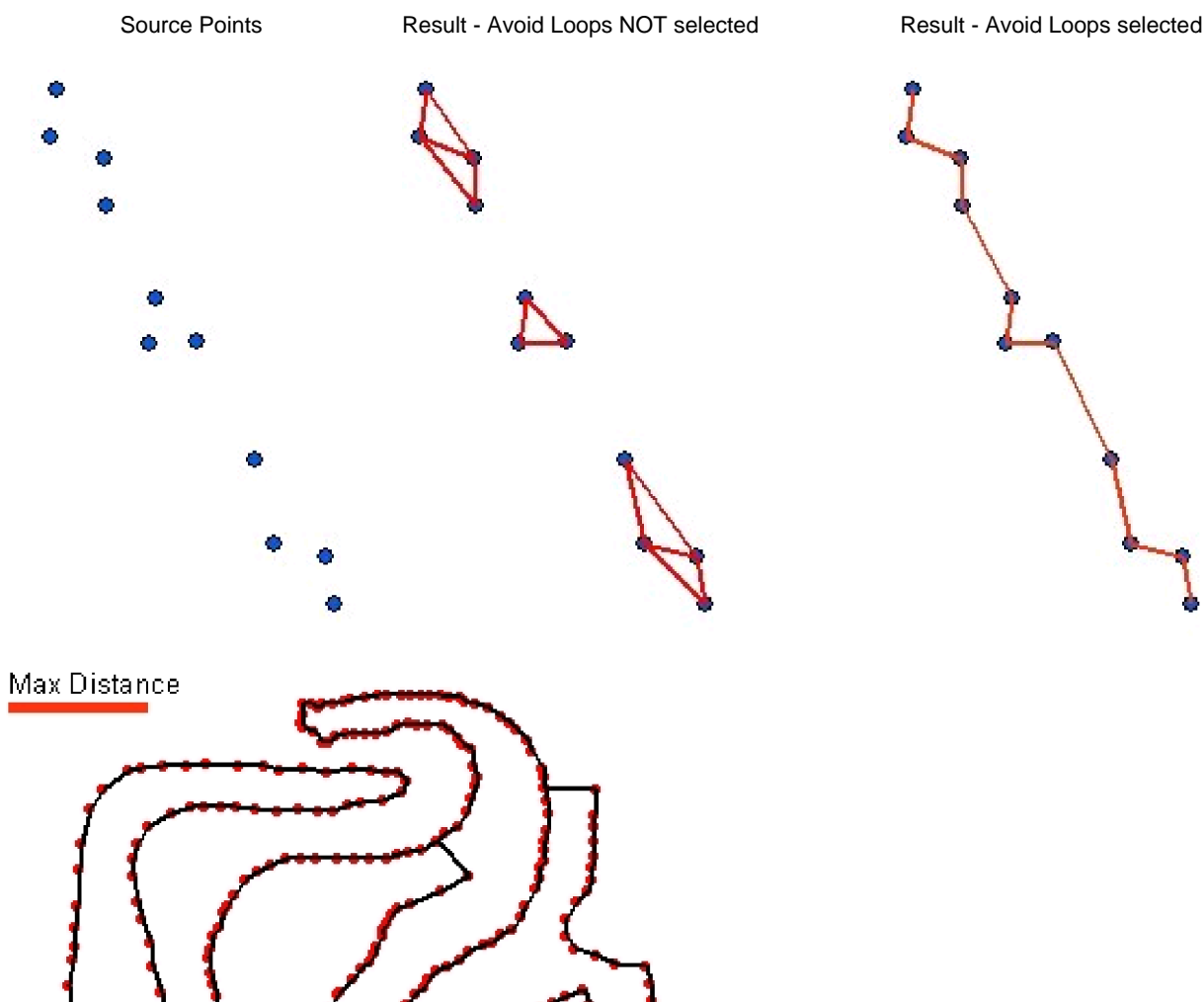
Outputs:

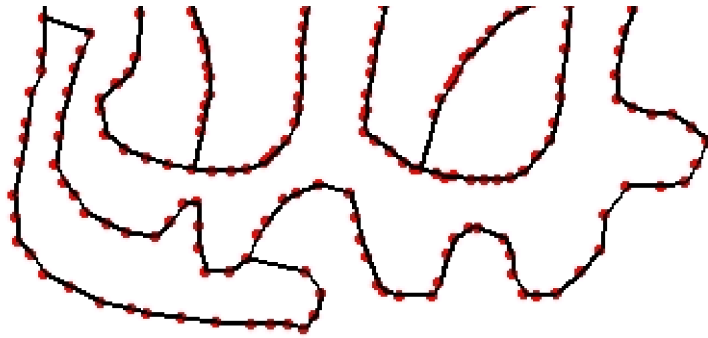
- A new Polyline feature class.

Notes:

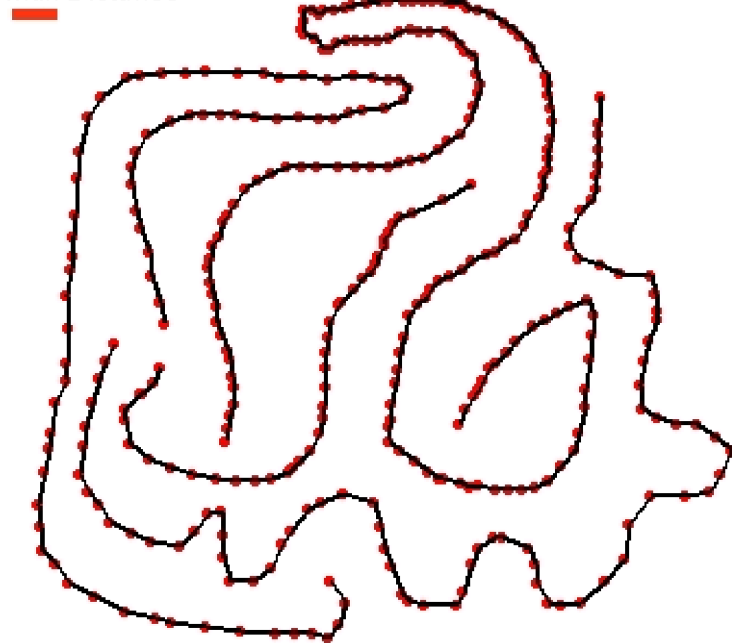
- If the distance from a point to the closest point is larger than the Cutoff distance then no link will be created between the two points
- If there are coincident points in the input dataset, the duplicates will be ignored.
- If the Avoid Loops option is not selected each point will be connected to 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:





Max Distance



ToolBox implementation

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

Command line syntax

ET_GPConnectUnstructuredPoints<input_dataset> <out_feature class> <cut_off>{avoid_loops}

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<cut_off>	A Double representing the maximum distance between two points to be considered neighbors - in the units of the spatial reference of the input dataset
{avoid_loops}	A Boolean indicating whether the function will try to avoid loops when connecting the points (see example above).

Scripting syntax

ET_GPConnectUnstructuredPoints(input_dataset,out_feature class, cut_off,avoid_loops)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ConnectUnstructuredPoints(pInFC As IFeatureClass, sOutFName As String, dMaxGap As Double, Optional bAvoidLoops As Boolean = False) As IFeatureClass

Copyright © Ianko Tchoukanski

Ungenerate

[Go to ToolBox Implementation](#)

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 forma - described below

Inputs:

- A feature layer
 - Point, PointZ, PointM
 - Polyline, PolylineZ, PolylineM
 - Polygon, PolygonZ, PolygonM
- Spatial reference. The coordinates of the features can be exported in the original projection of the dataset or in the spatial reference assigned to the Data Frame

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 rectangles only" is selected (available for Polyline and Polygon layers), the resulting text file will contain the coordinates of the extents for each shape.
- 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	id,x,y	1,34.5,-14.3	ID,X,Y,FIELD,FIELD	ID,X,Y,Town,Population
PointZ	id,x,y	2,12.8,-19.6	id,x,y,value,value	1,34.5,-14.3,London,44
PointM	id,x,y	3,13.4,-25.6	id,x,y,value,value	2,12.8,-19.6,Paris,34
	END	END	id,x,y,value,value	3,13.4,-25.6,Madrid,56
			END	END
Polyline	id	1	ID,FIELD,FIELD	ID,Street,Streettype
PolylineZ	x,y	34.5,-14.3	id,value,value	1,Church,Street
PolylineM	x,y	12.8,-19.6	x,y	34.5,-14.3
	END	END	x,y	12.8,-19.6
	id	2	END	END
	x,y	13.4,-25.6	id,value,value	2,Second,Avenue
	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
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
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	

ToolBox implementation

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

Command line syntax

ET_GPUngenerate <input_dataset> <out_file> <delimiter> {export_attributes}

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer
<out_file>	A String - the full name of the output text file
<delimiter>	A String indicating what separator to be used. Valid strings: <ul style="list-style-type: none">● "Comma"● "Space"● "Tab"
{export_attributes}	A Boolean indicating whether to export the attributes.

Scripting syntax

ET_GPUngenerate (input_dataset out_file delimiter export_attributes)

See the explanations above:
<> - required parameter
{ } - optional parameter

Generate Wizard

Creates a feature class from ArcInfo generate format text file. The function supports an extended text file format in order to be able to import attributes.

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,(z) id,x,y,(z) id,x,y,(z) 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,(z),value,value id,x,y,(z),value,value id,x,y,(z),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,(z) x,y,(z) END id x,y,(z) x,y,(z) x,y,(z) x,y,(z) 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,(z) x,y,(z) END id,value,value x,y,(z) x,y,(z) x,y,(z) END END	ID,Street,Streetytype 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 PolygonZ PolygonM	id,xLabel,yLabel x,y,(z) x,y,(z) x,y,(z) END id x,y,(z) x,y,(z) x,y,(z) 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,(z) x,y,(z) x,y,(z) END id,value,value x,y,(z) x,y,(z) x,y,(z) 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
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	

Import from Google Earth

[Go to ToolBox Implementation](#)

Converts the feature data contained in a KML or KMZ file to feature classes.

Inputs:

- A Google Earth KML or KMZ file
- Output workspace

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.
- Importing of KML files is slow in general, be patient.
- See [Google Earth general](#) for important additional information.

ToolBox implementation

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

Command line syntax

ET_GPImportFromGoogle <in_file> <out_workspace>

Parameters

Expression	Explanation
<in_file>	A String - the full name of the input Google Earth file.
<out_workspace>	A String - the full name of the output workspace (folder, personal or file geodatabase)

Scripting syntax

ET_GPImportFromGoogle (in_file out_workspace)

See the explanations above:

<> - required parameter

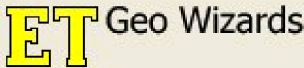
{ } - optional parameter

Copyright © Ianko Tchoukanski

Map To Google Earth

Exports all visible feature layers to Google Earth KML or KMZ file.

Export to Google Wizard



Export To Google

Layer	Name	Transp	ZFrom	ZValue	ZUnits	Altitude	Extrude	Label	Symbol	Scale
POI	Name	0	Z	Z	Meters	Absolute	<input type="checkbox"/>	<input checked="" type="checkbox"/>	P_Circle	0.8
Roads	Name	0	None	None	Meters	None	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Info	0.8
Stations	Name	0	None	None	Meters	None	<input type="checkbox"/>	<input checked="" type="checkbox"/>	P_Circle	0.8
Pipelines	Name	0	None	None	Meters	None	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Info	0.8
Cities	CITYNAME	0	None	None	Meters	None	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Info	0.8

Set parameters for the layers to be exported. Position the cursor over a column to see a description for the parameter that can be assigned in the column.

Help

Cancel

< Back

Next >

Finish

Inputs:

- All visible layers in ArcMap
- Map name
- Map description
- Output Google Earth file
- Export option:
 - All - all features of the feature classes of the visible layers will be exported.
 - Definition - the Query Definitions will be honored - the features excluded by the query definitions will not be exported.
 - Selected - only selected features will be exported
 - Visible - only features that intersect the current visible extent will be exported
- Coordinate Precision - number of digits after the decimal point for exported coordinates.
- Export parameters. For each layer the user can assign different parameters. See [Google Earth general](#) for descriptions of the parameters

Notes:

- For each layer the classification used in ArcMap will be exported with the symbology assigned.
 - Only single field unique value classification can be exported
 - For graduated classifications the use of Normalization field is supported.
 - For point layers only the size and the color of the ArcGIS markers will be used. Google Icons need to be selected for each layer.
 - Google icons need to be selected for Labels for polyline and polygon layers
- The Name field values will be used for the name of the feature and for the Label
- Transparency for each layer can be defined in the Transp field. Value of 0 means opaque and 100 means totally transparent.
- To the Map description the software will add "Exported with ET GeoWizards". Do not remove ET GeoWizards if you want to import the KML back to ArcGIS. This is used to identify that the KML structure is created by ET GeoWizards and more effective algorithm is used for import of the KML file.
- See [Google Earth general](#) for important additional information.

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.

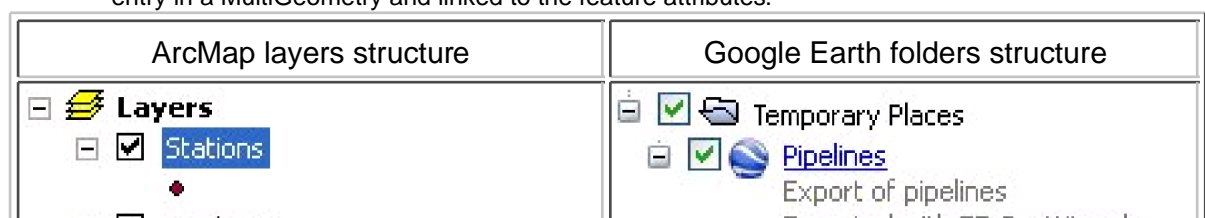
Google Earth projection

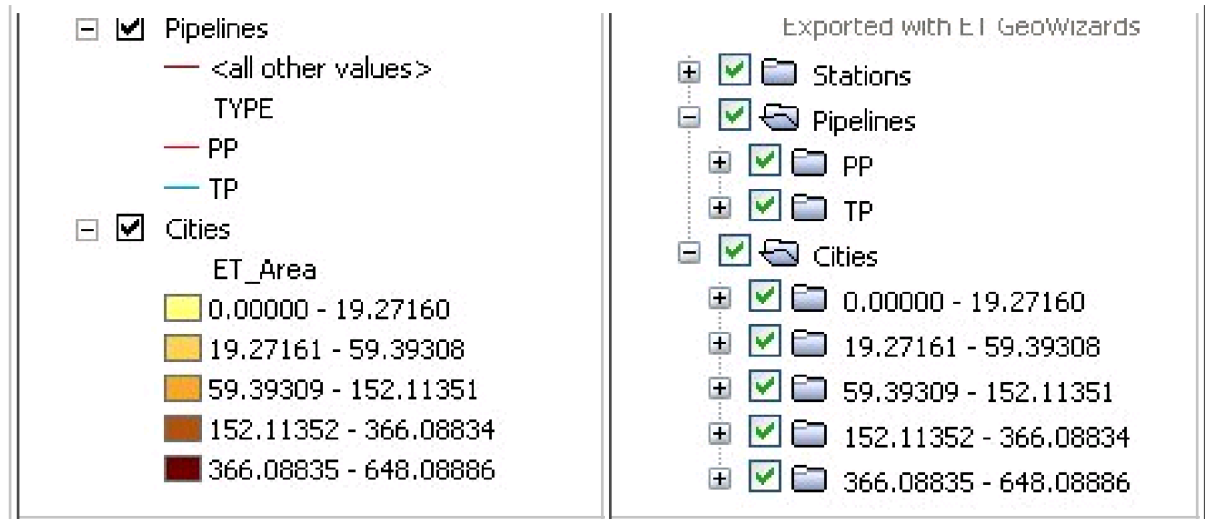
For its reference system, KML uses Geographic Coordinate System (GCS) with WGS84 datum. In ArcGIS this projection is called GCS_WGS_1984. The export to Google Earth functions of ET GeoWizards project the data on the fly to GCS_WGS_1984. If the source data is in a projection that have different datum, 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. A single KML file can contain several feature classes of different types (Point, Polyline and Polygon). The Export To Google Earth function of ET GeoWizards structure the data in the following manner:
 - Each layer has its own folder
 - If a layer is classified each class will have a subfolder
 - If the info points have been exported for polyline and polygon layers, they are written as a separate entry in a MultiGeometry and linked to the feature attributes.





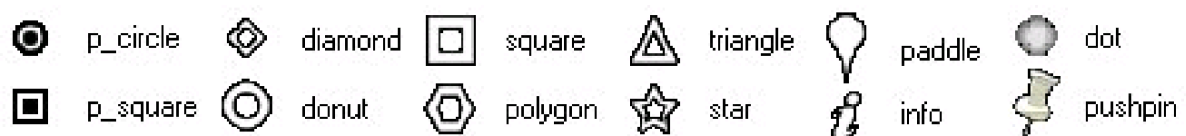
As of version 10.0 the export function of ET GeoWizards supports multipart features, which are exported as MultiGeometry in the KML file. These features will consist of several not connected geometries and will have a single label point. If the same features are imported back, each geometry will be created 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:

- Export To Google Earth GUI function - The symbols assigned in the map
- Feature Class To Google Earth scripting and toolbox function
 - 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.
 6. Editing attributes in Google Earth. The attributes cannot be edited in Google Earth. The only way to edit attributes and send them back to ArcGIS is to use the Name and the Description of the Google Earth features.

All ESRI products mentioned are trademarks of Environmental Systems Research Institute, Inc.

Google and Google Earth are trademarks of Google Inc

Copyright © Ianko Tchoukanski

Export To Google Earth

[\(See important information here\)](#)

This function is not available via the GUI of ET GeoWizards.













Converts a feature class to a Google Earth KML or KMZ file. Available only in the ToolBox implementation. See also the [Map to Google Earth function](#) available via the user interface.

Command line syntax

```
ET_GPEExportToGoogleEarth <input_dataset> <out_file> {KML_Description} {label_field} {description_field} {transparency}  
{Z_Source} {z_field} {z_constant} {z_units} {z_type} {extrude_geometries} {attributes} {marker_symbol} {marker_scale}  
{line_width} {export_info_points} {info_symbol} {info_scale} {coordinate_precision}
```

Expression	Explanation
<input_dataset>	A feature class or 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_Description}	A String that will be used for a general description of the KML file.
{label_field}	A String representing a field name. The values in this field will be used for naming the Google Earth features.
{description_field}	Required. A String representing a field name. The values in this field will be used for description of the Google Earth features. Read the general Google Earth notes.
{transparency}	A Double indicating the transparency to be used. 0 = Opaque, 100 = invisible
{Z_Source}	A String indicating what will be the source for the elevation values. Valid strings: <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}	A String indicating how the Z values will be interpreted. Valid strings: <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.
{attributes}	A String that indicates whether and how the attributes will be exported. Valid strings: <ul style="list-style-type: none">• "Features" - the attributes will be exported to the actual features• "Labels" - the attributes will be exported to the Info - Points• "Both" - the attributes will be attached to both, the actual features and the info points

- "None" - no attributes will be exported.

{marker_symbol}	<p>A String indicating which of the available Google Earth symbols will be used for point features. The available symbols are:</p> <div>  p_circle  diamond  square  triangle  paddle  dot </div> <div>  p_square  donut  polygon  star  info  pushpin </div>
{marker_scale}	A Double indicating the size of the Icon for the point features. Actually this is a scale factor for the Google Earth markers - values of 0.5 to 1.5 will give good results.
{line_width}	A Double representing the width of the Polyline features and outline width for Polygon features
{export_info_points}	A Boolean indicating whether the Info-Points will be exported.
{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_scale}	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.

Scripting syntax

ET_GPExportToGoogleEarth (input_dataset out_file KML_Description label_field description_field transparency Z_Source z_field z_constant z_units z_type extrude_geometries attributes marker_symbol marker_scale line_width export_info_points info_symbol info_scale coordinate_precision)

See the explanations above:

<> - required parameter

{ } - optional parameter

Vector Grid

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates a polyline or polygon vector grid using user defined extents and cell size.

Inputs:

- Initial extent of the grid - the grid will be generated using the coordinate system and units of:
 - the current view
 - a layer from the current view
- Output spatial reference - the grid will be stored in the spatial reference of
 - the current view
 - a layer from the current view
- Grid type
 - Polyline
 - Polygon
- Cell size in X and Y directions

Outputs:

- New Polyline or Polygon feature class

Notes:

- The initial extents of the grid are defined by the extents of the selected layer or current extents of the data frame. The extents can be changed manually during the dialog.
- The cell size will be in the units of the input layer or data frame. If the input is Unprojected (Geographic "Projection") there is an option to input values in Degrees - Minutes - Seconds
- If the input is Unprojected only values between X = -180 To 180 and Y = -90 To 90 will be accepted
- If the Input is Projected and the Output is in different projection, the user should make sure that the Grid extents are valid
- In order to avoid incorrect inputs, the size of the grid is limited to 4,000,000 cells (2000 by 2000)
- 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.

ToolBox implementation

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

Command line syntax - ET_GPVectorGridExtents

ET_GPVectorGridExtents <out_feature_class> {out_spatial_reference} <grid_type> {get_extents_from} <Extent>
<X_Cell_Size> <Y_Cell_Size>

Parameters

Expression	Explanation
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<grid_type>	A String controlling the shape type of the output point grid. Valid values "Polygon", "Polyline"
{get_extents_from}	A String - a layer name or the full name of the a feature class to be used as a source for the extents (see examples below)

<Extent>	An Envelope representing the extents - "XMin YMin XMax YMax"
<X_Cell_Size>	A Double representing the size of the call in X direction
<Y_Cell_Size>	A Double representing the size of the call in Y direction.

Scripting syntax - ET_GPVectorGridExtents

ET_GPVectorGridExtents (out_feature_ class out_spatial_reference grid_type get_extents_from Extent X_Cell_Size Y_Cell_Size)

Command line syntax - ET_GPVectorGridOrigin

ET_GPVectorGridOrigin <out_feature_class> {out_spatial_reference} <grid_type> <X_origin> <Y_origin> <Number_columns> <Number_rows> <X_Cell_Size> <Y_Cell_Size>

Parameters

Expression	Explanation
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<grid_type>	A String controlling the shape type of the output point grid. Valid values "Polygon", "Polyline"
<X_origin>	A Double representing the X coordinate of the lower left corner of the grid.
<Y_origin>	A Double representing the Y coordinate of the lower left corner of the grid.
<Number_Columns>	A Long representing the number of columns that the resulting grid will have
<Number_Rows>	A Long representing the number of rows that the resulting grid will have
<X_Cell_Size>	A Double representing the size of the call in X direction
<Y_Cell_Size>	A Double representing the size of the call in Y direction.

Scripting syntax - ET_GPVectorGridOrigin

ET_GPVectorGridOrigin (out_feature_ class out_spatial_reference grid_type X_origin Y_origin Number_Columns Number_Rows X_Cell_Size Y_Cell_Size)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

VectorGridExtent(sOutFName As String, sGridType As String, pExtent As IEnvelope, dCellSizeX As Double, dCellSizeY As Double, Optional pOutSref As ISpatialReference = Nothing) As IFeatureClass

VectorGridOrigin(sOutFName As String, sGridType As String, dLowerLeftX As Double, dLowerLeftY As Double, iNumColumns As Integer, iNumRows As Integer, dCellSizeX As Double, dCellSizeY As Double, Optional pOutSref As ISpatialReference = Nothing) As IFeatureClass

Point Grid

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

- Initial extent of the grid - the grid will be generated using the coordinate system and units of:
 - the current view
 - a layer from the current view
- Output spatial reference - the grid will be stored in the spatial reference of
 - the current view
 - a layer from the current view
- Grid type
 - Square
 - Rectangle
 - Triangle - equilateral
- Distance between the points - in the case of rectangle X and Y spacing are required

Outputs:

- New Point feature class

Notes:

- The initial extents of the grid are defined by the extents of the selected layer or current extents of the data frame. The extents can be changed manually during the dialog.
- The cell size will be in the units of the input layer or data frame. If the input is Unprojected (Geographic "Projection") there is an option to input values in Degrees - Minutes - Seconds
- If the input is Unprojected only values between X = -180 To 180 and Y = -90 To 90 will be accepted
- If the Input is Projected and the Output is in different projection, the user should make sure that the Grid extents are valid
- In order to avoid incorrect inputs, the size of the grid is limited to 4,000,000 cells (2000 by 2000)
- 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"

ToolBox implementation - two tools are available for the Point Grid function

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

Command line syntax - ET_GPPointGridOrigin

ET_GPPointGridOrigin <out_feature_class> {out_spatial_reference} <grid_type> <x_origin> <y_origin> <number_columns> <number_rows> <column_width> {row_height}

Parameters

Expression	Explanation
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<grid_type>	A String controlling the type of the output point grid. Valid values "Square", "Rectangle" or "Triangle"
<x_origin>	A Double representing the X coordinate of the lower left point of the grid.

<y_origin>	A Double representing the Y coordinate of the lower left point of the grid.
<number_columns>	A Long representing the number of columns that the resulting grid will have
<number_rows>	A Long representing the number of rows that the resulting grid will have
<column_width>	A Double representing the the width of each grid column (X - cell size)
{row_height}	A Double representing the the height of each grid column (Y - cell size). Needed only for a "Rectangle" type grid. If not used row_height = column_width will be assigned.

Scripting syntax - ET_GPPointGridOrigin

ET_GPPointGridOrigin (out_feature_ class out_spatial_reference grid_type x_origin y_origin number_columns number_rows column_width row_height)

Command line syntax - ET_GPPointGridExtents

ET_GPPointGridExtents <out_feature_class> {out_spatial_reference} <grid_type> {get_extents_from} <Extent> <X_Cell_Size> <Y_Cell_Size>

Parameters

Expression	Explanation
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<grid_type>	A String controlling the type of the output point grid. Valid values "Square", "Rectangle" or "Triangle"
{get_extents_from}	A String - a layer name or the full name of the a feature class to be used as a source for the extents (see examples below)
<Extent>	A Double representing the Y coordinate of the lower left corner of the grid. (see examples below)
<X_Cell_Size>	A Double representing the size of the call in X direction
<Y_Cell_Size>	A Double representing the size of the call in Y direction.

Scripting syntax - ET_GPPointGridExtents

ET_GPPointGridExtents (out_feature_ class out_spatial_reference grid_type get_extents_from Extent X_Cell_Size Y_Cell_Size)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointGridExtent(sOutFName As String, sGridType As String, pExtent As IEnvelope, dCellSizeX As Double, dCellSizeY As Double, Optional pOutSref As ISpatialReference = Nothing) As IFeatureClass

PointGridOrigin(sOutFName As String, sGridType As String, dLowerLeftX As Double, dLowerLeftY As Double, iNumColumns As Integer, iNumRows As Integer, dCellSizeX As Double, dCellSizeY As Double, Optional pOutSref As ISpatialReference =

Nothing) As IFeatureClass

Copyright © Ianko Tchoukanski

Random Points On Polylines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

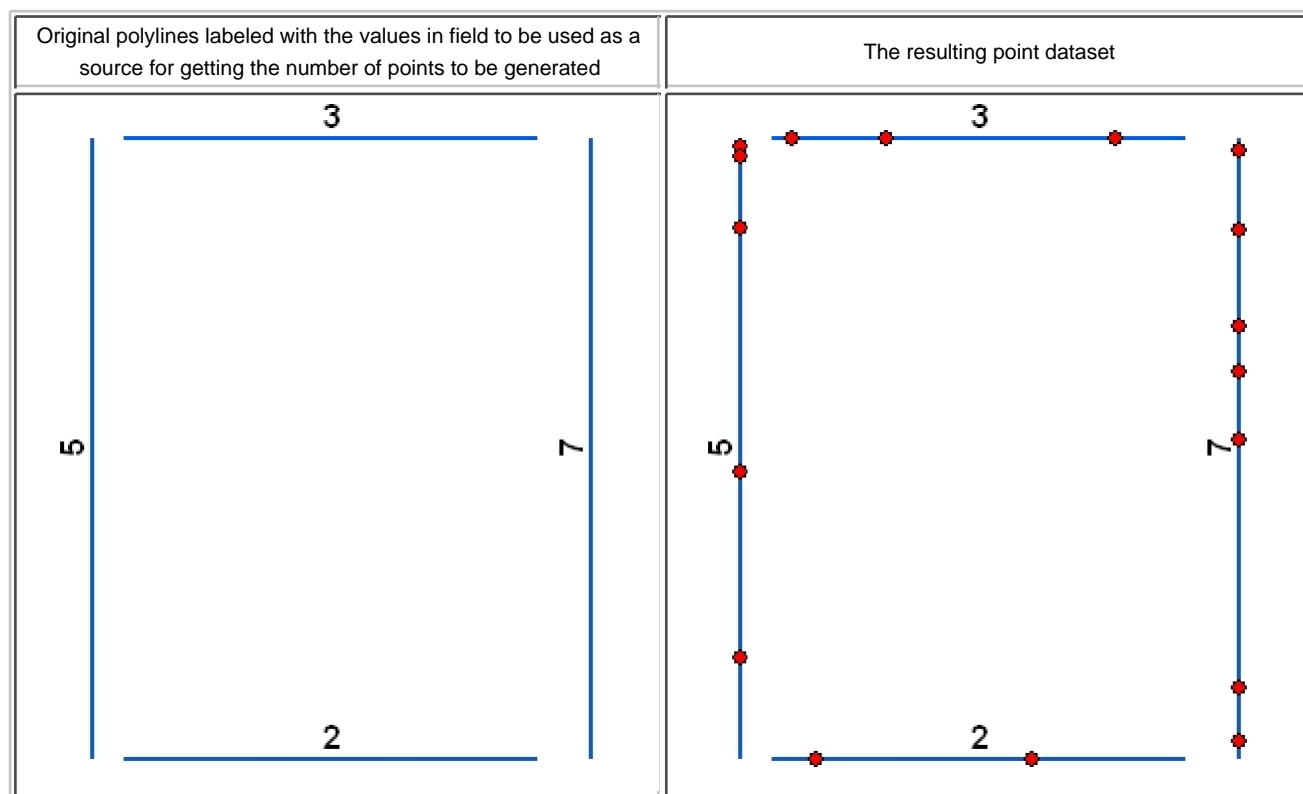
- A polyline feature class
- The number of points per polyline can be input in one of the following ways
 - A numeric field which values will be used to get the number of points to be generated per polyline.
 - A constant number
- Optional: The maximum number of points per polyline. This parameter is to enforce max number of points if a field is used for getting the number of points to be generated. For example if a field is selected and the values in this field range from 5 to 50,000. And the maximum number of points is set to 100, on the polylines that have values > 100 in the field only 100 points will be generated. The default value is 500.
- Optional: Minimum Distance from boundary - no point will be generated that is closer than this tolerance to the ends of the polyline (introduced in version 10.2).

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:



ToolBox implementation

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

Command line syntax

ET_GPRandomPointsOnPolylines<input_dataset> <out_feature class> {number_points_field} {number_points}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class.
{number_points_field}	Optional. A String numeric field which values will be used to get the number of points to be generated per polyline.
{number_points}	Optional. A number that is used in two ways <ul style="list-style-type: none">● If a number_points_field is specified, number_points is used as maximum points allowed per polyline● If a number_points_field is an empty string, number_points is used as a constant that define the number of points to be generated per polyline

Note: One of number_points_field or number_points should be specified.

Scripting syntax

ET_GPRandomPointsOnPolylines(input_dataset, out_feature class, number_points_field, number_points)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

RandomPointsOnPolylines(pInFC As IFeatureClass, sOutFName As String, Optional iNumberPoints As Integer = 0, Optional sNumberPointsField As String = "", Optional dMinDistanceToEnds As Double = 0) As IFeatureClass

Random Points In Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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: The maximum number of points per polygon. This parameter is to enforce max number of points if a field is used for getting the number of points to be generated. For example if a field is selected and the values in this field range from 5 to 50,000. And the maximum number of points is set to 100, on the polygons that have values > 100 in the field only 100 points will be generated. The default value is 500.
- Optional: Minimum Distance from boundary - no point will be generated that is closer than this tolerance to the boundary of the polygon (introduced in version 10.2).
- Optional: Minimum distance between the points (introduced in version 11.0)

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

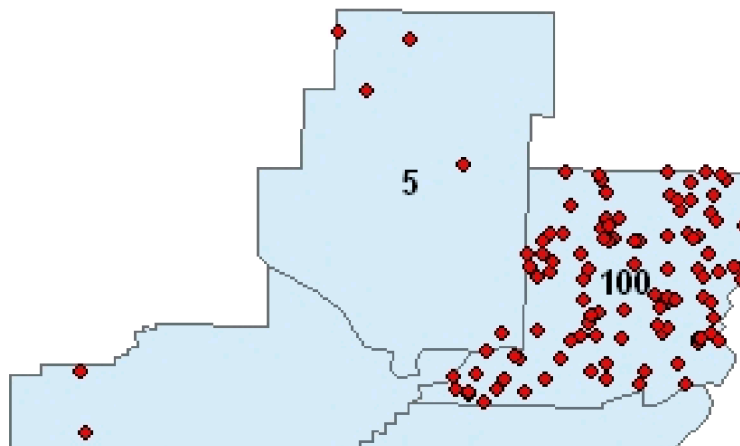
Notes:

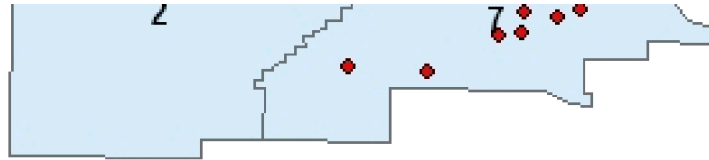
- If the number of points specified cannot be placed in specific polygon, a message will be stored in the log file.
- The larger minimum distance between the points specified, the more uniform the points created will be.

Illustration:

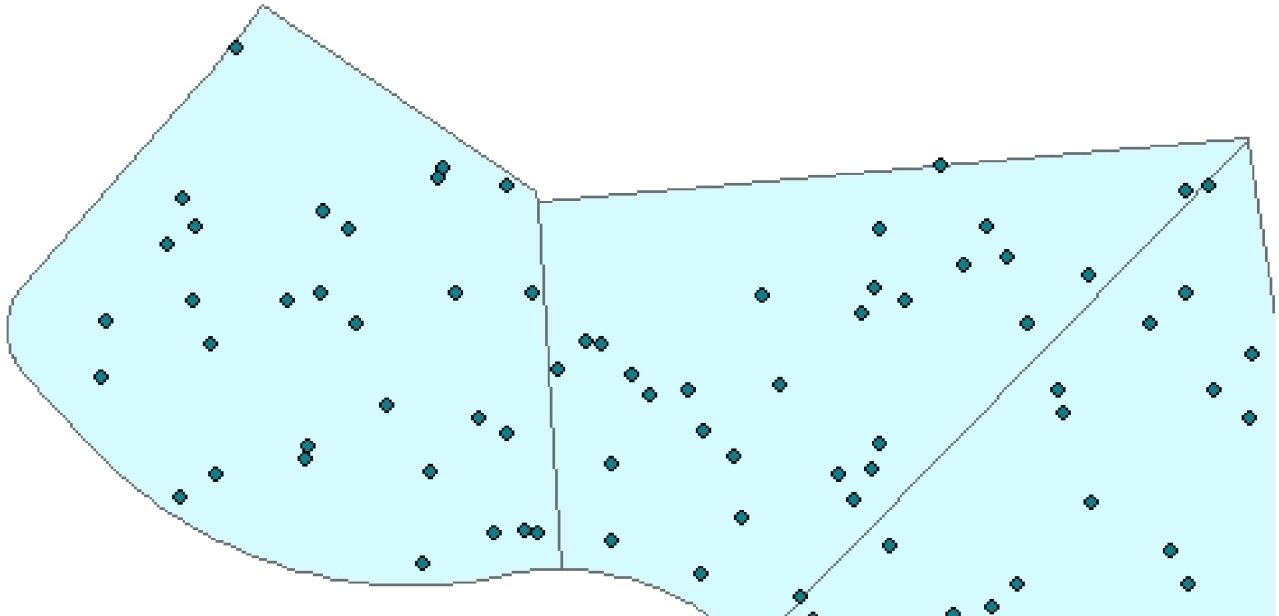
Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points.

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

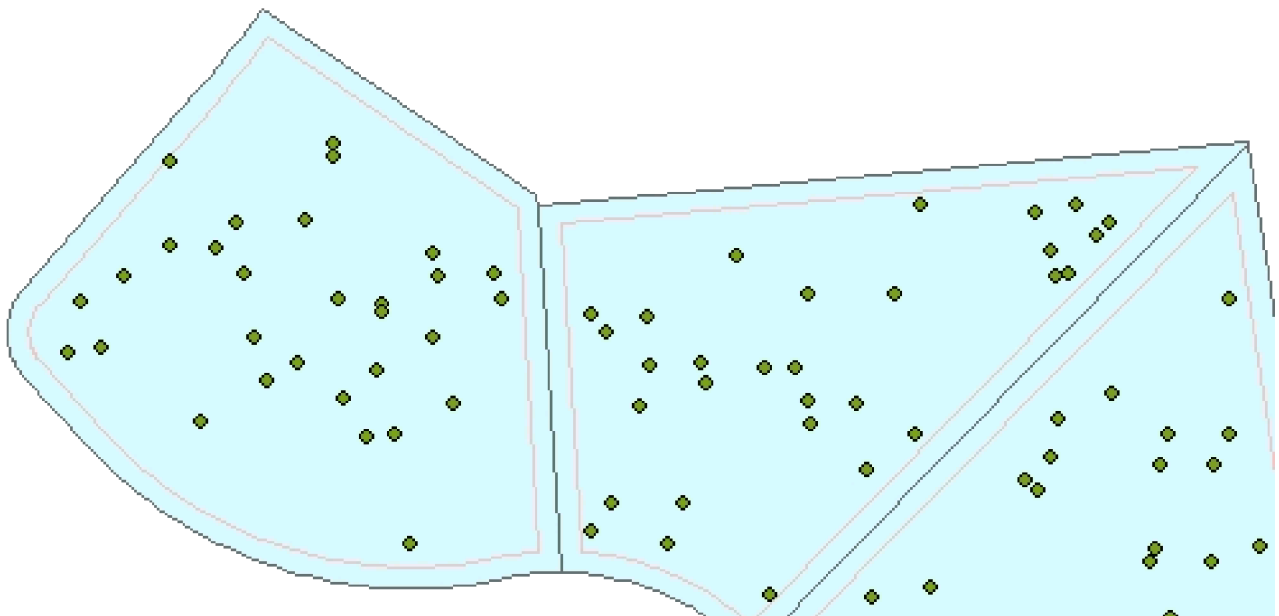




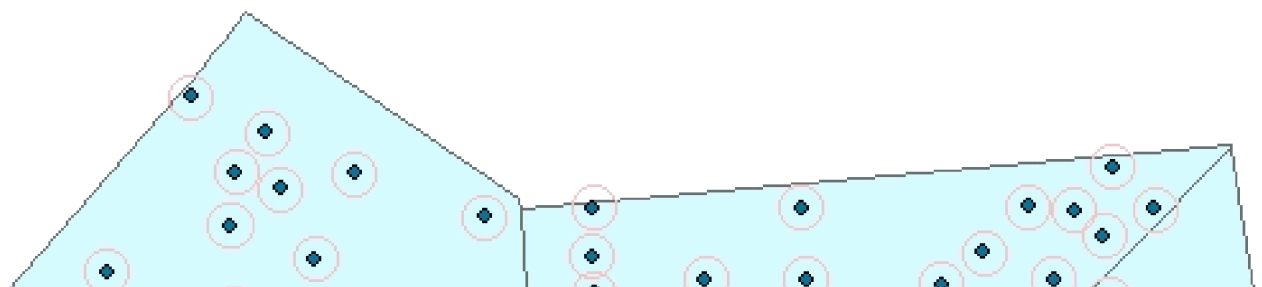
30 points per polygon.
 Distance between points = 0, Minimum distance to boundary = 0

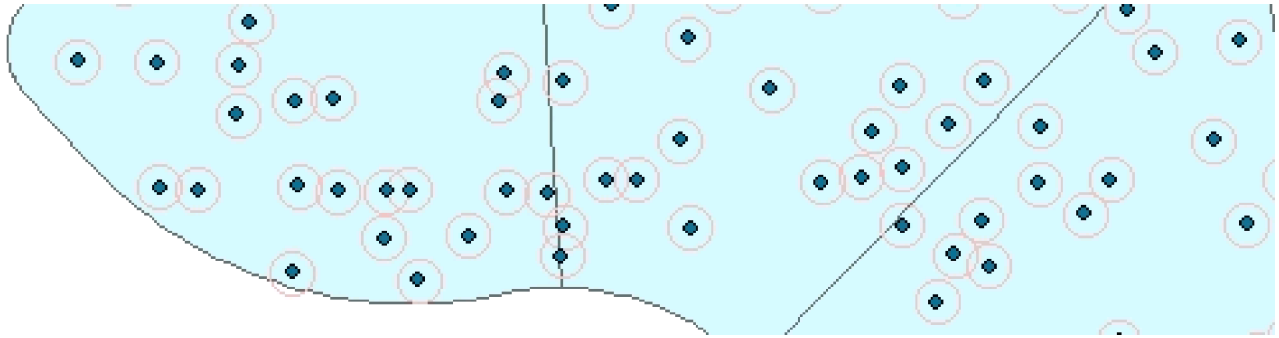


30 points per polygon.
 Distance between points = 0, Minimum distance to boundary = 5



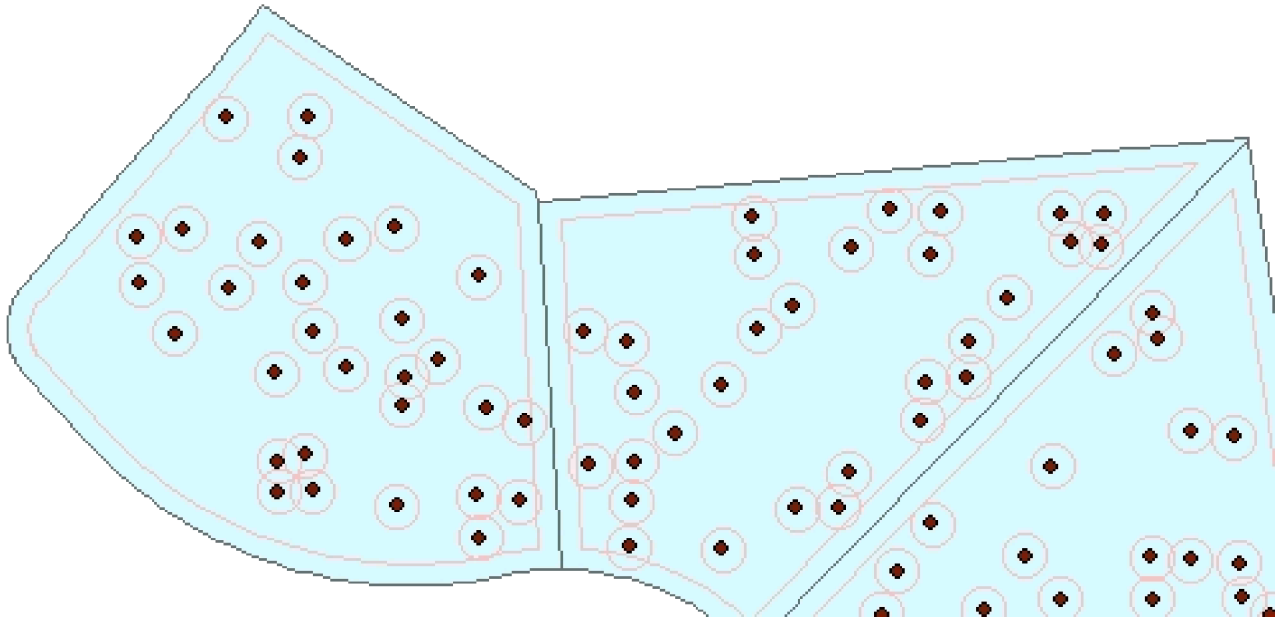
30 points per polygon.
 Distance between points = 5, Minimum distance to boundary = 0





30 points per polygon.

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



ToolBox implementation

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

Command line syntax

ET_GPRandomPointsInPolygons<input_dataset> <out_feature_class> {number_points_field} {number_points}
{min_distance_between_points} {min_distance}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
{number_points_field}	A String representing the name of a numeric field which values will be used to get the number of points to be generated per polygon.
{number_points}	Optional. A number that is used in two ways <ul style="list-style-type: none"> ● If a number_points_field is specified, number_points is used as maximum points allowed per polygon ● If a number_points_field is an empty string, number_points is used as

a constant that define the number of points to be generated per polygon

{min_distance_between_points} A Double specifying the minimum distance between the points.

{min_distance} A Double specifying the minimum distance from boundary.

Note: One of number_points_field or number_points should be specified.

Scripting syntax

ET_GPRandomPointsInPolygons(input_dataset, out_feature_class, number_points_field, number_points, min_distance_between_points, min_distance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

RandomPointsInPolygons(pInFC As IFeatureClass, sOutFName As String, Optional iNumberPoints As Integer = 0, Optional sNumberPointsField As String = "", Optional dMinDistanceBetween As Double = 0, Optional dMinDistanceToBoundary As Double = 0) As IFeatureClass

Point Grids In Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

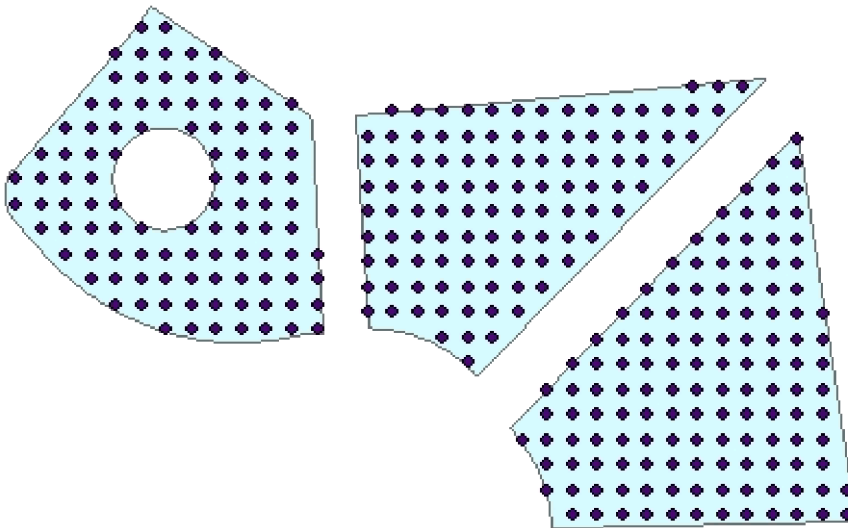
- A polygon feature class
- The distance between the points for each polygon can be input in one of the following ways
 - A numeric field which values will be used to get the distance between points for each polygon.
 - A constant number defining the distance between points for all polygons.
- Rotation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

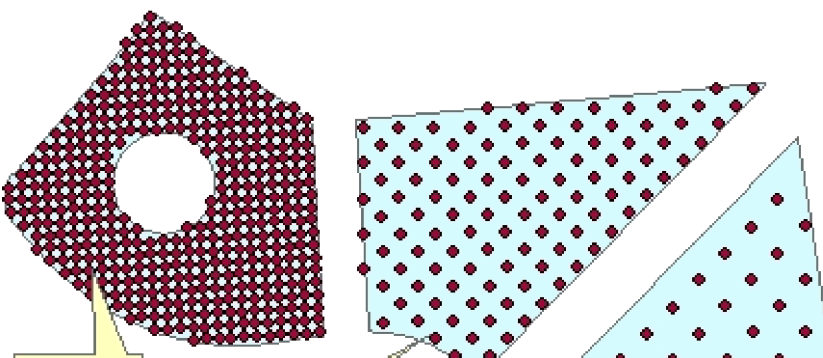
Illustration:

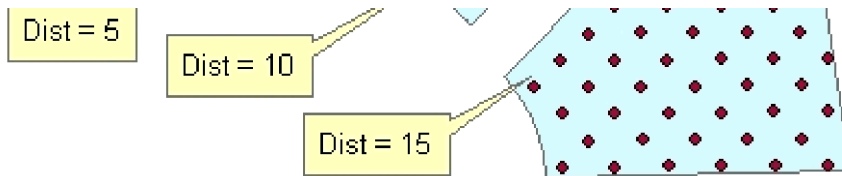
Distance between points = 10, Rotational angle = 0



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"





ToolBox implementation

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

Command line syntax

```
ET_GPPointGridsInPolygons<input_dataset> <out_feature_class> {distance} {distance_field} {angle_from} {rotation_angle}
{angle_field}
```

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
{distance}	A Double representing the distance between the points for all polygons
{distance_field}	A String representing the name of a numeric field which values will be used to get the distance between the points for each polygon
{angle_from}	A String defining how the rotation angle will be specified <ul style="list-style-type: none"> • "Constant" - 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) • "Longest Axis" - Along the longest axis of each polygon • "Longest Side" - Along the longest side of each polygon
{rotation_angle}	A Double - specifying the rotation angle for all polygons.
{angle_field}	A String - the name of the field to be used as source for rotation angle.

Scripting syntax

```
ET_GPPointGridsInPolygons(input_dataset, out_feature_class, distance, distance_field, angle_from, rotation_angle,
angle_field)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

```
PointGridsInPolygons(plnFC As IFeatureClass, sOutFName As String, sAngleFrom As String, Optional dDistance As Double =
0, Optional sDistanceField As String = "", Optional dAngle As Double = 0, Optional sAngleField As String = "") As
IFeatureClass
```

Square Grids In Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

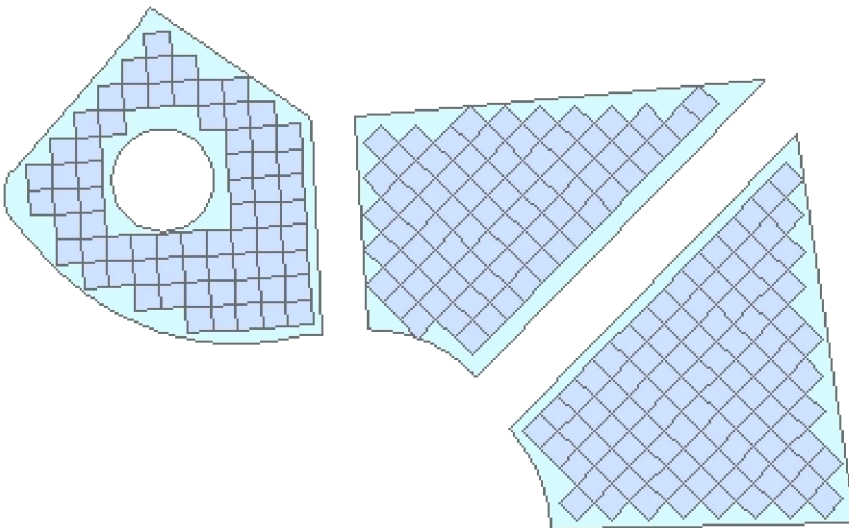
- A polygon feature class
- The cell size of the grid for each polygon can be input in one of the following ways
 - A numeric field which values will define the cell size for the grid for each polygon.
 - A constant number that defines the cell size (the same for all polygons)
- Rotation angle
 - Constant for all polygons - (in degrees starting from 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
- 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:

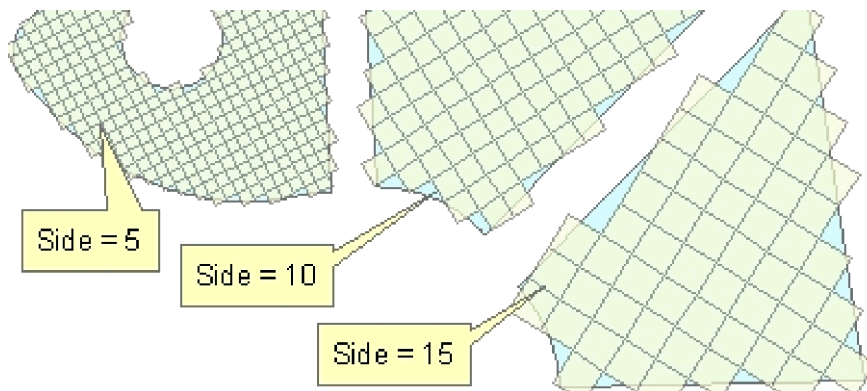
Cell size = 10, Rotational angle = "Along the longest side"
Completely inside = TRUE



Original polygons labeled with the values in field to be used as a source for the Cell Size.

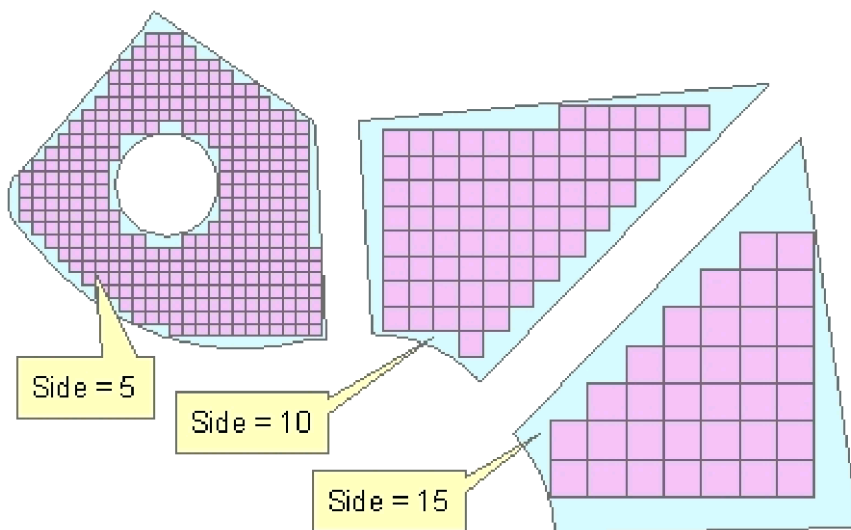
Rotational angle = "Along the longest Axis"
Completely inside = FALSE





Original polygons labeled with the values in field to be used as a source for the Cell Size.

Rotational angle = 0
Completely inside = TRUE



ToolBox implementation

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

Command line syntax

```
ET_GPSquareGridsInPolygons<input_dataset> <out_feature_class> {side_length} {side_length_field} {angle_from}
{rotation_angle} {angle_field} {inside_only}
```

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
{side_length}	A Double representing the cell size for all polygons
{side_length_field}	A String representing the name of a numeric field which values will be used to get the cell size for each polygon
{angle_from}	A String defining how the rotation angle will be specified <ul style="list-style-type: none"> • "Constant" - 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)

- "Longest Axis" - Along the longest axis of each polygon
- "Longest Side" - Along the longest side of each polygon

{rotation_angle}	A Double - specifying the rotation angle for all polygons.
{angle_field}	A String - the name of the field to be used as source for rotation angle.
{inside_only}	A Boolean defining whether the resulting square can intersect the polygon boundary or not.

Scripting syntax

ET_GPSquareGridsInPolygons(input_dataset, out_feature_class, side_length, side_length_field, angle_from, rotation_angle, angle_field, inside_only)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SquareGridsInPolygons(pInFC As IFeatureClass, sOutFName As String, sAngleFrom As String, Optional dCellSize As Double = 0, Optional sCellSizeField As String = "", Optional dAngle As Double = 0, Optional sAngleField As String = "", Optional bInsideOnly As Boolean = False) As IFeatureClass

Uniform Points In Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

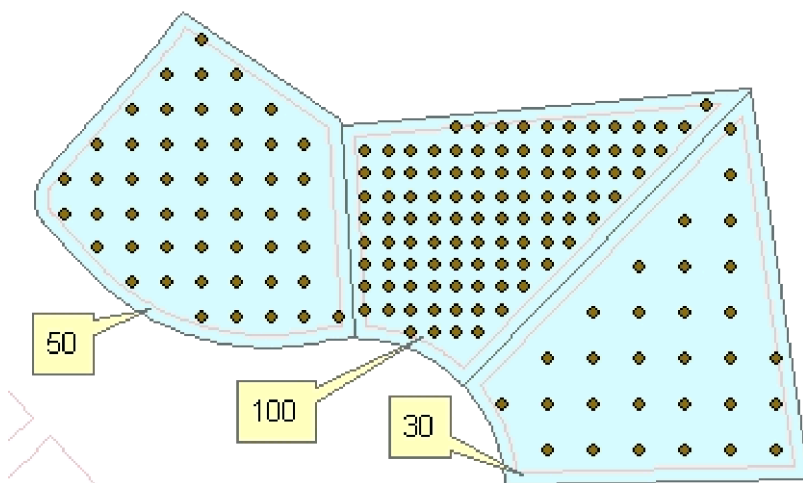
Outputs:

- 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:

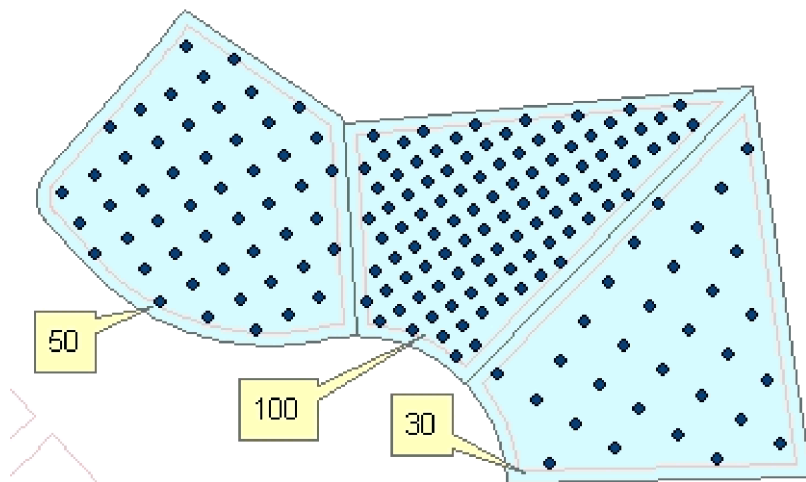
Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points.

Rotational angle = 0, Minimum distance to boundary = 5



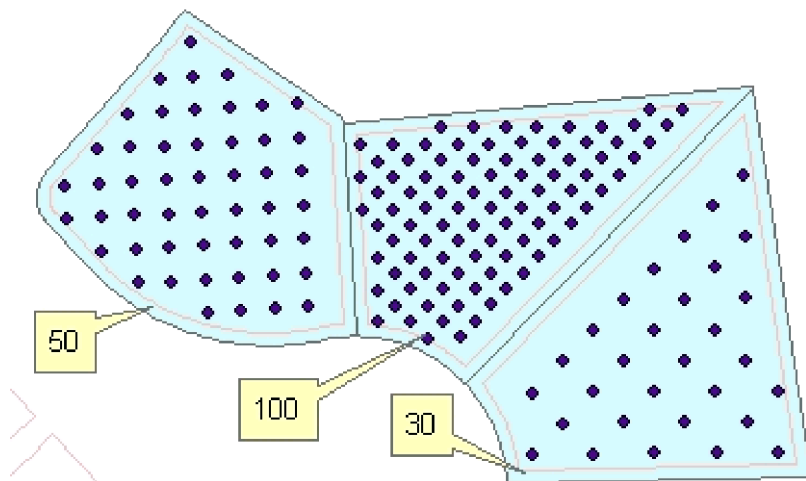
Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points.

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



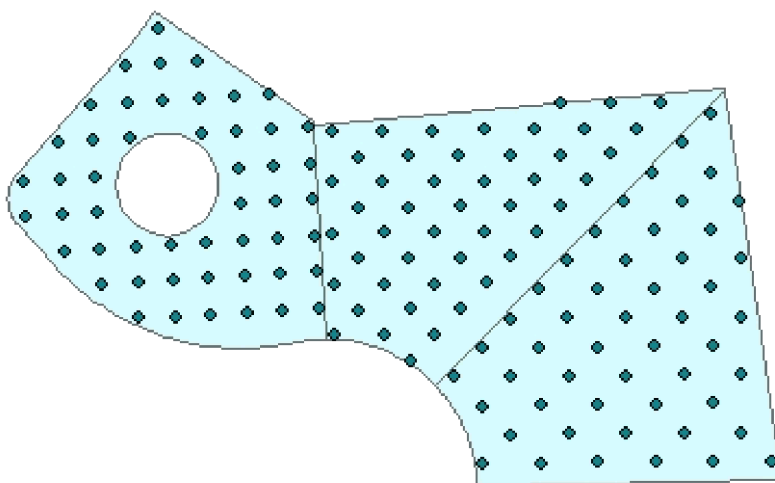
Original polygons labeled with the values in field to be used as a source for getting the number of points to be generated and the resulting points.

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



ToolBox implementation

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

Command line syntax

ET_GPUniformPointsInPolygons<input_dataset> <out_feature_class> {number_points_field} {number_points} {min_distance}
{angle_from} {rotation_angle} {angle_field}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
{number_points_field}	A String representing the name of a numeric field which values will be used to get the number of points to be generated per polygon.
{number_points}	A Double representing the number of points for to be created in each polygons (the same for all polygons)
{min_distance}	A Double specifying the minimum distance from boundary.
{angle_from}	A String defining how the rotation angle will be specified <ul style="list-style-type: none">• "Constant" - 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)• "Longest Axis" - Along the longest axis of each polygon• "Longest Side" - Along the longest side of each polygon
{rotation_angle}	A Double - specifying the rotation angle for all polygons.
{angle_field}	A String - the name of the field to be used as source for rotation angle.

Scripting syntax

ET_GPUniformPointsInPolygons(input_dataset, out_feature_class, number_points_field, number_points, min_distance, angle_from, rotation_angle, angle_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

UniformPointsInPolygons(plnFC As IFeatureClass, sOutFName As String, sAngleFrom As String, Optional iNumberPoints As Integer = 0, Optional sNumberPointsField As String = "", Optional dAngle As Double = 0, Optional sAngleField As String = "", Optional dMinDistanceToBoundary As Double = 0) As IFeatureClass

Create Tiles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates a regular polygons grid with user defined extents, tile shape and size.

Inputs:

- Initial extent of the grid - the grid will be generated using the coordinate system and units of:
 - the current view
 - a layer from the current view
- Output spatial reference - the grid will be stored in the spatial reference of
 - the current view
 - a layer from the current view
- Grid type
 - Triangle
 - Square
 - Hexagon
- Output spatial reference
- Cell size.
- 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.
- Shape orientation (for square and hexagon tiles only) - see examples below
 - Flat
 - Pointy

Outputs:

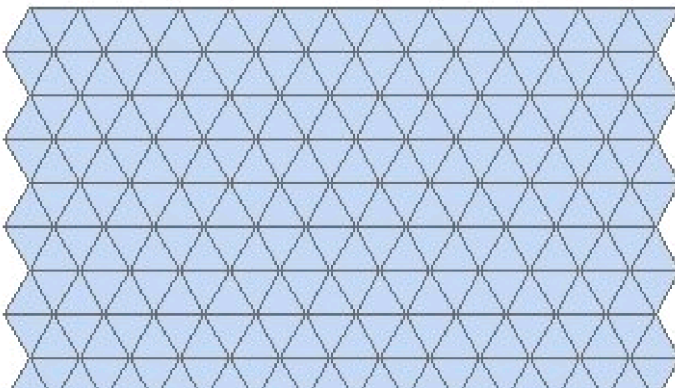
- New Polygon feature class

Notes:

- The initial extents of the grid are defined by the extents of the selected layer or current extents of the data frame. The extents can be changed manually during the dialog.
- The cell size will be in the units of the input layer or data frame.
- If the input is Unprojected only values between X = -180 To 180 and Y = -90 To 90 will be accepted
- If the Input is Projected and the Output is in different projection, the user should make sure that the Grid extents are valid
- In order to avoid incorrect inputs, the size of the grid is limited to 4,000,000 cells (2000 by 2000)
- A ET_Index field will be added to the attribute table. The values will indicate the index of each Grid cell. The cell in the bottom left corner of the Grid will have an index of "0-0".

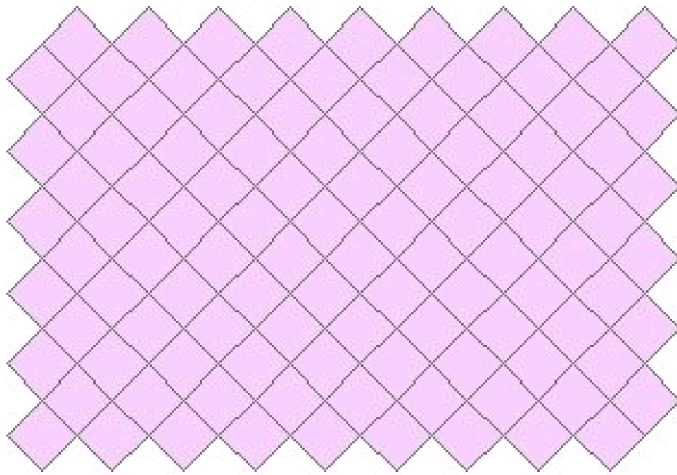
Examples:

Shape = Triangle

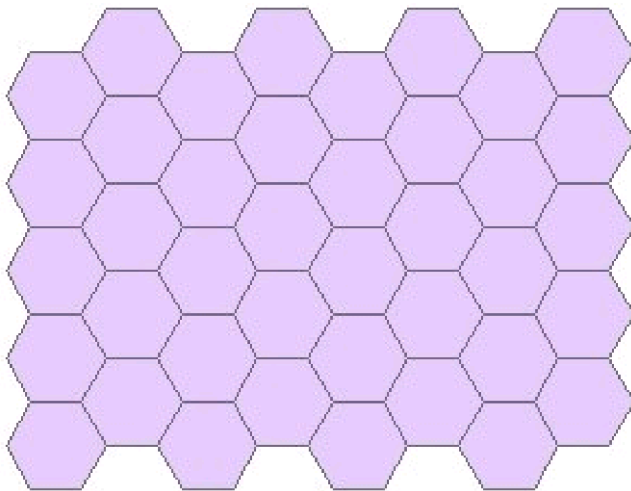




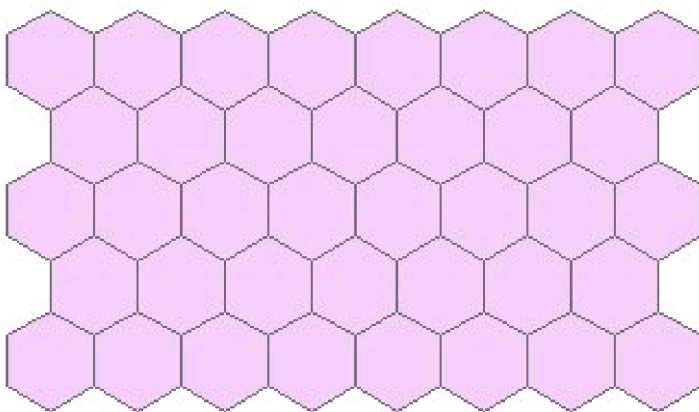
Shape = Square, Orientation = "Pointy"



Shape = Hexagon, Orientation = "Flat"



Shape = Hexagon, Orientation = "Pointy"



ToolBox implementation

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

Command line syntax - ET_GPCreateTilesExtent

ET_GPCreateTilesExtent <out_feature_class> {out_spatial_reference} <shape_type> {get_extents_from} <Extent>
<Size_represents> <Size> {Orientation}

Parameters

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

<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<shape_type>	A String controlling the shape type of the output tiles - "Triangle", "Square" or "Hexagon"
{get_extents_from}	A String - a layer name or the full name of the a feature class to be used as a source for the extents.
<Extent>	An Envelope representing the extents - "XMin YMin XMax YMax" Required. A String indicating the meaning of the <SIZE> parameter
<Size_represents>	<ul style="list-style-type: none"> ● Radius In - radius of the inscribed circle ● Radius Out - radius of the circumscribed circle ● Side - the side of the polygon
<Size>	A Double representing the size (see above for options)
{Orientation}	A String representing the tile orientation - "Flat" or "Pointy" - for Square or Hexagon tiles only.

Scripting syntax - ET_GPCreateTilesExtent

ET_GPCreateTilesExtent (out_feature_class, out_spatial_reference, shape_type, get_extents_from, Extent, Size_represents, Size, Orientation)

Command line syntax - ET_GPCreateTilesOrigin

ET_GPCreateTilesOrigin <out_feature_class> {out_spatial_reference} <shape_type> <X_origin> <Y_origin> <Number_columns> <Number_rows> <Size_represents> <Size> {Orientation}

Parameters

Expression	Explanation
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	The spatial reference of the output feature class. If not specified the output will be created with Unknown coordinate system
<shape_type>	A String controlling the shape type of the output tiles - "Triangle", "Square" or "Hexagon"
<X_origin>	A Double representing the X coordinate of the lower left corner of the grid.
<Y_origin>	A Double representing the Y coordinate of the lower left corner of the grid.
<Number_Columns>	A Long representing the number of columns that the resulting grid will have
<Number_Rows>	A Long representing the number of rows that the resulting grid will have Required. A String indicating the meaning of the <SIZE> parameter
<Size_represents>	<ul style="list-style-type: none"> ● Radius In - radius of the inscribed circle ● Radius Out - radius of the circumscribed circle ● Side - the side of the polygon
<Size>	A Double representing the size (see above for options)

{Orientation} A String representing the tile orientation - "Flat" or "Pointy" - for Square or Hexagon tiles only.

Scripting syntax - ET_GPCreateTilesOrigin

ET_GPCreateTilesOrigin (out_feature_class, out_spatial_reference, shape_type, X_origin, Y_origin, Number_Columns, Number_Rows, Size_represents, Size, Orientation)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CreateTilesExtent(sOutFName As String, sShapeType As String, pExtent As IEnvelope, sSizeRepresents As String, dSize As Double, sOrientation As String, Optional pOutSref As ISpatialReference = Nothing) As IFeatureClass

CreateTilesOrigin(sOutFName As String, sShapeType As String, dLowerLeftX As Double, dLowerLeftY As Double, iNumColumns As Integer, iNumRows As Integer, sSizeRepresents As String, dSize As Double, sOrientation As String, Optional pOutSref As ISpatialReference = Nothing) As IFeatureClass

Clean Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Removes the duplicate points from a point feature data set.

Inputs:

- A point feature layer

Outputs:

- New Point feature class
 - 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 feature class 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"

ToolBox implementation

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

Command line syntax

ET_GPCleanPoints <input_dataset> <out_feature class> <duplicates_feature_class>

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{duplicates_feature_class}	A String - the full name of the output feature class that identifies the points removed as duplicates. (A feature class with the same full name should not exist)

Scripting syntax

ET_GPCleanPoints (input_dataset, out_feature class, duplicates_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanPoints(plnFC As IFeatureClass, sOutFName As String, sDupFName As String) As IFeatureClass

Connect Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 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.

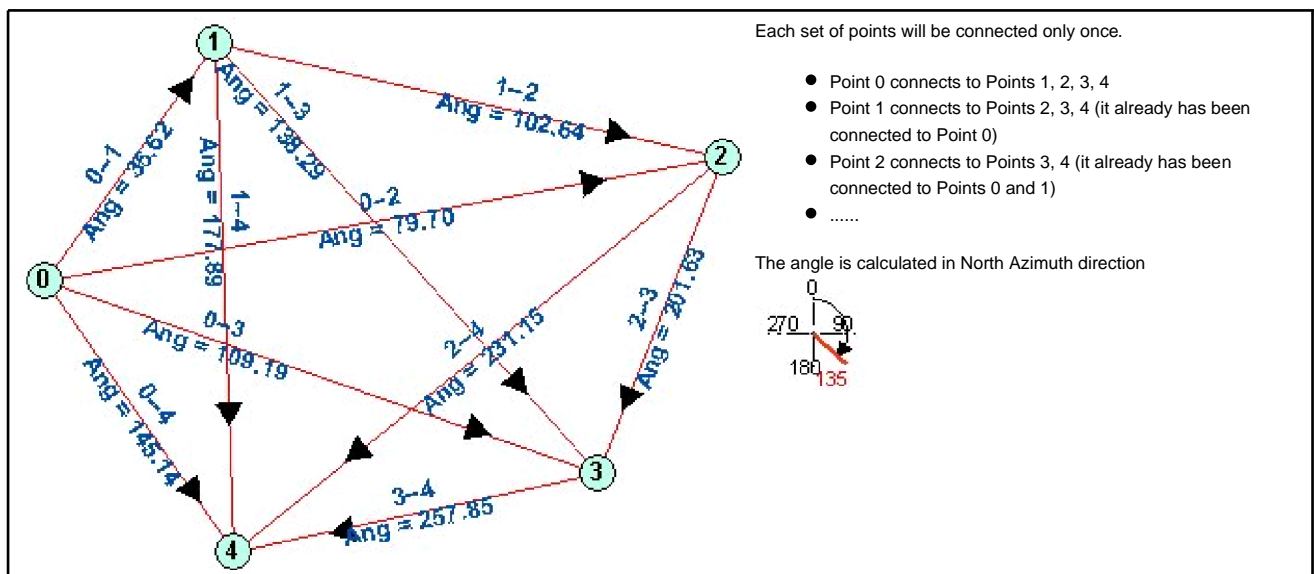
Outputs:

- New point feature class. 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:



ToolBox implementation

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

Command line syntax

ET_GPConnectPoints<input_dataset> <out_feature class> {Cutoff_distance}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Cutoff_distance}	A Double representing the Generalization tolerance.

Scripting syntax

ET_GPThinPoints (input_dataset, out_feature_class, Cutoff_distance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ConnectPoints(pInFC As IFeatureClass, sOutFName As String, dCutOff As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Disperse Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

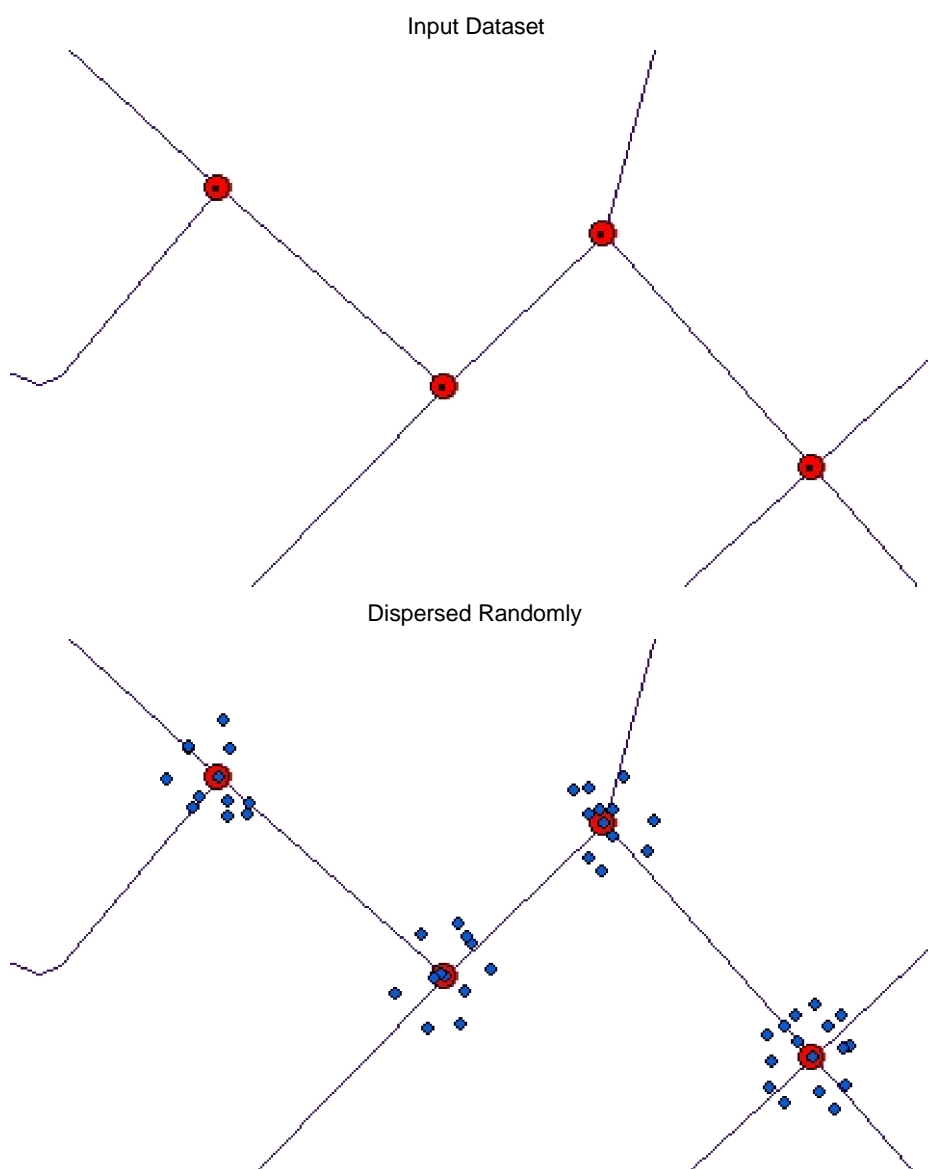
Inputs:

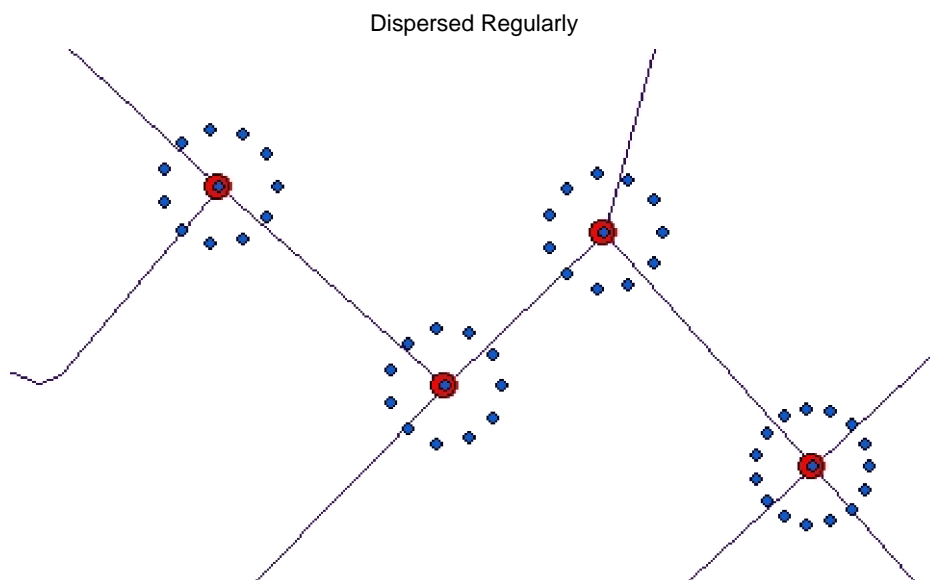
- A Point feature class
- Maximum allowed offset - the duplicate points will move no further than this tolerance from their original location.
- Dispersion method:
 - Random - the points will be relocated randomly within the specified allowed distance
 - Regular - the resulting points will be placed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





ToolBox implementation

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

Command line syntax

ET_GPDispersePoints<input_dataset> <out_feature_class> <tolerance> <disperse_method>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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 - "Random" or "Regular"

Scripting syntax

ET_GPDispersePoints(input_dataset, out_feature_class, tolerance, disperse_method)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

DispersePoints(pInFC As IFeatureClass, sOutFName As String, dTol As Double, sDisperseMethod As String) As IFeatureClass

Perpendiculars from Points to Polylines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

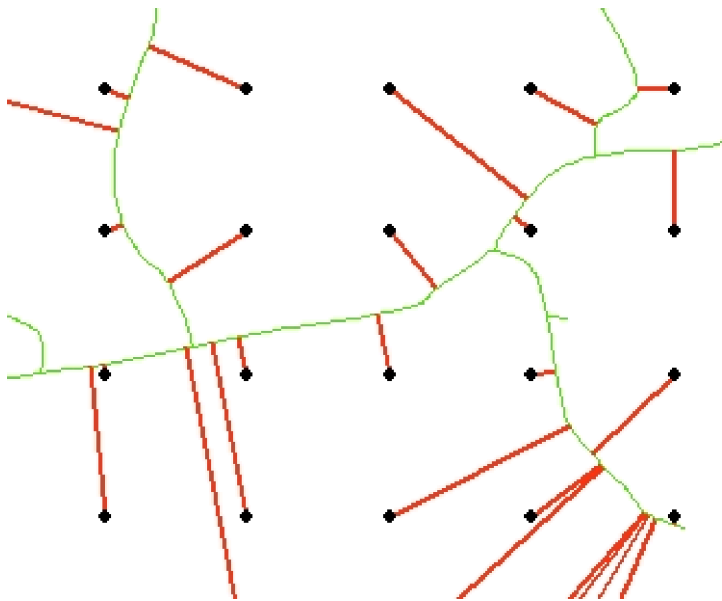
Inputs:

- A point feature class
- A reference polyline feature class
- Search tolerance.

Outputs:

- New Polyline feature class with lines from the source points and perpendicular to the closest polyline from the reference feature class.
- The attributes of the original points are transferred to the resulting lines.
- The following fields are added to the attribute table of the resulting feature class.
 - [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 feature class)

Illustration:



ToolBox implementation

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

Command line syntax

ET_GPPerpendicularsToPolylines<input_dataset> <Reference_dataset> <out_feature class> <search_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<Reference_dataset>	A Polyline feature class or feature layer.

<out_feature class>	A String - the full name of the output feature class.
<search_tolerance>	A Double representing the Search tolerance (in the units of the spatial reference of the input point dataset) to be used

Scripting syntax

ET_GPPerpendicularsToPolylines (input_dataset Reference_dataset out_feature class search_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PerpendicularsToPolylines(pInFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String, dSearchTol As Double) As IFeatureClass

Point Angle and Position

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

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 distances are calculated in the Spatial Reference of the input feature class
- All the attributes of the input point dataset are transferred to the output
- The function incorporates the functionality of the Point Distance and Measure Points functions available in the pre - 11.0 versions of ET GeoWizards.

ToolBox implementation

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

Command line syntax

ET_GPPointAngleAndPosition <input_dataset> <Reference_dataset> <out_feature_class> <search_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<Reference_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<search_tolerance>	A Double representing the Search tolerance (in the units of the calc_spatial_reference) to be used

Scripting syntax

ET_GPPointAngleAndPosition (input_dataset, Reference_dataset, out_feature_class, search_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointAngleAndPosition(pInFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String, dSearchTol As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Point Global Snap

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

- A point layer to be snapped
- A snap layer - point, polyline or polygon
- Snap tolerance
- Snap options

Outputs:

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

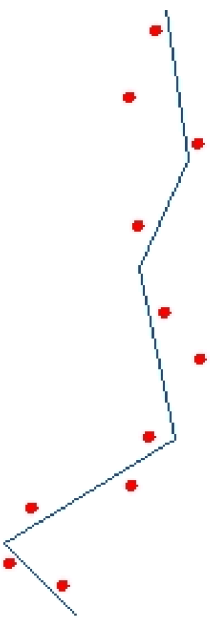
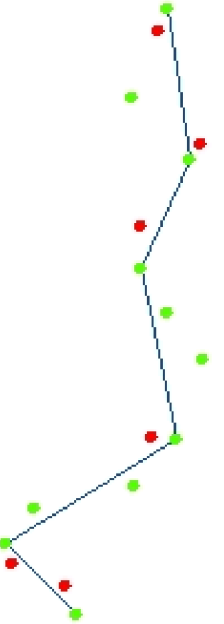
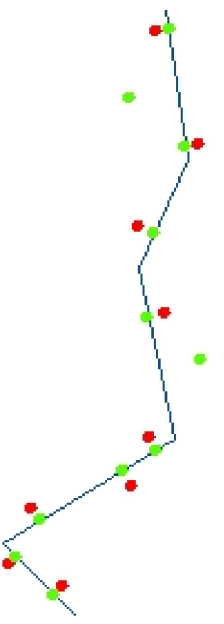
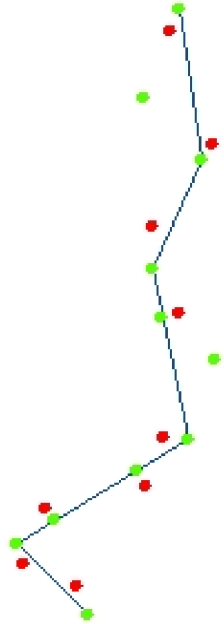
Options:

- Vertices: The points will be snapped to the nearest vertex of the nearest feature from the Snap layer
- Nearest edge: The points will be snapped to the nearest point of the nearest feature from the Snap layer
- Vertices & Edges: 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 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:

Before Snap	After Snap Option: Vertices	After Snap Option: Nearest Edge	After Snap Option: Vertices & Edges
			

ToolBox implementation

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

Command line syntax

ET_GPSnapPoints <input_dataset> <Reference_dataset> <out_feature_class> <snap_tolerance> {snap_vertices}
{snap_nearest}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<Reference_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<snap_tolerance>	A Double representing the Search tolerance (in the units of the input_dataset) to be used
{snap_vertices}	A Boolean indicating whether snapping to the closest vertex of the nearest feature from the Reference_dataset to be used
{snap_nearest}	A Boolean indicating whether snapping to the nearest point of the nearest feature from the Reference_dataset to be used

Scripting syntax

ET_GPSnapPoints (input_dataset, Reference_dataset, out_feature_class, snap_tolerance, snap_vertices, snap_nearest)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SnapPoints(pInFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String, dSnapTol As Double, Optional bVertex As Boolean = False, Optional bNearest As Boolean = False, Optional bSnapToZ As Boolean = False) As IFeatureClass

Point Intersection

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

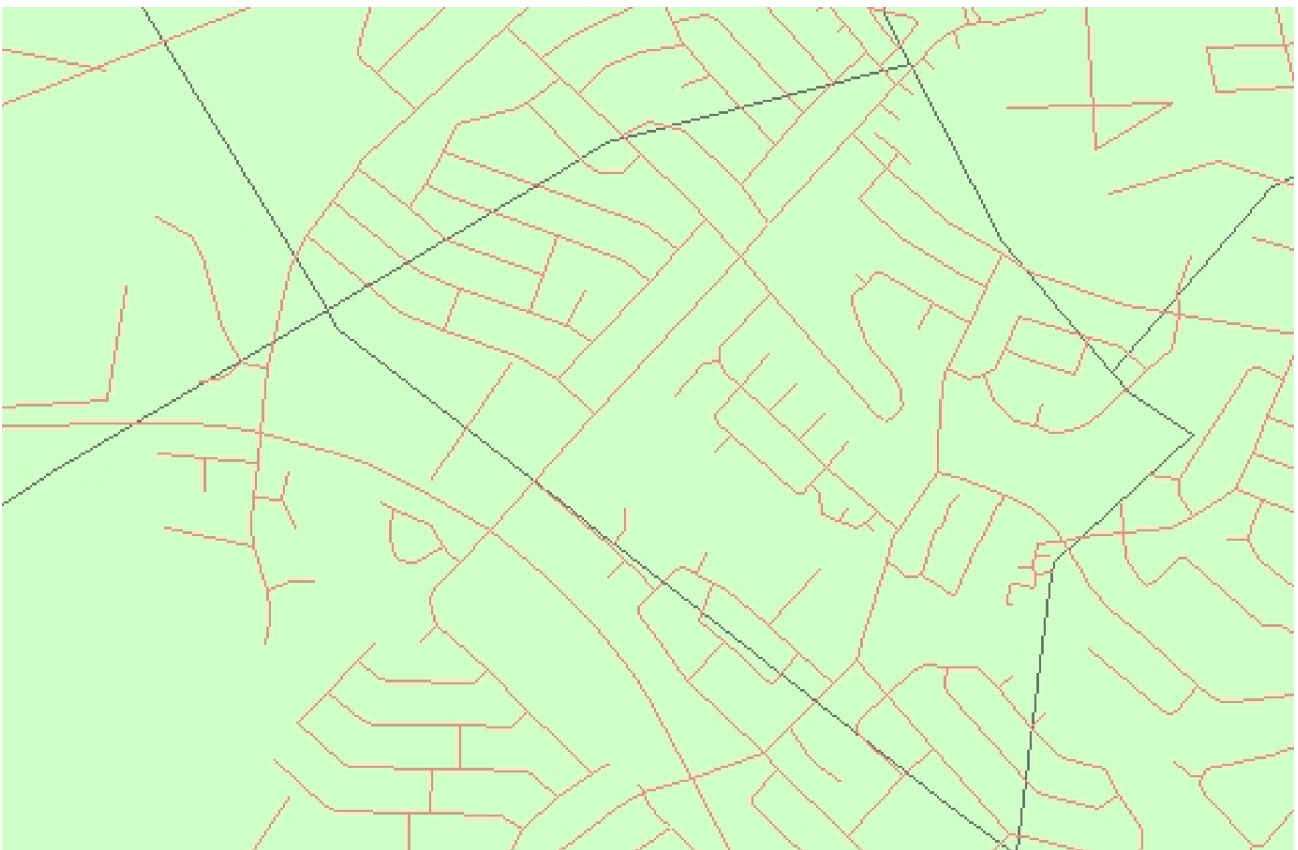
Outputs:

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

Notes:

- The two layers should have the same Spatial Reference
- 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





ToolBox implementation

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

Command line syntax

ET_GPPointIntersection <input_dataset> <second_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<second_dataset>	A Polyline or Polygon feature class or feature class. NOTE: The spatial references of <second_dataset> and the <input_dataset> must have the same Geographic Coordinate System
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPointIntersection (input_dataset, second_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointIntersection(plnFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String) As IFeatureClass

Points To Rectangles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

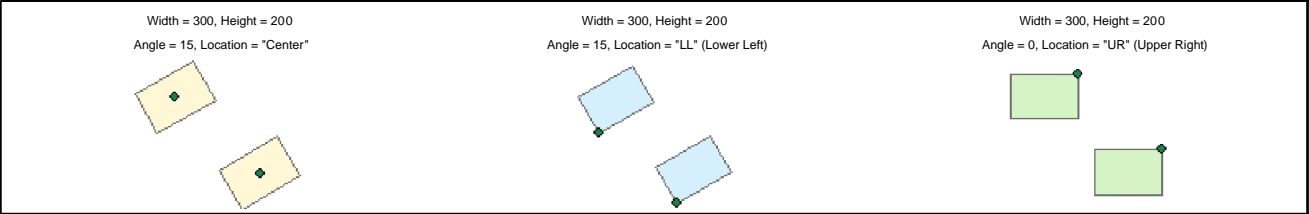
- New polygon feature class. All the original field values will be transferred from the points to the polygons.

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:



ToolBox implementation

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

Command line syntax

ET_GPPointsToRectangles <input_dataset> <out_feature_class> <width_field> <height_field> <angle_field> <LowerLeft | UpperLeft | UpperRight | LowerRight | Center>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<width_field>	A String representing the name of a field in the attribute table of the input dataset field name. The field has the values for the width of the rectangles to be created.
<height_field>	A String representing the name of a field in the attribute table of the input dataset field name. The field has the values for the height of the rectangles to be created.
<angle_field>	A String representing the name of a field 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.
<LowerLeft UpperLeft UpperRight LowerRight Center>	Required. A String - This parameter defines what will be the location of the resulting rectangles in relation to the original points.. The available options are (Case sensitive): <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 input point.● "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.

Scripting syntax

ET_GPPointsToRectangles (input_dataset, out_feature_class,width_field,height_field, angle_field, location_type)

See the explanations above:

<> - required parameter

() - optional parameter

.NET implementation

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

PointsToRectangles(plnFC As IFeatureClass, sOutFName As String, sWidthField As String, sHeightField As String, sAngleField As String, sLocation As String) As IFeatureClass

Reverse Geocoding

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

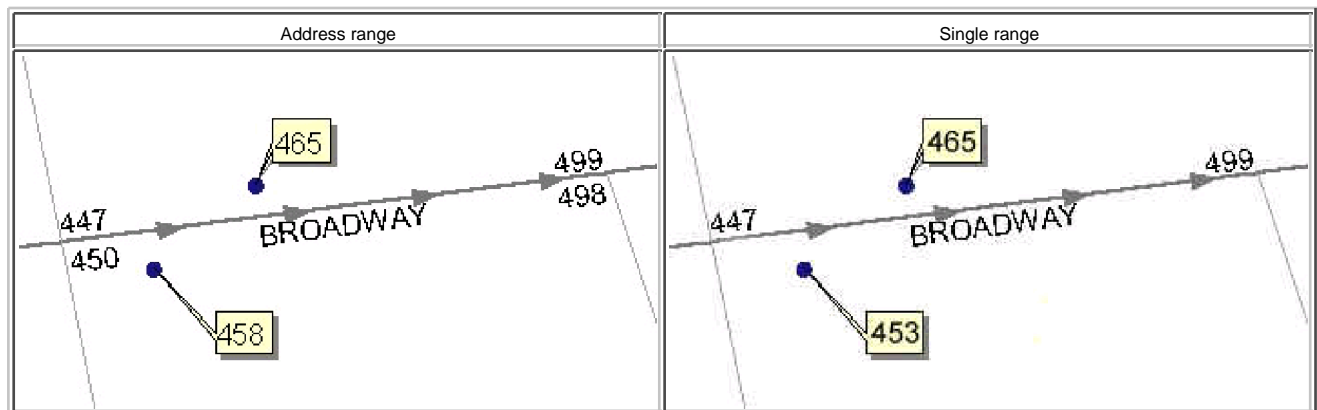
Outputs:

- A point feature class with addresses assigned to the points

Notes:

- Range attributes from string and numeric fields can be handled. The records that have range values containing non numeric characters will not be used.

Example:



Copyright © Ianko Tchoukanski

Create Station Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 feature class 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

ToolBox implementation

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

Command line syntax

ET_GPStationPoints <input_dataset> <out_feature class> <station_distance> {out_spatial_reference}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<station_distance>	A Double representing the step with which the points will be stationed on the input polylines
{out_spatial_reference}	The spatial reference in which the calculations will be performed. If not specified the spatial reference of the input dataset will be used.

Scripting syntax

ET_GPStationPoints (input_dataset, out_feature class, station_distance, out_spatial_reference)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

StationPoints(pInFC As IFeatureClass, sOutFName As String, dStationDistance As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Thin (Generalize) Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 link from the original points to the generalized one can be added.

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.

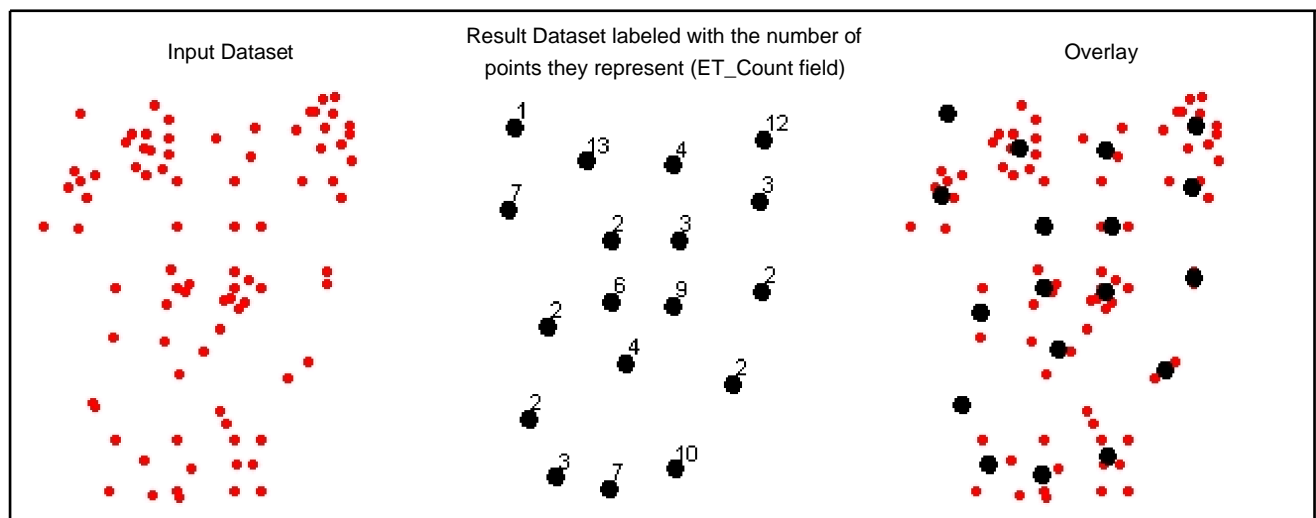
Outputs:

- New point feature class. Fields:
 - [ET_Count] - the number of points from the input feature class represented by each output point
 - Value Field - the sum of the values in the user specified field
 - [ET_Link] - added only if the "Add Link from the original points" option is selected.

Notes:

- If no Generalization tolerance is specified, the tolerance will be set to 0 - only the exact duplicates will be removed

Examples:



ToolBox implementation

[Go to TOP](#)

Command line syntax

ET_GPThinPoints<input_dataset> <out_feature class> <tolerance> {add_link} {value_field}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<tolerance>	A Double representing the Generalization tolerance.

{add_link}	A Boolean. If TRUE, a link field is added to both Original and Generalized points. The values in this field ("ET_Link") can be used to back link the generalized points to the original ones.
{value_field}	A String representing the Data field.

Scripting syntax

ET_GPThinPoints (input_dataset, out_feature_class, tolerance, add_link, value_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ThinPoints(plnFC As IFeatureClass, sOutFName As String, dTolerance As Double, Optional bBackLink As Boolean = False, Optional ByVal sValueField As String = "") As IFeatureClass

Points To Regular Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

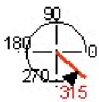
- A Point feature class
- Number of sides of the polygons to be created.
- Size represents option - depending on the user input the size parameter can represent
 - The side of the polygon
 - The radius of the circle inscribed in the polygon
 - The radius of the circle circumscribed around the polygon.
- 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.

Outputs:

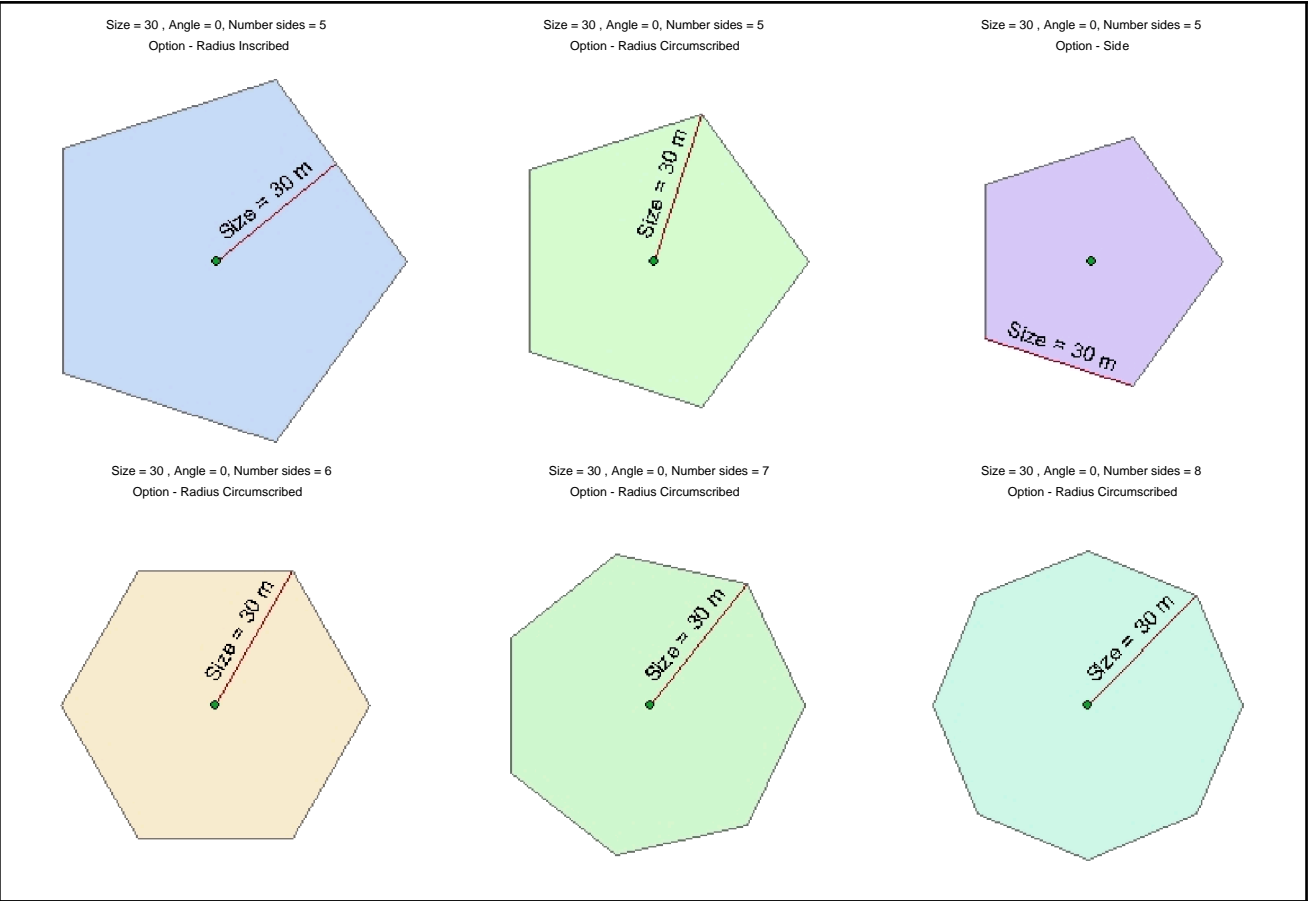
- New polygon feature class. All the original field values will be transferred from the points to the polygons.

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:



ToolBox Implementation

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

Command line syntax

ET_GPPointsToRegularPolygons <input_dataset> <out_feature_class> <number_sides> <Radius In | Radius Out | Side> <Size> <size_field> <rotation_angle>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

<number_sides>	An Integer defining the number of sides of the polygons to be created.
<Radius In Radius Out Side>	Required. A String indicating the meaning of the <SIZE> parameter <ul style="list-style-type: none"> ● Radius In - radius of the inscribed circle ● Radius Out - radius of the circumscribed circle ● Side - the side of the polygon
<Size>	A Double representing the size (see above for options)
<size_field>	A String representing the name of a field in the attribute table of the input dataset. The field has the values for the size of the polygons to be created.
{rotation_angle}	A Double representing the rotation angle (see above)

Scripting syntax

ET_GPPointsToRegularPolygons (input_dataset, out_feature_class,number_sides,size_represents, size, size_field, rotation_angle)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointsToRegularPolygons(plnFC As IFeatureClass, sOutFName As String, iNumberSides As Integer, sSizeRepresents As String, Optional dSize As Double = 0, Optional sSizeField As String = "", Optional dRotationAngle As Double = 0) As IFeatureClass

Buffer Polylines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

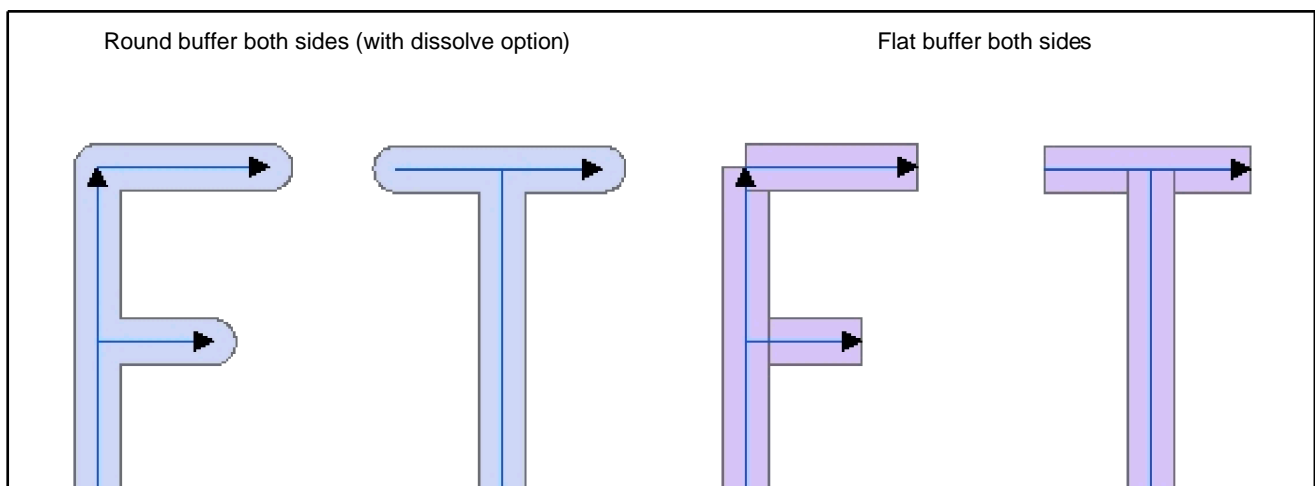
Outputs:

- 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.

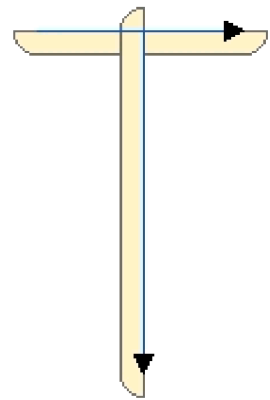
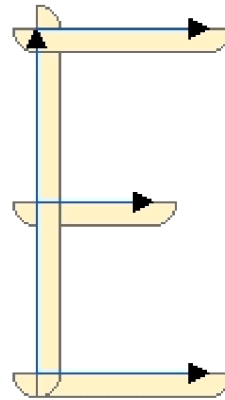
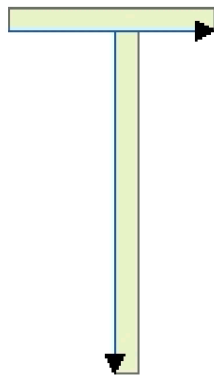
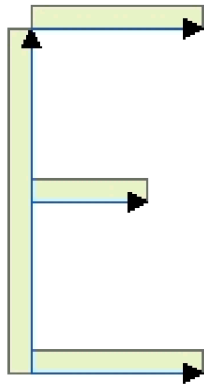
Examples:





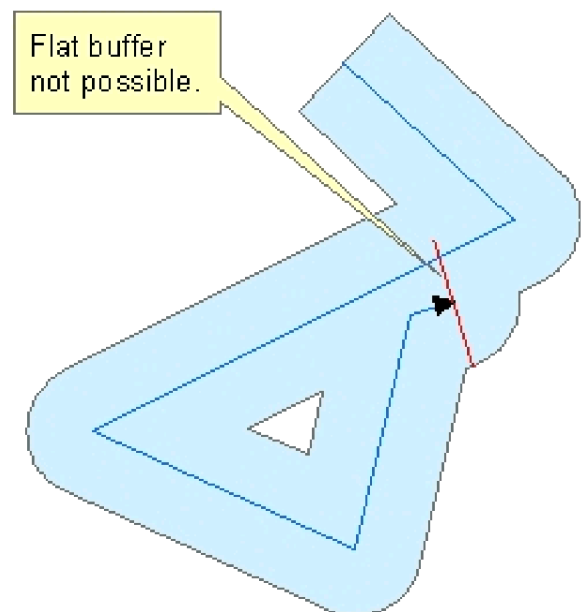
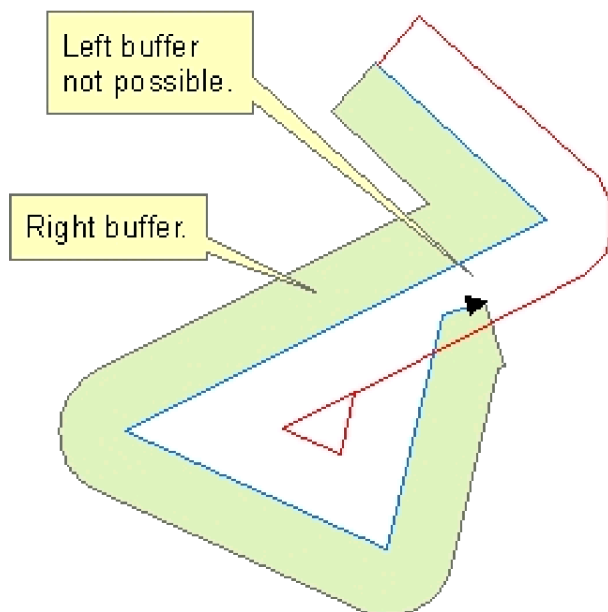
Flat buffer on the left side

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.



ToolBox implementation

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

Command line syntax

ET_GPBufferPolylines<input_dataset> <out_feature class> {buffer_distance_field} {buffer_distance} <Both | Left | Right>
<Round | Flat> <dissolve_buffers>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer

<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{buffer_distance_field}	A String representing the name of a field in the attribute table of the input dataset field name. The field has the values for the buffer distance to be used.
{buffer_distance}	A Double representing the buffer distance (in the units of the spatial reference of the input dataset).
<buffer_side>	A String representing the side of the polylines on which the buffers will be created. The available options are: Left, Right and Both
<shape_end>	Required. A String representing the shape of the buffer at the ends of the polylines. The available options are: <ul style="list-style-type: none"> ● 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_buffers>	A Boolean. If True - the boundaries of the intersecting buffers will be dissolved.

Scripting syntax

ET_GPBufferPolylines(input_dataset, out_feature class,buffer_distance_field,buffer_distance, buffer_side, shape_end, dissolve_buffers)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

BufferPolylines(pInFC As IFeatureClass, sOutFName As String, sSideOption As String, sEndOption As String, Optional sBufferDistanceField As String = "", Optional dBufferDistance As Double = 0, Optional bDissolveBuffers As Boolean = False) As IFeatureClass

Clean Contour Gaps

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

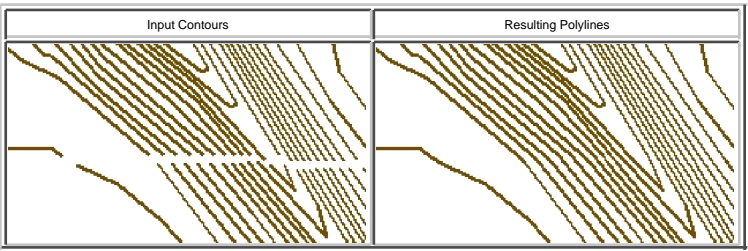
Outputs:

- New polyline feature class. The gaps in the contours that a smaller then 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 :



ToolBox implementation

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

Command line syntax

ET_GPCleanContourGaps <input_dataset> <out_feature_class> <gap_size> <height_field>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
<height_field>	A String - the name of the field having the elevation value of the contours.
<gap_size>	A Double - the gaps smaller than this tolerance will be closed. The units of the parameter are in the spatial reference of the input feature class

Scripting syntax

ET_GPCleanContourGaps (input_dataset, out_feature_class, gap_size, height_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanContourGaps(plnFC As IFeatureClass, sOutFName As String, ByVal dMaxGapToFill As Double, ByVal sElevationField As String) As IFeatureClass

Clean Polyline

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Ensures topological correctness of a polyline feature data set.

Inputs:

- A polyline feature layer
- Fuzzy tolerance

Outputs:

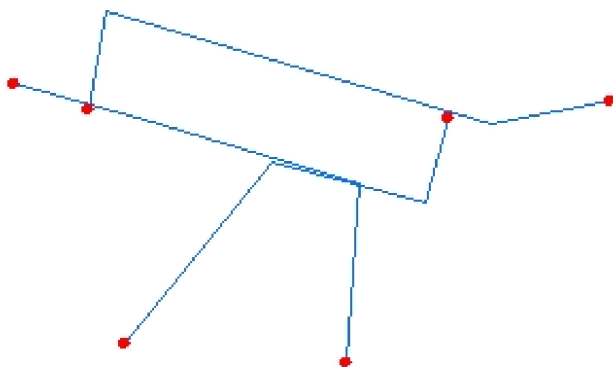
- New topologically correct Polyline feature class
 - Nodes will be created in all intersection points
 - Redundant data (vertices and nodes closer to each other than the fuzzy tolerance) will be eliminated.
 - Each set of duplicate lines (closer to each other than the fuzzy tolerance) will be replaced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

Notes :

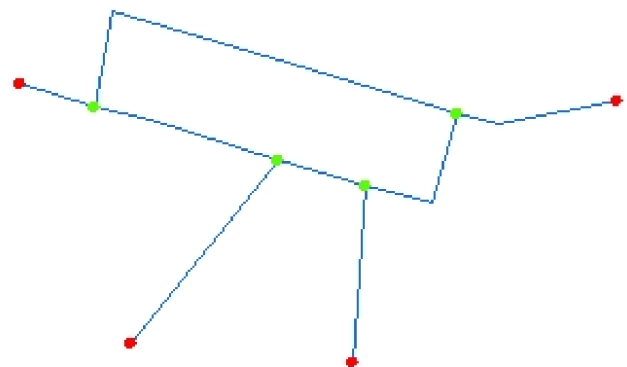
- The default Fuzzy tolerance is calculated from the extents of the input layer using the formulae $(W + H) / 2000000$ where W and H are the width and height of the extent envelope.
- Larger values of the Fuzzy tolerance may be used to clean some bigger Over and Under shoots, but it might lead to unwanted 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"
- 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
- 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.

Example:

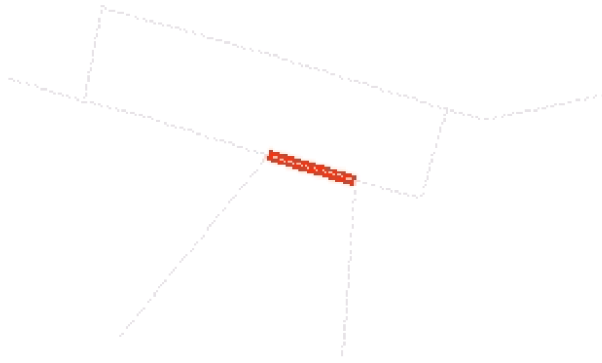
Input Layer (with analyzed nodes)



After Clean (with analyzed nodes)



Duplicates layer (The Clean layer as a background)



ToolBox implementation

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

Command line syntax

ET_GPCleanPolyline <input_dataset> <out_feature class> <fuzzy_tolerance>{duplicates_feature_class}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{fuzzy_tolerance}	A Double setting the Fuzzy tolerance (in the units of the input dataset) to be used if the {clean_polylines} is True. If {clean_polylines} is False this parameter is ignored
{duplicates_feature_class}	A String - the full name of the output polyline feature class that identifies the polylines removed as duplicates. (A feature class with the same full name should not exist)

Scripting syntax

ET_GPCleanPolyline (input_dataset, out_feature class, fuzzy_tolerance,duplicates_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanPolylines(pInFC As IFeatureClass, sOutFName As String, dFuzzy As Double, Optional sDuplicatesFName As String = "")
As IFeatureClass

Clean Dangling Nodes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

- A polyline feature layer
- Dangling tolerance

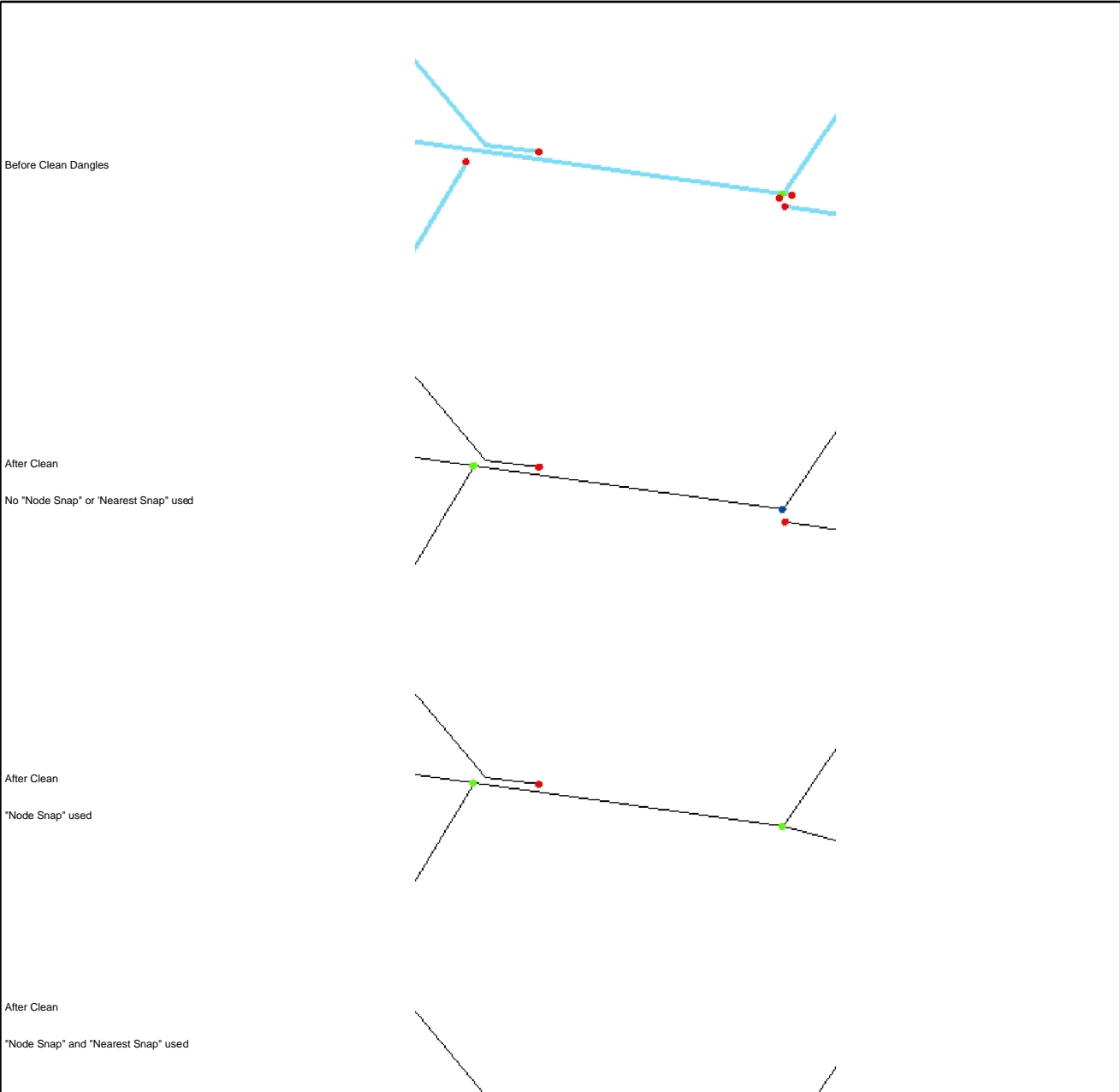
Outputs:

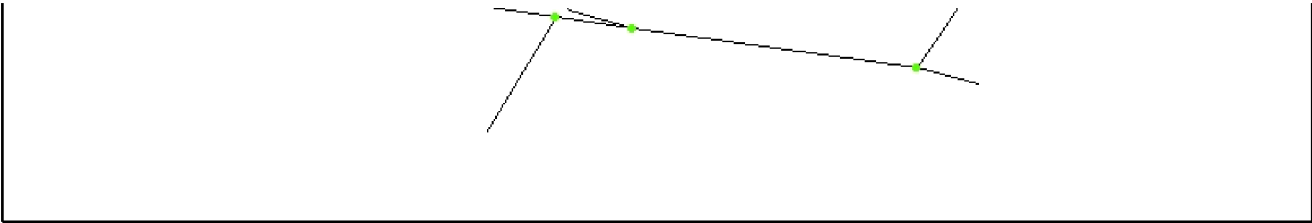
- New polyline feature class
 - Over-shoots: All the Polylines having a Dangling node with length less than the dangling tolerance will be deleted
 - Under-shoots: All the Polylines having a Dangling node and length larger than the tolerance will be processed.
 - The function first checks whether the 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 "Use closest node snap" option is selected the function checks whether the dangling node is closer than the tolerance specified to another node and if so, snaps it to that node.
 - If the "Use nearest feature snap" 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:





ToolBox implementation

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

Command line syntax

ET_GPCleanDangle <input_dataset> <out_feature class> <dangling_tolerance> {closest_node_snap} {nearest_feature_snap}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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
{closest_node_snap}	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 will be snapped (if the dangling node is within a distance smaller than the <dangling_tolerance> to an existing node) to the closest existing node.
{nearest_feature_snap}	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 <dangling_tolerance> to an existing polyline) to the closest point of the closest polyline

Scripting syntax

ET_GPCleanDangle (input_dataset, out_feature class, dangling_tolerance, closest_node_snap, nearest_feature_snap)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanDangles(plnFC As IFeatureClass, sOutFName As String, dDanglingTol As Double, Optional bClosestNode As Boolean = False, Optional bNearest As Boolean = False, Optional dFuzzy As Double = 0) As IFeatureClass

Clean Pseudo Nodes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 function works like the UNSPLIT command of ArcEdit. 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

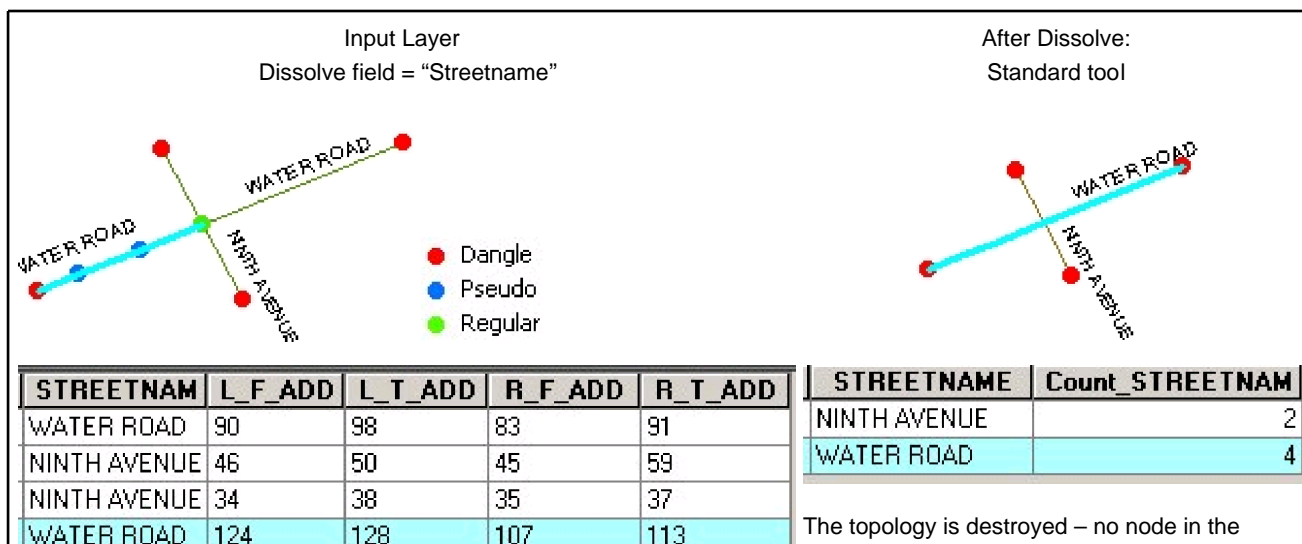
Outputs:

- 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 Wizard has several advantages over the dissolve function of GeoProcessing Wizard:
 - Multiple dissolve fields can be used
 - Works like the UNSPLIT command of ArcEdit – 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
- If multi-part features are already present in the layer to be dissolved, the Clean Pseudo Nodes Wizard will explode them first. The attributes will be distributed as follows:
 - For the fields that the user has selected SUM as an update rule, the values will be proportionally distributed between the parts
 - For the rest of the fields the attributes will be copied over.
- It is recommended the Explode Wizard 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 string and numeric fields can be handled. The records that have range values containing non numeric characters will be copied arbitrary to the resulting features.

Example:



WATER ROAD	110	118	93	101
WATER ROAD	118	124	101	107

intersection point of the two streets

After Clean Pseudo Nodes:

ET GeoWizards

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

Preserved topology. Address ranges updated correctly

ToolBox implementation

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

Command line syntax

ET_GPCleanPseudo <input_dataset> <out_feature class> <dissolve_field_list> {Update_Rules_List}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{dissolve_field_list}	A String - a list of field names to be used for dissolving.
{Update_Rules_List}	A String - a list of fields with their update rules.

Scripting syntax

ET_GPCleanPseudo (input_dataset, out_feature class, "dissolve_field; dissolve_field", "field update_rule; field update_rule; field update_rule")

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\streets.shp"
result = "C:\\data\\fgdb_test.gdb\\dissolved"
arcpy.gp.ET_GPCleanPseudo (input_dataset, result, "Name;Type", "Meters Sum; Suburb First; L_F_ADD Range Address L_T_ADD ;L_T_ADD Range Address L_F_ADD")
```

.NET implementation

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

CleanPseudoNodes(plnFC As IFeatureClass, sOutFName As String, dissolveFields As List(Of String), Optional updateRules
As Dictionary(Of String, String) = Nothing) As IFeatureClass

Copyright © Ianko Tchoukanski

Densify

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

- A polyline feature layer
- Curves simplification method
 - Using deviation distance
 - Using deviation angle
- Maximum segment length
- Deviation value (Have no impact if there are no curve segments in the polylines - in most of the cases can be set to 0)

Outputs:

- New polyline feature class
 - The output feature class will contain all the features of the original data set
 - If the "Densify selected" option is used, only selected polylines will be densified, the rest will preserve their original shape.
 - The attributes of the input data set are preserved.

Notes :

- The Deviation value parameter has no impact if there are no curve segments in the polylines - in most of the cases can be set to 0 (default)
- The function uses the standard ArcObjects methods which are fast and efficient.
- For more information see Densify and DensifyByAngle methods in ArcObjects Developer Help.

ToolBox implementation

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

Command line syntax

ET_GPDensify <input_dataset> <out_feature class> <max_segment_length> <deviation_value>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<max_segment_length>	A Double representing the Maximum segment length
<deviation_value>	A Double representing the Deviation value (Has no impact if there are no curve segments in the polylines - in most of the cases can be set to 0)

Scripting syntax

ET_GPDensify (input_dataset, out_feature class, max_segment_length, deviation_value)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

Densify(pInFC As IFeatureClass, sOutFName As String, dMaxSegmentLength As Double, Optional dDeviation As Double = 0)
As IFeatureClass

Copyright © Ianko Tchoukanski

Export Nodes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

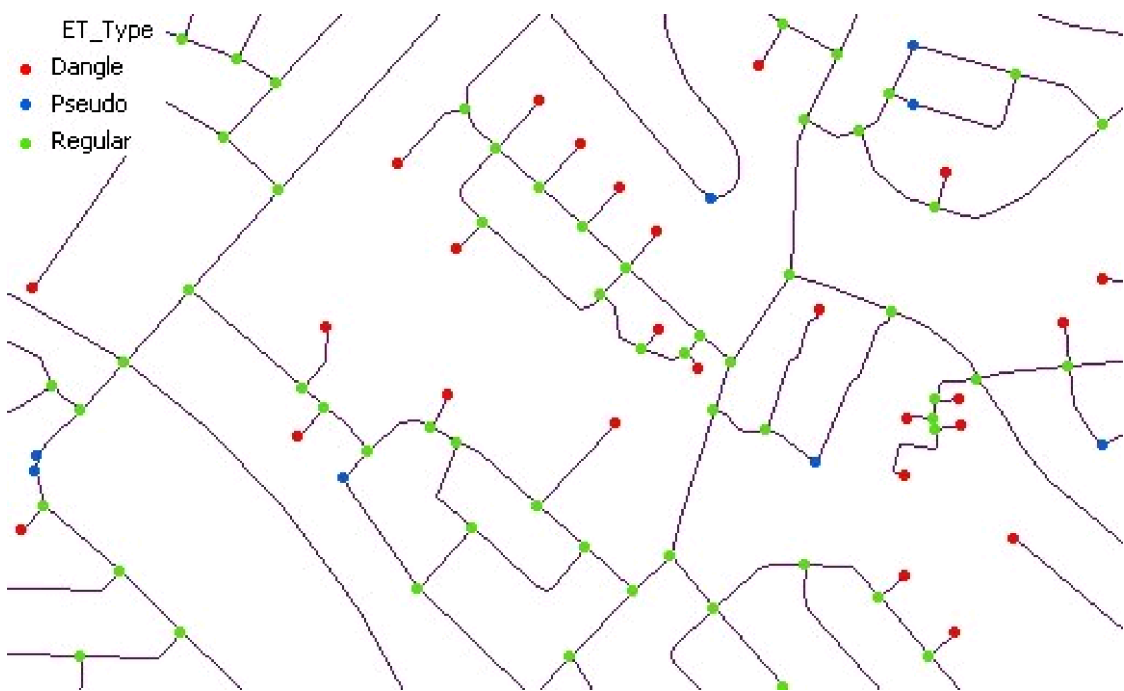
Analyzes the nodes of a polyline layer and exports them as a point feature class.

Inputs:

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

Outputs:

- New point feature class
 - 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 a 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_FID1], [PL_FID2] ...[PL_FIDn] carrying the IDs of the polylines that intersect in the node



ToolBox implementation

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

Command line syntax

ET_GPExportNodes <input_dataset> <out_feature class> <link_field> {export_dangling} {export_pseudo} {export_regular}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer

<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<link_field>	A String representing the name of the field that will be attached to the point attribute table. If for example the function is used to export the nodes of a street dataset and the "StreetName" is the field name to be used for each point in the point attribute table will be recorded all the streets that intersect in this node.
{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

Scripting syntax

ET_GPExportNodes (input_dataset, out_feature class, link_field, export_dangling, export_pseudo, export_regular)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ExportNodes(pInFC As IFeatureClass, sOutFName As String, sLinkField As String, Optional bDangling As Boolean = True, Optional bPseudo As Boolean = True, Optional bRegular As Boolean = True) As IFeatureClass

Flip Polylines

Changes the direction of the polylines from a polyline layer based on user defined start point.

Inputs:

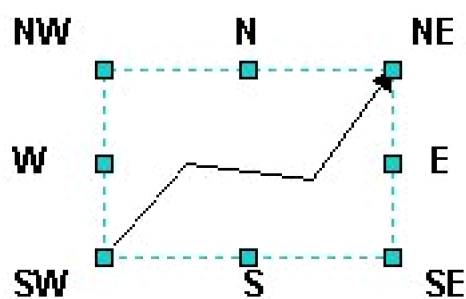
- A polyline feature layer
- Start Point to be used

Outputs:

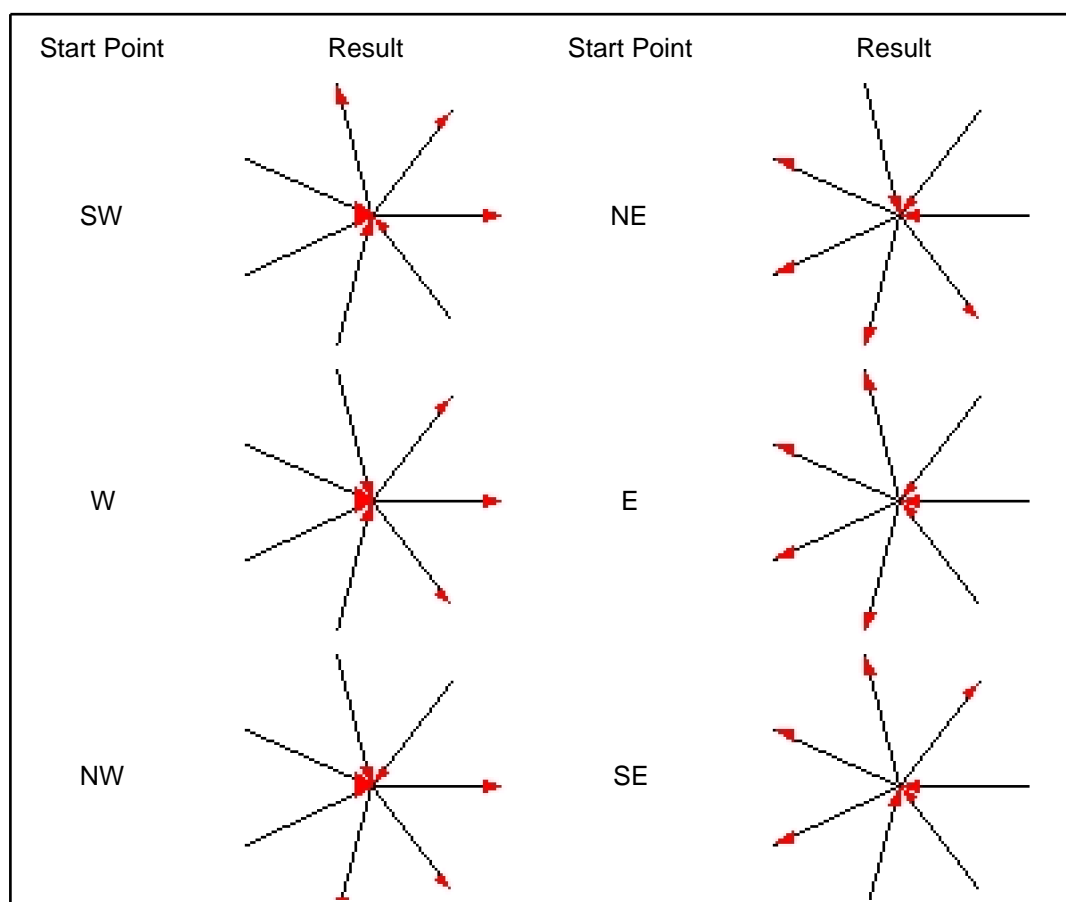
- New Polyline feature class - all polylines have their from node closer to the user specified start points

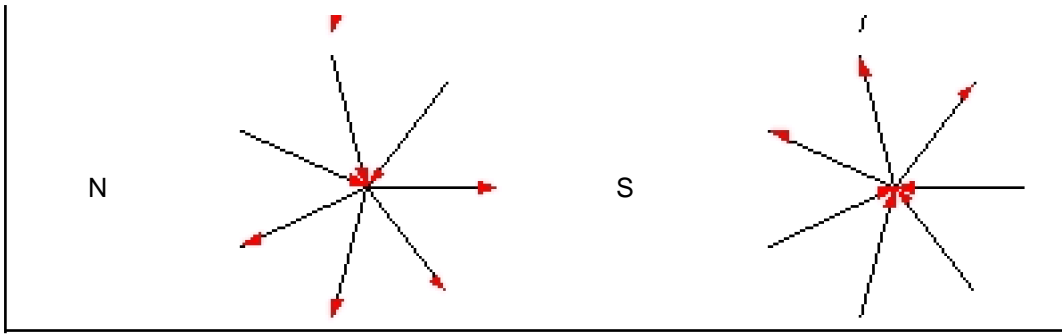
Notes :

- The corners and the middle points of the smallest bounding rectangle of each polyline are used



Examples:





Copyright © Ianko Tchoukanski

Generalize

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
 - If the "Generalize selected" option is used, only selected polylines will be generalized, the rest will preserve their original shape.
 - The attributes of the input data set are preserved.

Notes :

- The function uses the standard ArcObjects method which is fast and efficient.
- For more information see Generalize method in ArcObjects Developer Help.

ToolBox implementation

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

Command line syntax

ET_GPGeneralize <input_dataset> <out_feature_class> <generalize_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<generalize_tolerance>	A Double representing Generalize tolerance (maximum offset) - the maximum distance that the generalized polyline will differ from the original one

Scripting syntax

ET_GPGeneralize (input_dataset, out_feature_class, generalize_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

GeneralizePolylines(pInFC As IFeatureClass, sOutFName As String, dGenTol As Double) As IFeatureClass

Polyline Global Snap

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Snaps the features of a polyline layer to another layer (Point, 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 feature class - the polylines from the source layer will be moved to snap to the features of the Snap Layer (if within the snap tolerance)





Options:

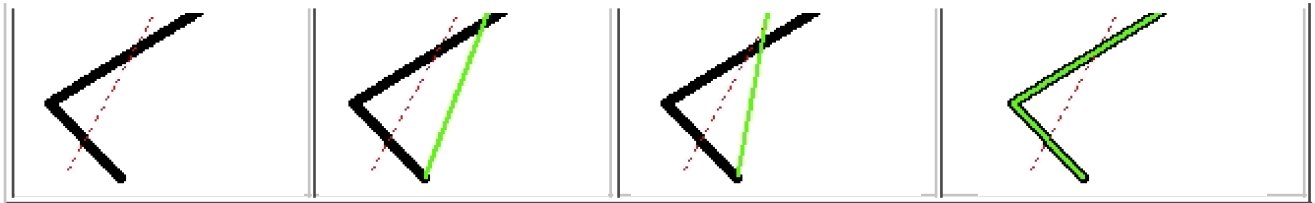
- Snap Options 1 (Snap What) - this options lets the user set which elements of the source polylines to be used for snapping
 - Nodes: Only nodes (end points) of the polylines will be snapped
 - Vertices: All the vertices of the source polylines will be used.
 - Insert Vertices: This option will get 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 & 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 & Edges
			



ToolBox implementation

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

Command line syntax

ET_GPSnapPolylines <input_dataset> <Reference_dataset> <out_feature class> <snap_tolerance> <snap_what> {snap_to_vertices}
{snap_to_nearest}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<Reference_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<snap_tolerance>	A Double representing the Search tolerance (in the units of the input_dataset) to be used
<snap_what>	A String indicating what parts of the input polylines will be snapped. Possible values: <ul style="list-style-type: none"> ● Nodes: Only nodes (end points) of the polylines will be snapped ● Vertex: All the vertices of the source polylines will be used. ● Insert: This option will get the vertices from the features of the Reference_dataset 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 Reference_dataset.
{snap_to_vertices}	A Boolean indicating whether snapping to the closest vertex of the nearest feature from the Reference_dataset to be used
{snap_to_nearest}	A Boolean indicating whether snapping to the nearest point of the nearest feature from the Reference_dataset to be used

Scripting syntax

ET_GPSnapPolylines (input_dataset, Reference_dataset, out_feature_class, snap_tolerance, snap_what,snap_to_vertices, snap_to_nearest)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SnapPolylines(pInFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String, dSnapTol As Double, sSnapWhat As String, Optional bVertex As Boolean = False, Optional bNearest As Boolean = False, Optional bSnapToZ As Boolean = False) As IFeatureClass

PolylineZ Characteristics

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Calculates several characteristics of PolylineZs. The results are stored in fields in the polyline attribute table. The results are stored in the attribute table of the input layer or in a new feature class

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 input dataset or in a new feature class. 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

ToolBox implementation

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

Command line syntax

ET_GPGetZChars<input_dataset> <out_feature class> {linear_precision} {angular_precision} {nodata}

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{linear_precision}	A Double - the number of digits after the decimal point for linear measures.
{angular_precision}	A Double - the number of digits after the decimal point for angular measures.
{nodata}	A Double - represents undefined Z values.

Scripting syntax

ET_GPGetZChars(input_dataset, out_feature class,linear_precision,angular_precision, nodata)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

GetZCharacteristics(plnFC As IFeatureClass, sOutFName As String,Optional iLinearPrecision As Short = 2, Optional iAngularPrecision As Short = 2, Optional dNodata As Double = 999999999) As IFeatureClass

Copyright © Ianko Tchoukanski

Renode

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
- Two fields are added to the original 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

ToolBox implementation

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

Command line syntax

ET_GPRenode <input_dataset> <nodes_dataset>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<nodes_feature_class>	A String - the full name of the output point feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPRenode (input_dataset, nodes_dataset)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

RenodePolylines(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Smooth Polyline

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

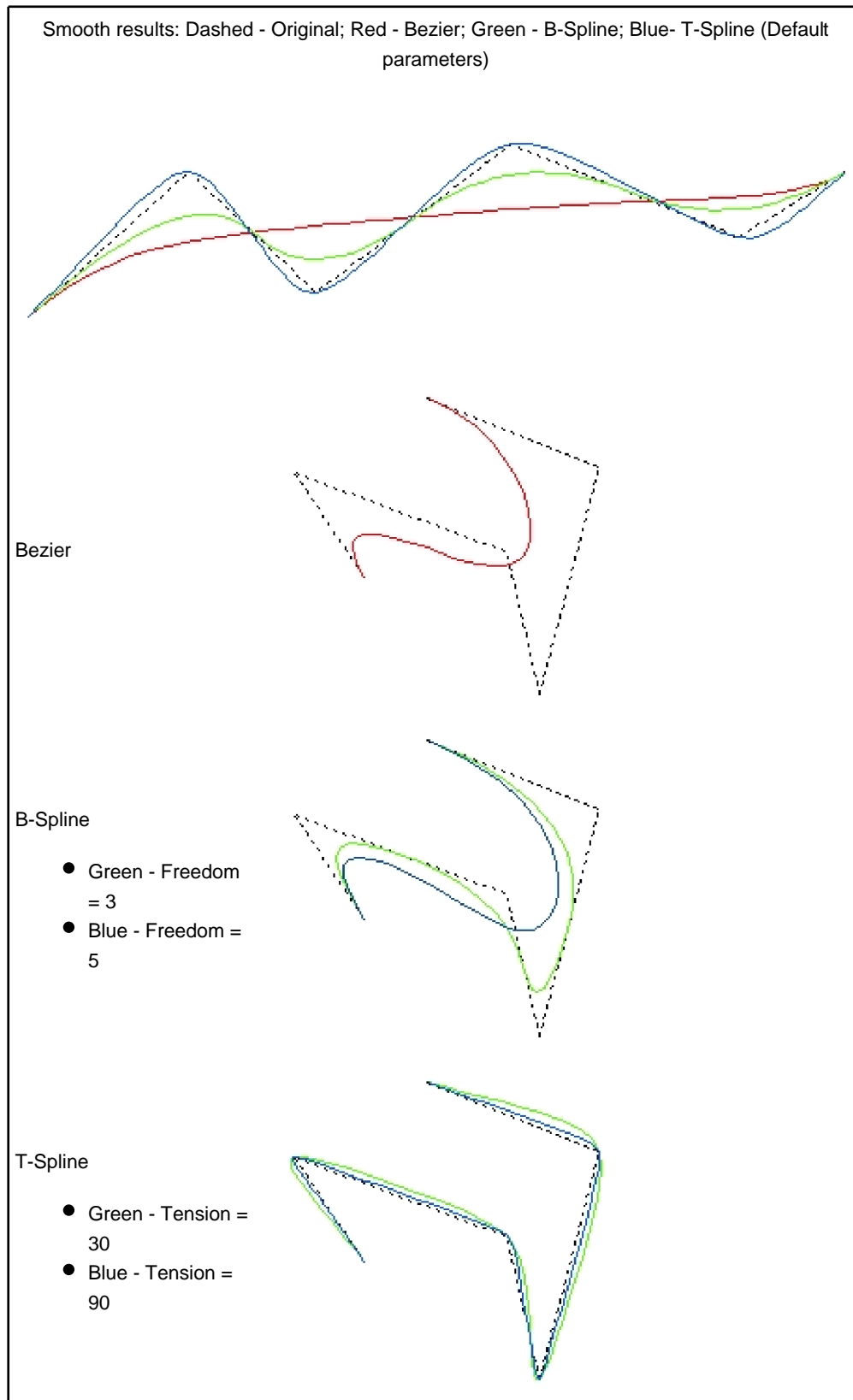
- New polyline feature class
 - The output feature class will contain all the features of the original data set
 - If the "Smooth selected" option is used, only selected polylines will be smoothed, the rest will preserve their original shape.
 - The attributes of the input data set are preserved.

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 feature class

Examples:



ToolBox implementation

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

Command line syntax - ET_GPSSmoothBezier

ET_GPSSmoothBezier <input_dataset> <out_feature class> <smoothness>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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 <smoothness> parameter, the slower the process will be.

Scripting syntax - ET_GPSSmoothBezier

ET_GPSSmoothBezier (input_dataset, out_feature class, smoothness)

Command line syntax - ET_GPSSmoothBSpline

ET_GPSSmoothBSpline <input_dataset> <out_feature class> <smoothness> <freedom>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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 <smoothness> parameter, the slower the process will be.
<freedom>	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

Scripting syntax - ET_GPSSmoothBSpline

ET_GPSSmoothBSpline (input_dataset, out_feature class, smoothness, freedom)

Command line syntax - ET_GPSSmoothBSpline

ET_GPSSmoothTSpline <input_dataset> <out_feature class> <smoothness> <tension>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

- <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 <smoothness> parameter, the slower the process will be.
- <tension> 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.

Scripting syntax - ET_GPSmoothBSpline

ET_GPSmoothTSpline (input_dataset, out_feature class, smoothness, tension)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SmoothBSpline(pInFC As IFeatureClass, sOutFName As String, iSmoothness As Short, iFreedom As Short, sPolylinesType As String) As IFeatureClass

SmoothTSpline(pInFC As IFeatureClass, sOutFName As String, iSmoothness As Short, iFreedom As Short, sPolylinesType As String) As IFeatureClass

SmoothBezier(pInFC As IFeatureClass, sOutFName As String, iSmoothness As Short, sPolylinesType As String) As IFeatureClass

Split Polyline Wizard

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
 - Segments length - two options available
 - 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 update rules for each field

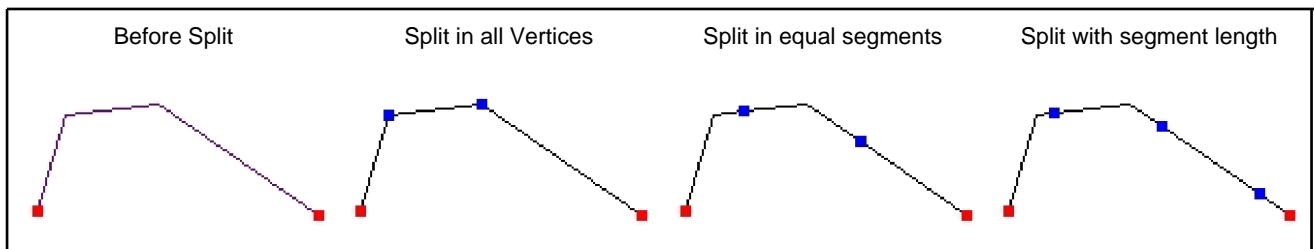
Outputs:

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

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 the "Split selected features only" option is selected only the selected features from the input layer will be split.
- 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:



ToolBox implementation

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

Command line syntax

```
ET_GPSplitPolylines<input_dataset> <out_feature_class> <NumberIntervals | NumberVertices | SegmentLength |  
EqualLength | Vertex> <split_tolerance>
```

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<NumberIntervals NumberVertices SegmentLength EqualLength Vertex>	A String - the split option to be used.
<split_tolerance>	Required. A Double representing depending on the option selected: <ul style="list-style-type: none"> • "Vertex" - not used. • "NumberIntervals" - the number of segments • "SegmentLength" - the length of the segments • "EqualLength" - the length of the segments • "NumverVertices"- the number of vertices per segment
{Update_Rules_List}	A String - a list of fields with their update rules.

Scripting syntax

ET_GPSplitPolylines(input_dataset, out_feature_class, split_option, split_tolerance, Update_Rules_List)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\streets.shp"
result = "C:\\data\\fgdb_test.gdb\\split"
arcpy.gp.ET_GPSplitPolylines (input_dataset, result, "SegmentLength",50, "Meters Proportion; Suburb Copy; L_F_ADD
Range Address L_T_ADD ;L_T_ADD Range Address L_F_ADD")
```

.NET implementation

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

SplitPolylines(pInFC As IFeatureClass, sOutFName As String, sSplitOptions As String, Optional dSplitTol As Double = 0, Optional updateRules As Dictionary(Of String, String) = Nothing) As IFeatureClass

Split Polyline With Layer

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

- The two layers should have the same Spatial Reference
- 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.

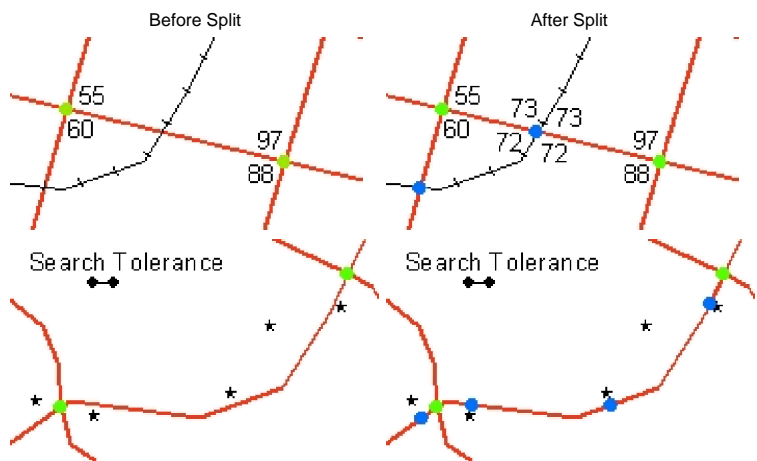
Examples:

Polyline Layer with Polyline Layer.

Attributes updated with Range Address split rule

Polyline Layer with Point Layer

Only the points within the search tolerance from the polylines are used.



ToolBox implementation

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

Command line syntax

ET_GPSplitPolylinesWithFeatureClass <input_dataset> <Split_dataset> <out_feature_class> {search_tolerance}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<Split_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{search_tolerance}	A Double representing the Search tolerance (in the units of the input dataset) to be used. Ignored if the split feature class is of Polyline or Polygon type
{Update_Rules_List}	A String - a list of fields with their update rules.

Scripting syntax

ET_GPSplitPolylinesWithFeatureClass (input_dataset, Split_dataset, out_feature_class, search_tolerance, Update_Rules_List)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\streets.shp"
split_dataset = "C:\\data\\fgdb_test.gdb\\pg2"
result = "C:\\data\\fgdb_test.gdb\\split"
arcpy.gp.ET_GPSplitPolylinesWithFeatureClass (input_dataset, split_dataset , result, 50, "Meters Proportion; Suburb Copy; L_F_ADD Range Address L_T_ADD ;L_T_ADD Range Address L_F_ADD")
```

.NET implementation

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

SplitPolylinesWithFeatureClass(plnFC As IFeatureClass, pSplitFC As IFeatureClass,sOutFName As String, Optional dSearchTol As Double = 0, Optional updateRules As Dictionary(Of String, String) = Nothing) As IFeatureClass

Copyright © Ianko Tchoukanski

Polyline Characteristics

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Calculates some characteristics of the polylines from a polyline dataset

Inputs:

- A Polyline feature class

Outputs:

- The results can be added to the input feature class or a new polygon feature class. All attributes of the original features are preserved
- New fields added to the attribute table
 - ET_Sinous - 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

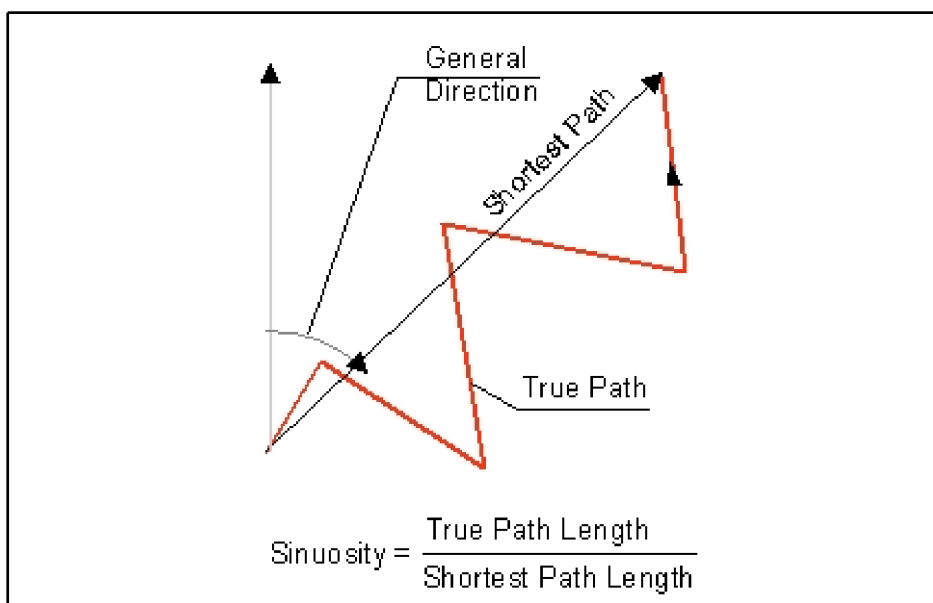


- ET_Parts - the number of parts that the polyline has
- ET_HasArcs - if the polyline has true arc segments - 1 otherwise - 0
- ET_Closed - if the polyline is closed - 1 otherwise - 0
- ET_Fract - the fractal dimension (indication of the complexity) of the polyline. The value is between 1 and 2. The 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:



References:

1. Bourke, P., 1993. Fractal Dimension Calculator User Manual, Online. Available: <http://paulbourke.net/fractals/fracdim/>

ToolBox implementation

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

Command line syntax

ET_GPPolylineCharacteristics <input_dataset> {fractal_dimension} {precision}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer.
{fractal_dimension}	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.

Scripting syntax

ET_GPPolylineCharacteristics (input_dataset, fractal_dimension, precision)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolylineCharacteristics(pInFC As IFeatureClass, Optional bCalculateFract As Boolean = False, Optional iPrecision As Short = 2) As Boolean

Flip PolylineZ

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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)

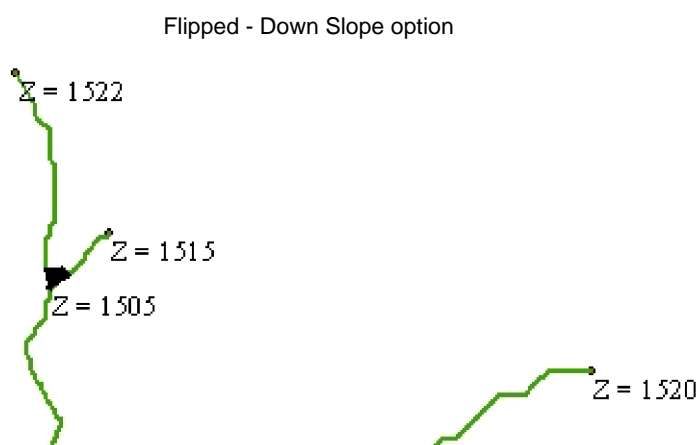
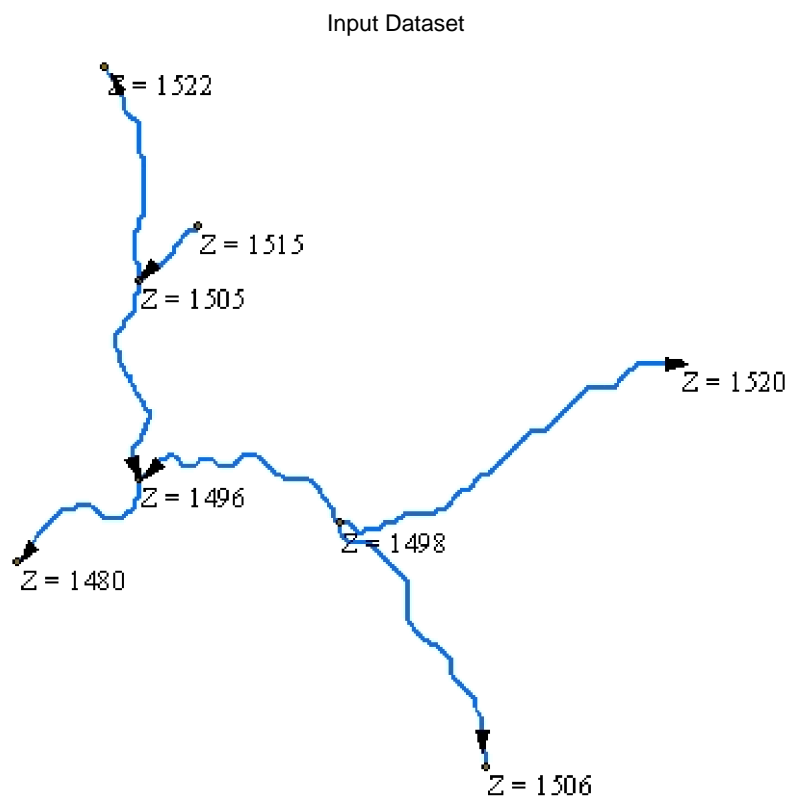
Inputs:

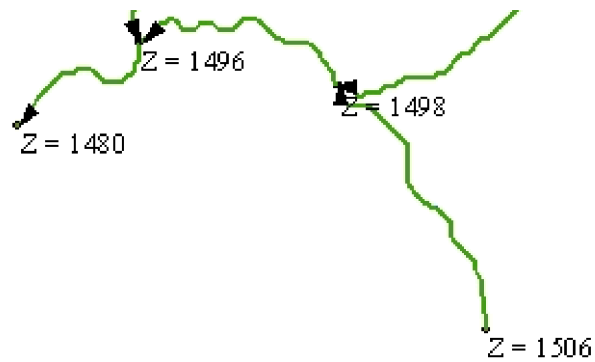
- A PolylineZ feature layer
- Flip Option

Outputs:

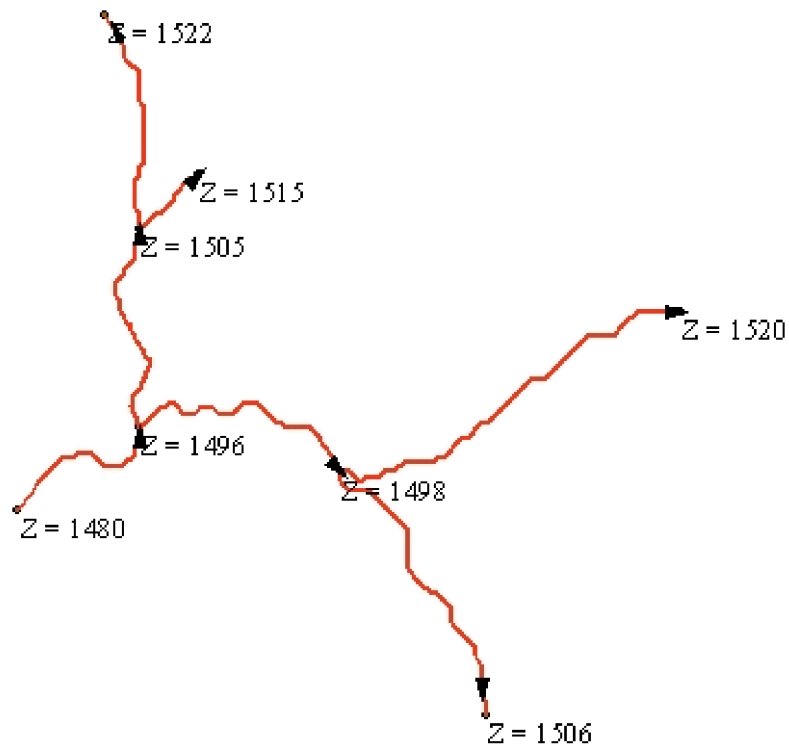
- A PolylineZ feature class.
 - A field called ET_Status will be added to the attribute table. The values in this field will indicate whether a polyline has been flipped ("Flipped") or not ("Original")

Example:





Flipped - Up Slope option



ToolBox implementation

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

Command line syntax

ET_GPFlipPolylineZ <input_dataset> <out_feature class> <direction>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<direction>	A String indicating the direction of the output PolylineZs - Allowed values - "Down Slope" and "Up Slope"

Scripting syntax

ET_GPFlipPolylineZ (input_dataset, out_feature class, direction)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FlipPolylineZ(pInFC As IFeatureClass, sOutFName As String, sDirection As String) As IFeatureClass

Copyright © Ianko Tchoukanski

Advanced Merge

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Merges two polygon data sets. The result does not contain overlaps

Inputs:

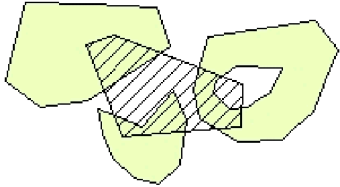
- Base layer - a polygon layer that will keep all attribute fields
- Merge layer - a polygon layer that will be merged to the Base layer.
- Priority of the Merge layer as described below
 - Priority - **"Erase"** - the polygons from the Base layer are erased with the polygons of the Merge layer
 - Priority - **"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:

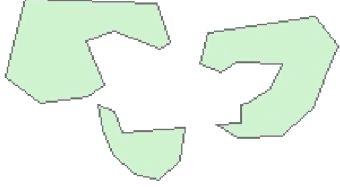
- New Polygon feature class
 - No overlaps present. The polygons from the Base and the Merge layers are overlaid depending the priority of the Merge layer
 - All the attributes of the Base layer are preserved
 - Only the attributes in fields with the same name and type as these from the Base layer are 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



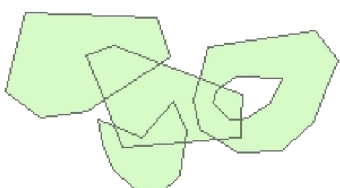
Result: Priority of "-1" (Erase)




Result: Priority of "0" (Low)



Result: Priority of "1" (Standard)



Result: Priority of "2" (High)



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 table

Id	a	b
2	bbb	eee
3	ccc	fff
1	aaa	ddd

Result table

Id	a	b
1	aaa	ddd
2	bbb	eee
3	ccc	fff
1	xxx	
1	xxx	

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 table

Id	a	b
2	bbb	eee
3	ccc	fff
1	aaa	ddd
1	xxx	



ToolBox implementation

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

Command line syntax

ET_GPAAdvancedMerge <input_dataset> <merge_dataset> <out_feature class> <merge_priority> {fuzzy_tolerance}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<merge_dataset>	A Polygon feature class or feature class. NOTE: The spatial references of <merge_dataset> and the <input_dataset> must have the same Geographic Coordinate System
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<merge_priority>	Required. A string representing the priority of the polygons to be merged <ul style="list-style-type: none">● Priority of Erase - the polygons from the Base dataset are erased with the polygons of the Merge dataset● Priority of 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● Priority of 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● Priority of 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.
{fuzzy_tolerance}	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPAAdvancedMerge (input_dataset, merge_dataset, out_feature class, merge_priority, fuzzy_tolerance)

Example Python script

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
input_dataset = "C:\\data\\pg1.shp"
merge_dataset = "C:\\data\\fgdb_test.gdb\\pg2"
result = "C:\\data\\fgdb_test.gdb\\erased"
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx";
arcpy.ET_GPAAdvancedMerge(input_dataset, merge_dase, result, "Erase", "0.003426878")
```

.NET implementation

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

AdvancedMerge(plnFC As IFeatureClass, pMergeFc As IFeatureClass, sOutFName As String, sMergePriority As String, dFuzzy As Double) As IFeatureClass

Aggregate Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Combines the polygons that are within the user specified distance into new polygons. Can be used also to generalize buildings.

Inputs:

- A polygon feature class
- 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:

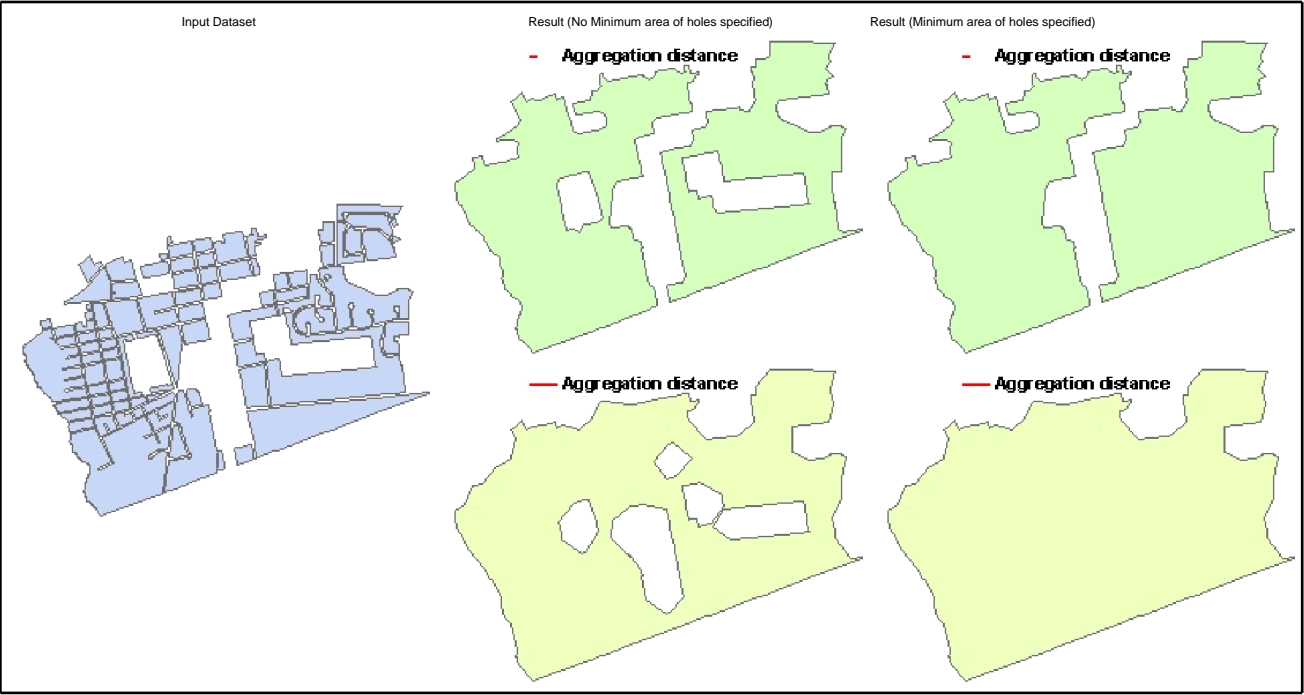
- New polygon feature class

Notes:

- The new feature class 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 feature class
- 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
- The function is CPU and RAM intensive. Using it on large datasets might need long processing time.

Examples:

- General polygons



- Buildings



ToolBox implementation

[Go to TOP](#)

Command line syntax

ET_GPAggregatePolygons <input_dataset> <out_feature_class> <Aggregate_tolerance> <area_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Aggregate_tolerance>	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.

Scripting syntax

ET_GPAggregatePolygons (input_dataset, out_feature_class, Aggregate_tolerance,area_tolerance)

See the explanations above:

<> - required parameter

() - optional parameter

Example Python script

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\aggregated"
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
arcpy.ET_GPAggregatePolygons(input_dataset, result, 10.00, 1000.00)
```

.NET implementation

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

AggregatePolygons(plnFC As IFeatureClass, sOutFName As String, dAggregateTol As Double, Optional dAreaTol As Double = 0) As IFeatureClass

Build Polygon

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Builds polygon feature class 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.

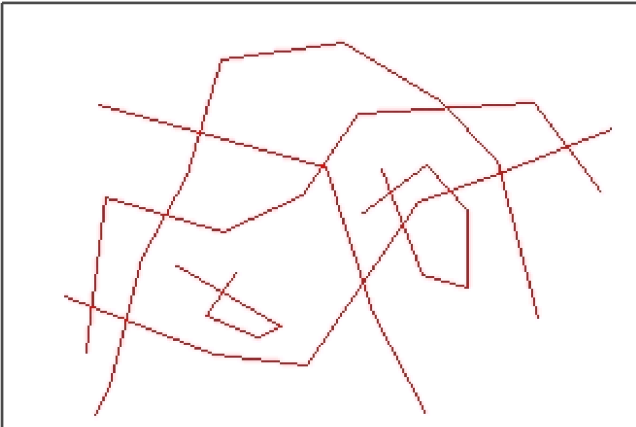
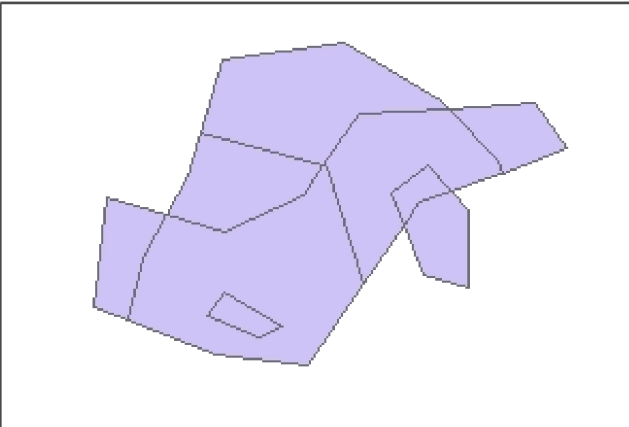
Outputs:

- 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 Wizard.
 - 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 do not make an attempt to snap the dangling nodes to the closest polylines, therefore it is very important to use [Clean Dangling Nodes function](#) in order to be ensured that there will be no loss of data. The process of removing non polygon elements might go through several iterations, because in many cases removing one dangling polyline makes another polyline dangling.
 - The third step is the actual building of the polygons
 - The forth step cleans the resulting polygon feature class from some minor inconsistencies
- 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
	

ToolBox implementation

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

Command line syntax

ET_GPBUILDpolygons <input_dataset> <out_feature class> {clean_polylines} {fuzzy_tolerance}

Parameters

Expression**Explanation**

<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{clean_polylines}	A Boolean indicating whether the input polyline dataset will be cleaned before building.
{fuzzy_tolerance}	A Double setting the Fuzzy tolerance (in the units of the input dataset) to be used if the {clean_polylines} is True. If {clean_polylines} is False this parameter is ignored
{label_points}	A point feature class or feature layer to be used as a source for the polygon attributes.

Scripting syntax

ET_GPBuildPolygons (input_dataset, out_feature class, clean_polylines, fuzzy_tolerance,label_points)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

BuildPolygons(pInFC As IFeatureClass, sOutFName As String, Optional bCleanPolylines As Boolean = True, Optional dFuzzy As Double = 0, Optional pLabelFC As IFeatureClass = Nothing) As IFeatureClass

Clean Gaps

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Finds the gaps between polygons and holes within polygons and fills them with new polygons.

Inputs:

- A polygon feature layer

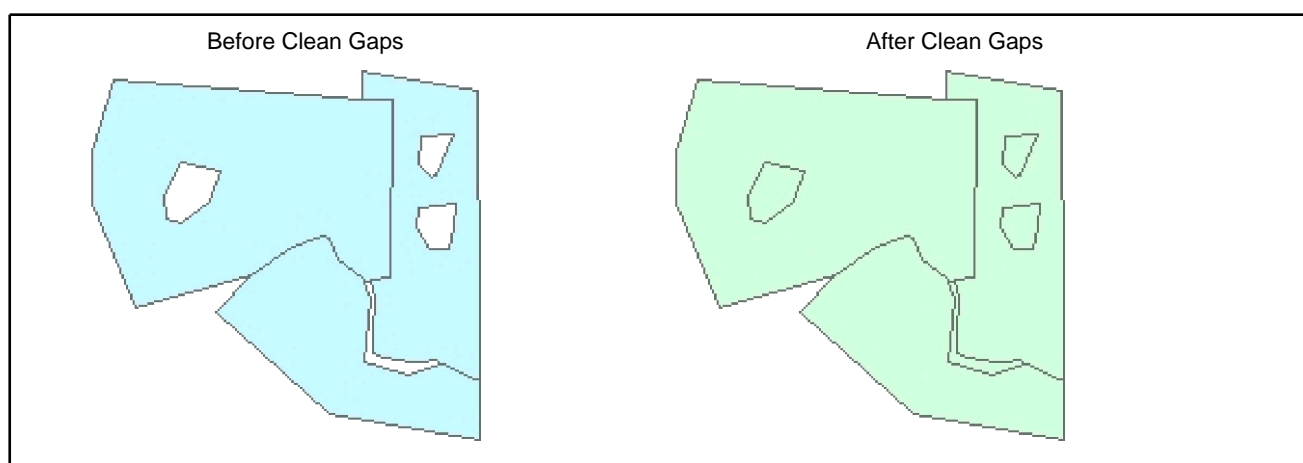
Outputs:

- New polygon feature class 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:



ToolBox implementation

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

Command line syntax

ET_GPCleanGaps <input_dataset> <out_feature_class>

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPCleanGaps (input_dataset, out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanGaps(plnFC As IFeatureClass, sOutFName As String, Optional bCleanPolygons As Boolean = False, Optional dFuzzy
As Double = 0.000001) As IFeatureClass

Copyright © Ianko Tchoukanski

Clean Polygon

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Ensures topological correctness of a polygon feature data set.

Inputs:

- A polygon feature layer
- Fuzzy tolerance

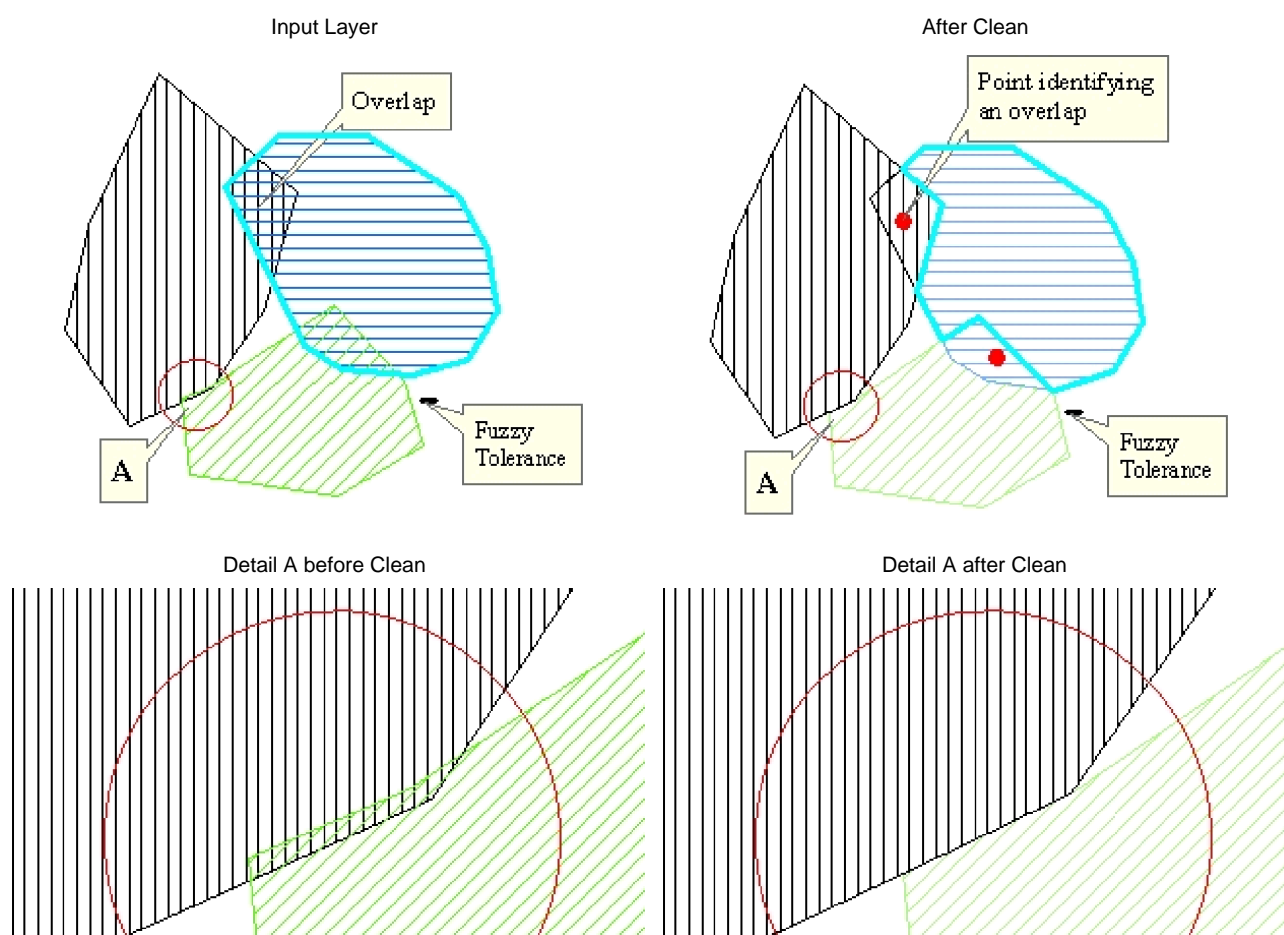
Outputs:

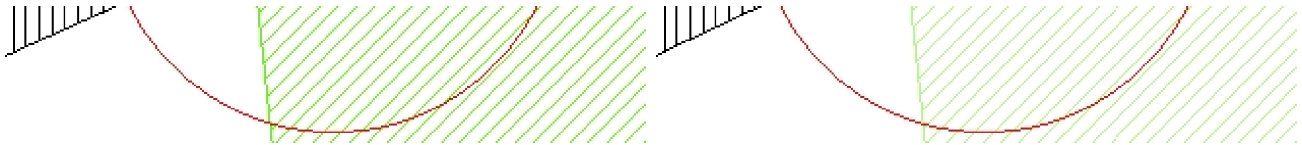
- New topologically correct Polygon feature class (no overlaps present)
 - Redundant data (overlaps and gaps smaller than the fuzzy tolerance) will be eliminated
 - The overlaps greater than the fuzzy tolerance are converted into new polygons.
 - Every new polygon carries the attributes of one of the 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. Several fields are added to the point attribute table enabling the user to identify the overlapping polygons in this location :
 - [Num_Over] - the number of overlapping polygons in this location.
 - [Old_FID1], [Old_FID2] ...[Old_FIDn] carrying the original IDs of the overlapping polygons

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 Wizard" and "Eliminate Wizard"
- A Fuzzy tolerance of 0 may be used if the original shapes have to be preserved exactly the same. In this case all the overlaps will be converted into new polygons.

Example:





Attribute tables

Input table

FID	Id	aaa	bbb
0	70	a	xxx
1	70	b	yyy
2	70	c	zzz

Result table

Id	aaa	bbb
70	c	zzz
70	b	yyy
70	a	xxx
70	b	yyy
70	a	xxx

Overlaps table

Num_Over	Old_FID1	Old_FID2
2	1	2
2	0	1

ToolBox implementation

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

Command line syntax

ET_GPCleanPolygon <input_dataset> <out_feature class> <fuzzy_tolerance>{overlaps_feature_class}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{fuzzy_tolerance}	A Double setting the Fuzzy tolerance (in the units of the input dataset) to be used if the {clean_polylines} is True. If {clean_polylines} is False this parameter is ignored
{overlaps_feature_class}	A String - the full name of the output point feature class that identifies the overlaps in the input feature class. (A feature class with the same full name should not exist)

Scripting syntax

ET_GPCleanPolygon (input_dataset, out_feature class, fuzzy_tolerance, overlaps_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CleanPolygons(pInFC As IFeatureClass, sOutFName As String, dFuzzy As Double, Optional sOverlapsFName As String = "") As IFeatureClass

Create Centerlines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Not available in the versions of ET GeoWizards for ArcGIS 8.x

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:

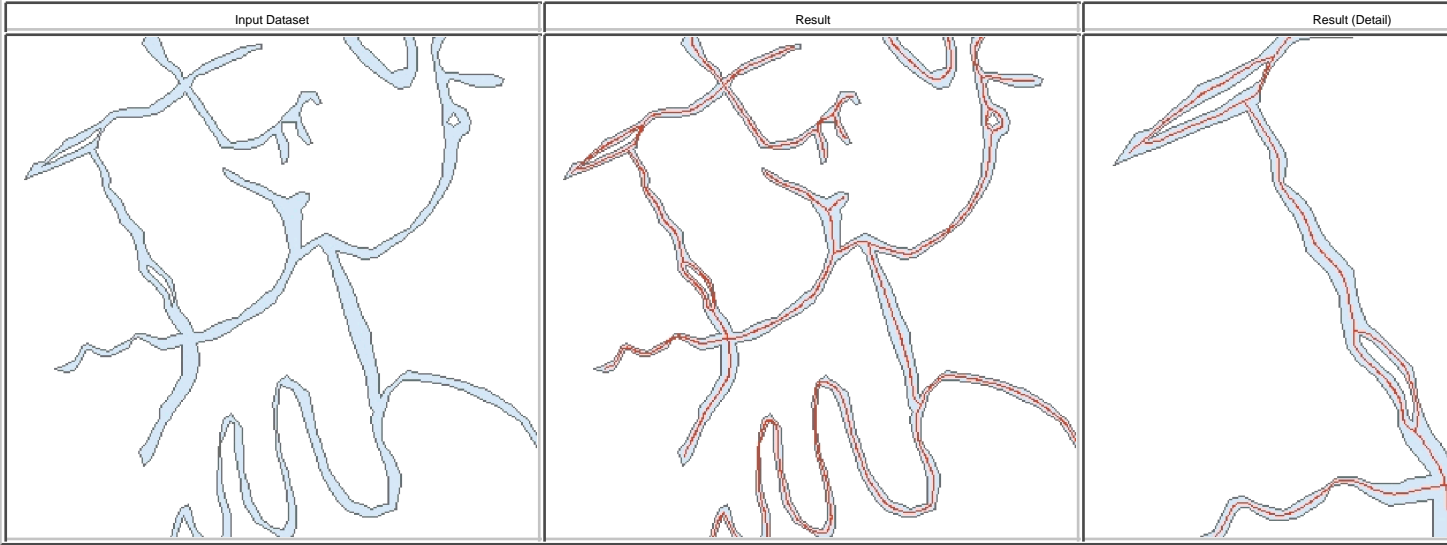
- New polyline feature class

Notes:

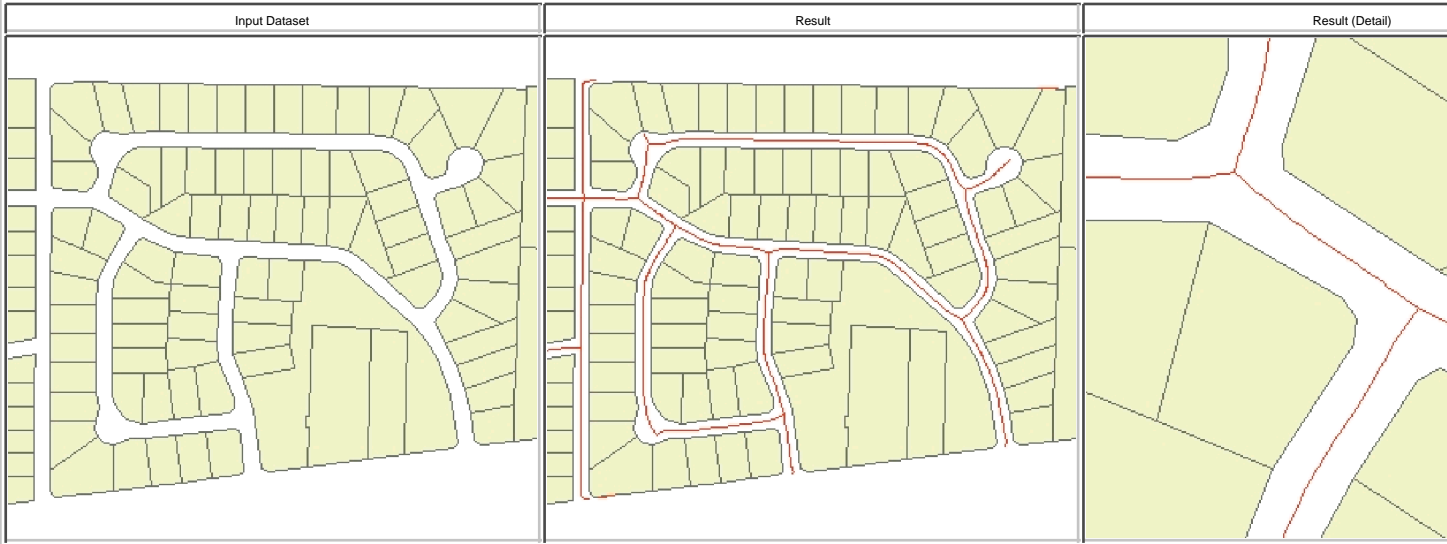
- If the "Inside" option is used, the input polygons should represent linear by nature features (rivers, roads, etc.)
- If the "Outside" option is used, the gaps between the input polygons should represent linear by nature features (streets, etc.)
- The new feature class will not have any attributes.
- 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.
- An option is available for smoothing the resulting centerlines. It is recommended not to use this option, but use the [Smooth Polylines](#) function after inspecting the results.
- The function is CPU and RAM intensive. Using it on large datasets might need long processing time.

Examples:

- Centerlines Inside Polygons



- Centerlines Outside Polygons



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

Command line syntax

ET_GPCreateCenterlines <input_dataset> <out_feature class> <maximum_width> <minimum_width> <centerlines_type> {smooth_centerlines}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<maximum_width>	A Double representing the aggregation distance. The polygons that are closer to each other than this distance will be combined
<minimum_width>	A Double representing the minimum area of holes to be preserved. All holes with area less than this tolerance will be removed.
<centerlines_type>	Required. A String indicating the type of centerlines to be created. Case sensitive. <ul style="list-style-type: none">● 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.
{smooth_centerlines}	A Boolean indicating whether the resulting centerlines will be smoothed. B-Spline smoothing algorithm is used

Scripting syntax

ET_GPCreateCenterlines (input_dataset, out_feature class, maximum_width, minimum_width,centerlines_type, smooth_centerlines)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CreateCenterlines(plnFC As IFeatureClass, sOutFName As String, dMaxWidth As Double,dMinWidth As Double, sCenterlinesType As String, Optional bSmooth As Boolean = False) As IFeatureClass

Dissolve Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

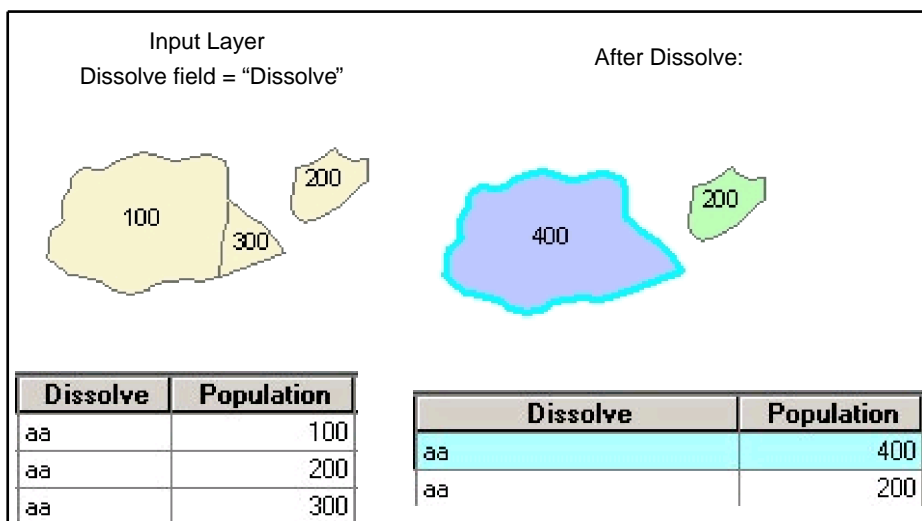
Outputs

- An aggregated polygon feature class. Only the polygons with common boundaries that have the same values for the dissolve fields will be aggregated
- No multi-part polygons 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 Dissolve Polygon Wizard works similarly to the dissolve function of GeoProcessing Wizard with a couple of improvements:
 - Multiple dissolve fields can be used
 - It does real dissolving – the non adjacent polygons with the same values for the dissolve fields will not be merged and no multi-part features will be created.
- If multi-part features are already present in the layer to be dissolved, the Dissolve Polygons Wizard will explode them first. The attributes will be distributed as follows:
 - For the fields that the user has selected SUM as an update rule, the values will be proportionally distributed between the parts
 - For the rest of the fields the attributes will be copied over.
- It is recommended the Explode Wizard to be used before dissolve in order to ensure proper distribution of the attribute values of the numeric fields.

Example:



ToolBox implementation

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

Command line syntax

ET_GPDissolvePolygons<input_dataset> <out_feature class> <dissolve_field_list> {Update_Rules_List}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{dissolve_field_list}	A String - a list of field names to be used for dissolving.
{Update_Rules_List}	A String - a list of fields with their update rules.

Scripting syntax

ET_GPDissolvePolygons(input_dataset, out_feature class, dissolve_field_list, Update_Rules_List)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\suburbs.shp"
result = "C:\\data\\fgdb_test.gdb\\dissolved"
arcpy.gp.ET_GPDissolvePolygons(input_dataset, result, "Name;Type", "Population Sum; City First")
```

.NET implementation

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

DissolvePolygons(pInFC As IFeatureClass, sOutFName As String, dissolveFields As List(Of String), Optional updateRules As Dictionary(Of String, String) = Nothing) As IFeatureClass

Eliminate

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Eliminates unwanted polygons (slivers) by merging them into the neighboring polygons or deleting them.

Inputs:

- A polygon feature layer
- Selection method
 - Attribute query - introduces the standard ArcGIS 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
 - Delete - will delete all selected polygons (considered slivers)
 - 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.
 - Identify - will record in the attribute table selected potential slivers. This allows for visual inspection before proceeding with elimination.

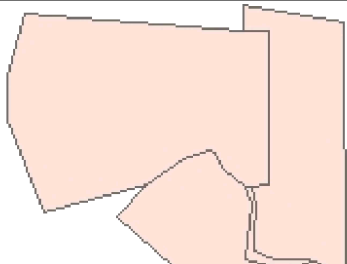
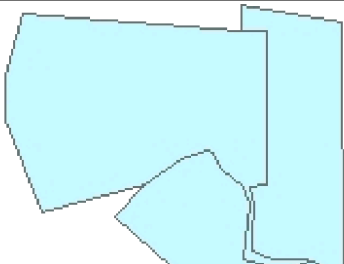
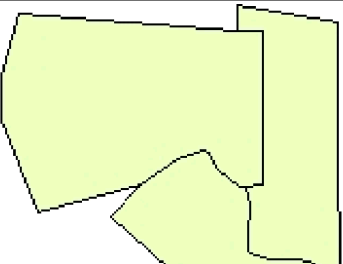
Outputs:

- New polygon feature class with selected polygons eliminated or identified as slivers
- New field is added to the attribute table :
 - [ETO_Type]
 - The polygons not affected from elimination will have value - "Original"
 - The slivers that were not eliminated will have value - "Sliver"
 - The polygons that a sliver has been merged to - value - "Changed"

Notes :

- If the source of the original data set is a feature class, to operation can be performed on the original. If the source is a Coverage or Geodatabase, the only option is creating a new feature class
- If the option "Calculate statistics" is selected, the next step will contain valuable statistics - Min, Max and Mean values for the selected method (Thickness ratio or circularity) that will help the user when assigning the ratio to be used for elimination
- Checked "Update Area and Perimeter" option will cause recalculation of the Area and perimeter values. The rest of the original attributes are preserved
- 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.

Examples:

Before Eliminate	Eliminated with Largest Area option	Eliminated with Longest boundary option
		



ToolBox implementation

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

Command line syntax

ET_GPEliminate <input_dataset> <out_feature class> <Attrib | Thickness | Circularity> <Delete | Largest | Longest | Join Field> {SQL_Expression} {thickness_circularity_ratio} {Join_Field}

Parameters

Part	Description
<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Attrib Thickness Circularity>	Selection Method. A String - the method to be used for selection of the polygons to be eliminated. The values can be: <ul style="list-style-type: none"> "Attrib" - an attribute query will be used "Thickness" - the thickness ratio will be used for selection. The thickness 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. "Circularity" - the thickness ratio will be used for selection. For a circle the circularity will be 1. The thinner the polygon is the smaller the circularity will be.
<Delete Largest Longest Join Field>	Elimination Method. A String representing the elimination method to be used. The values can be: <ul style="list-style-type: none"> "Delete" - The selected polygons will be deleted "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 - the Where Clause that will be used if "Attrib" selection method is used. Ignored if any of the other selection methods is used.
{Join_Field}	A String - A String representing the name of the field to be used with the "Join Field" option.
{thickness_circularity_ratio}	A Double representing the value of the ratio to be used. If the selection method is "Thickness" this is the Thickness ratio, if "Circularity" - this is the Circularity ratio. If "Attrib" selection method is specified this value is ignored

Scripting syntax

ET_GPEliminate (input_dataset, out_feature class, selection_method, elimination_method, SQL_Expression, thickness_circularity_ratio,Join_Field)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\eliminate_result"
arcpy.gp.ET_GPEliminate (input_dataset, result, "Thickness", "Longest", "", 0.35, "")
```

.NET implementation

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

Eliminate(pInFC As IFeatureClass, sOutFName As String, sSelectionMethod As String, sEliminationMethod As String, Optional sExpression As String = "", Optional dRatio As Double = 0, Optional sJoinField As String = "") As IFeatureClass

Copyright © Ianko Tchoukanski

Generalize Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

Inputs:

- A polygon feature class
- Generalization Tolerance (maximum offset) - the maximum distance that the generalized polyline will differ from the original one

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:

- The function goes through a complex process in order to generalize the polygons and preserve the topological relations between them. See [this white paper](#) for details about the process.
- If you have stand alone polygons only (there are no polygons that share a common boundary) you can use the "Stand alone only" option to make the process faster. DO NOT use this option if there are polygons that share boundary.
- The Generalization tolerance should be specified in the units of the spatial reference of the input feature class

ToolBox implementation

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

Command line syntax

ET_GPGeneralizePolygons <input_dataset> <out_feature_class> <generalize_tolerance> {stand_alone}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<generalize_tolerance>	A Double representing Generalize tolerance (maximum offset) - the maximum distance that the generalized polygons will differ from the original ones
{stand_alone}	A Boolean. Indicates whether the option for stand-alone polygons to be used. VERY IMPORTANT - Set this parameter to TRUE only on polygon datasets that have only stand-alone polygons (no common boundaries between the polygons).

Scripting syntax

ET_GPGeneralizePolygons (input_dataset, out_feature_class, generalize_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

GeneralizePolygons(pInFC As IFeatureClass, sOutFName As String, dGenTol As Double, Optional bStandAlone As Boolean = False) As IFeatureClass

Copyright © Ianko Tchoukanski

Get Adjacent Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
- Option: Consider touching in a single point polygons neighbours - see example below

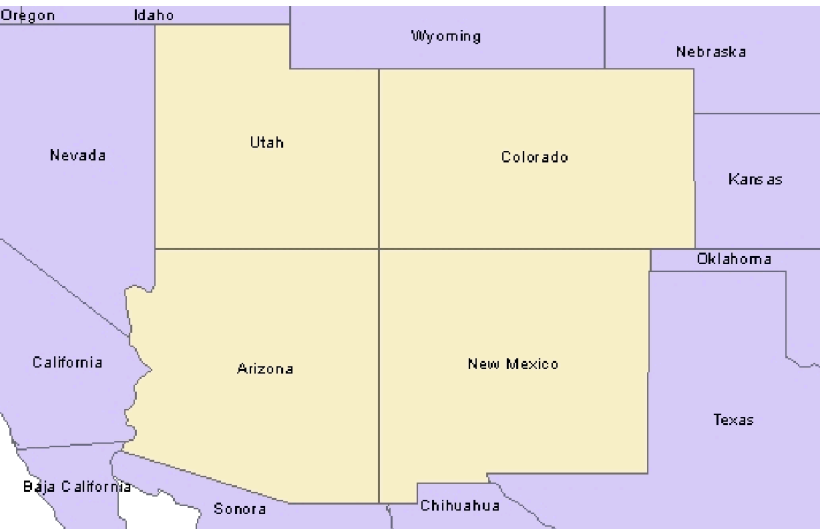
Outputs:

- 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:

- The input polygon feature class should not have overlapping polygons. If overlaps are present - new polygons will be created from them (Clean will be performed) before collecting the adjacent polygons.
- 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	Do not consider Touching (in a single point) polygons neighbours.	Consider Touching (in a single point) polygons neighbours.	
	ET_Adj	ET_Count	ET_Adj
Arizona	California,Sonora,Nevada,NewMexico,BajaCalifornia,Utah	6	California,Sonora,NewMexico,BajaCalifornia,Nevada,Utah, Colorado
Colorado	NewMexico,Utah,Wyoming,Kansas,Nebraska,Oklahoma	6	Arizona ,NewMexico,Utah,Oklahoma,Wyoming,Nebraska,Kansas
New Mexico	Chihuahua,Sonora,Arizona,Colorado,Oklahoma,Texas	6	Chihuahua,Sonora,Texas,Arizona, Utah ,Colorado,Oklahoma
Utah	Nevada,Arizona,Colorado,Wyoming,Idaho	5	Nevada,Arizona,Idaho, NewMexico ,Colorado,Wyoming

ToolBox implementation

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

Command line syntax

ET_GPPolygonAdjacency <input_dataset> <out_feature_class> <ID_field> {include_touching}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<ID_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 used values will be used to save in the adjacency string.
{include_touching}	A Boolean indicating whether the polygons touching in a single point to be considered adjacent (see example above).

Scripting syntax

ET_GPPolygonAdjacency (input_dataset, out_feature class,ID_field,include_touching)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonAdjacency(plnFC As IFeatureClass, sOutFName As String, sIDField As String, Optional bTouching As Boolean = False) As IFeatureClass

Partition Polygons with Polylines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

- The function goes through a complex process. See [this white paper](#) for details about the process.
- Both datasets should have the same spatial reference.

ToolBox implementation

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

Command line syntax

ET_GPPartitionPolygons <input_dataset> <split_polylines> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<split_polylines>	A Polyline feature class or feature class. NOTE: The spatial references of <input_dataset> and the <split_polylines> must have the same Geographic Coordinate System
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPartitionPolygons (first_dataset, second_dataset, out_feature_ class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PartitionPolygons(pInFC As IFeatureClass, pSplitFc As IFeatureClass, sOutFName As String) As IFeatureClass

Polygon Global Snap

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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, polyline or polygon
- Snap tolerance
- Snap options1 (Snap What)
- Snap options2 (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 tolerances)

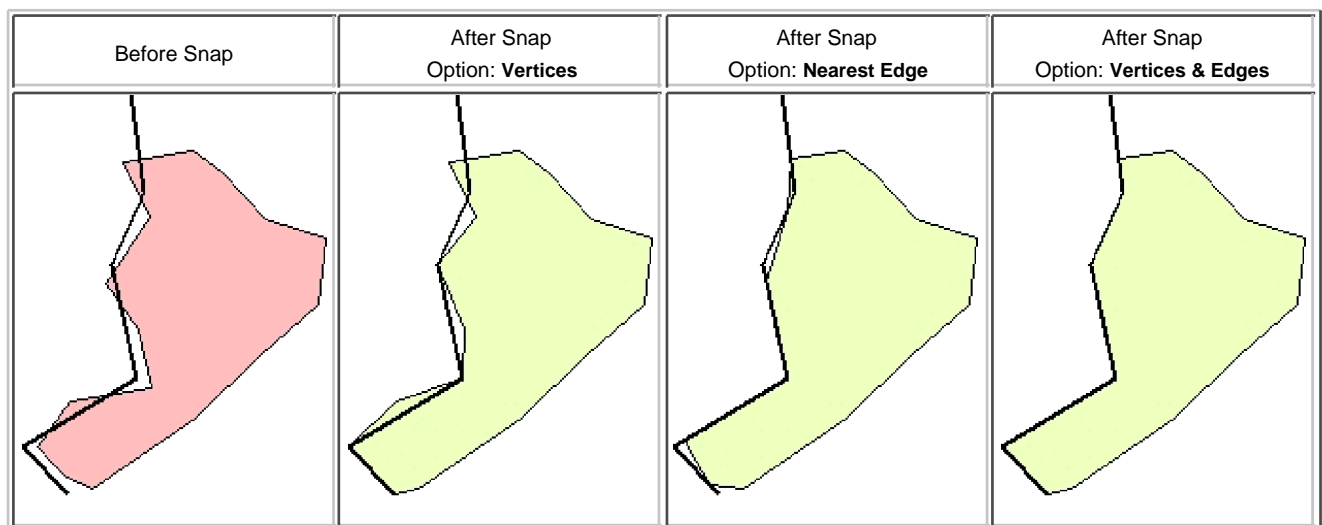
Options:

- Snap Options 1 (Snap What) - this options lets the user set which elements of the source polylines to 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 polylines 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 & 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:

- If the "Insert new Vertices" option is used the polygons will be cleaned from overlaps.
- 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:



ToolBox implementation

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

Command line syntax

ET_GPSnapPolygons <input_dataset> <Reference_dataset> <out_feature class> <snap_tolerance> <snap_what> {snap_to_vertices}

{snap_to_nearest}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<Reference_dataset>	A Point, Polyline or Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<snap_tolerance>	A Double representing the Search tolerance (in the units of the input_dataset) to be used
<snap_what>	A String indicating what parts of the input polylines will be snapped. Possible values: <ul style="list-style-type: none">● Vertex: All the vertices of the source polylines will be used.● Insert: This option will get the vertices from the features of the Reference_dataset 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 Reference_dataset.
{snap_to_vertices}	A Boolean indicating whether snapping to the closest vertex of the nearest feature from the Reference_dataset to be used
{snap_to_nearest}	A Boolean indicating whether snapping to the nearest point of the nearest feature from the Reference_dataset to be used

Scripting syntax

ET_GPSnapPolygons (input_dataset, Reference_dataset, out_feature_class, snap_tolerance, snap_what, snap_to_vertices, snap_to_nearest)

.NET implementation

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

SnapPolygons(pInFC As IFeatureClass, pRefFC As IFeatureClass, sOutFName As String, dSnapTol As Double, sSnapWhat As String, Optional bVertex As Boolean = False, Optional bNearest As Boolean = False) As IFeatureClass

Smooth Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Smooth the features of a polygon layer using three different smoothing algorithms"

Inputs:

- A polygon feature class
- Smooth method (see examples [here](#))
 - 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 - best for smoothing polygons
 - 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) - **not recommended when smoothing polygons**
 - 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 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:

- The function goes through a complex process in order to smooth the polygons and preserve the topological relations between them. See [this white paper](#) for details about the process.
- Using T-Spline method can produce sometimes incorrect results (missing polygons). Inspect your data after smoothing.

- The Generalization and Densification tolerances should be specified in the units of the spatial reference of the input feature class

ToolBox implementation

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

Command line syntax

ET_GPSSmoothPolygonsBS <input_dataset> <out_feature_class> <smoothness> <freedom> {densify_tolerance}
{generalize_tolerance}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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 <smoothness> parameter, the slower the process will be.
<freedom>	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
{densify_tolerance}	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_tolerance}	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.

Scripting syntax

ET_GPSSmoothPolygonsBS (input_dataset, out_feature_class, smoothness, freedom, densify_tolerance, generalize_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SmoothPolygonsBS(plnFC As IFeatureClass, sOutFName As String, iSmoothness As Short, iFreedom As Short, Optional dDensTol As Double = 0, Optional dGenTol As Double = 0) As IFeatureClass

Polygon Characteristics

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Calculates some characteristics of the polygons from a polygon dataset

Inputs:

- A Polygon feature class

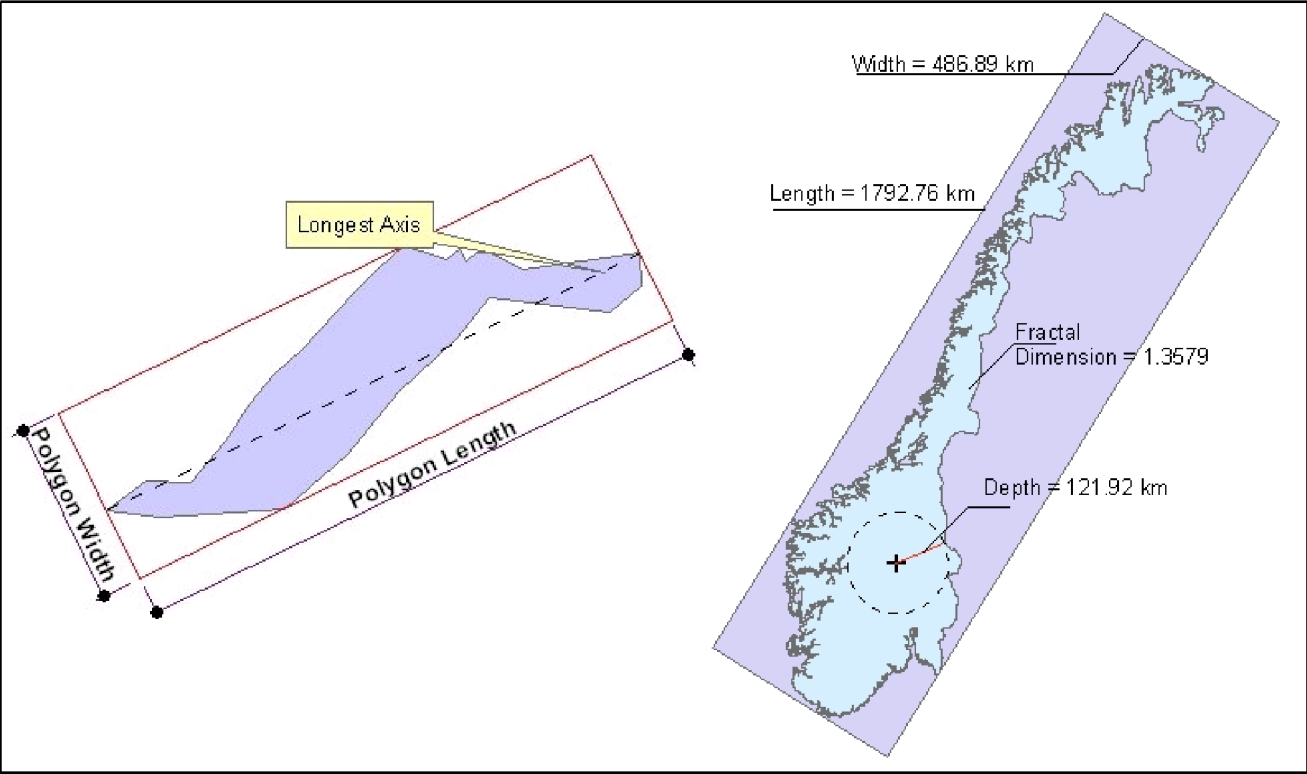
Outputs:

- The results can be added to the input feature class or a new polygon feature class. All attributes of the original features 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 Point](#) and [Polygon To Maximum Inscribed Circle](#) functions
 - 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/>

ToolBox implementation

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

Command line syntax

ET_GPPolygonCharacteristics <input_dataset> {polygon_depth} {fractal_dimension} {precision}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer.
{fractal_dimension}	A Boolean indicating whether to calculate polygon depth or not.

{polygon_depth}	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.

Scripting syntax

ET_GPPolygonCharacteristics (input_dataset, polygon_depth, fractal_dimension, precision)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonCharacteristics(plnFC As IFeatureClass, Optional bCalculateDepth As Boolean = False, Optional bCalculateFract As Boolean = False, Optional iPrecision As Short = 2) As Boolean

Polygon To Polyline Advanced

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts the polygon boundaries to polylines.

- Creates a topologically correct (nodes at intersections, no overlaps) Polyline feature class
- 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

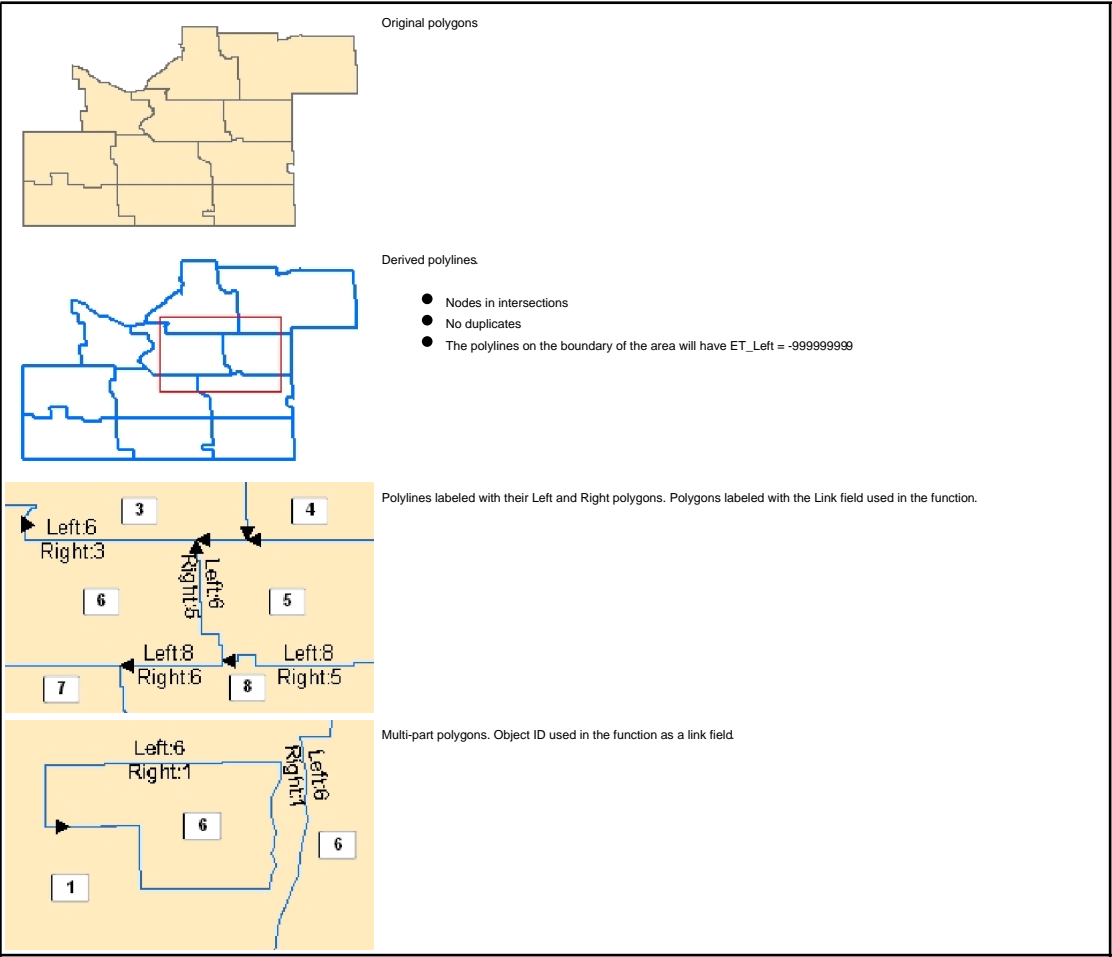
Outputs:

- New topologically correct Polyline feature class. 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 feature class representing the labels of the input polygons. The attributes of the input polygons are preserved in the Point Attribute Table

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.
- A Fuzzy tolerance of 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 -999999999. In topologically correct polygon dataset this should indicate the outer boundary (neighboring with the so-called Universal Polygon. Values of -999999999 in the interior of the polygon dataset will indicate gaps or overlaps in the data.
- If the input feature class has multi-part features and the Object ID field is used as a link field, the polylines will have for Left and Right polygons the IDs of the original polygons. The label points however will carry in the ET_ID field the IDs of the exploded polygons.

Example:



ToolBox implementation

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

Command line syntax

ET_GPPolygonToPolylineAdvanced <input_dataset> <out_feature_class> <fuzzy_tolerance> <link_field> {<label_feature_class>}

Parameters

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

<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class.
<fuzzy_tolerance>	A Double - the gaps smaller than this tolerance will be closed. The units of the parameter are in the spatial reference of the input feature class
<link_field>	A String - the name of the field to be used as a link. The field must exist in the input feature class. The type of the field can be Short Integer , Long Integer, Double, Single, String or Object ID. The values of this field will be saved as Left and Right polygons for each polyline.
{label_feature_class}	A String - the full name of the label feature class.

Scripting syntax

```
ET_GPCleanContourGaps (input_dataset, out_feature_class, fuzzy_tolerance, link_field,label_feature_class)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

```
PolygonToPolylineAdvanced(pInFC As IFeatureClass, sOutFName As String, dFuzzy As Double, sLinkField As String, Optional sLabelsFName As String = "") As IFeatureClass
```

Fill Polygon Holes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Ensures topological correctness of a polygon feature data set.

Inputs:

- A polygon feature layer
- Maximum area of the holes to be removed

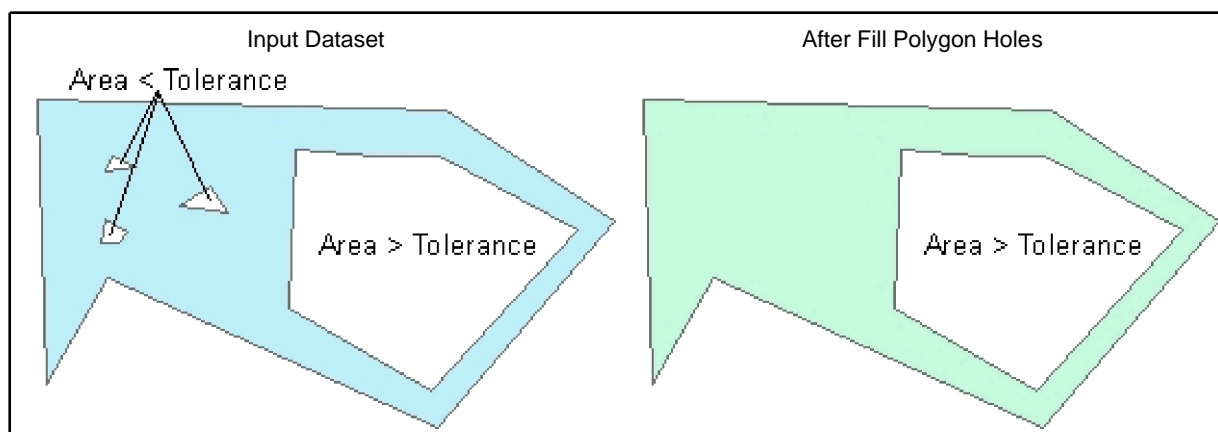
Outputs:

- 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:



ToolBox implementation

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

Command line syntax

ET_GPFillPolygonHoles<input_dataset> <out_feature_class>{max_area}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{max_area}	A Double. Holes with larger area than this tolerance will not be removed.

Scripting syntax

ET_GPFillPolygonHoles(input_dataset, out_feature_class,max_area)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FillPolygonHoles(plnFC As IFeatureClass, sOutFName As String, Optional dAreaTol As Double = 0) As IFeatureClass

Copyright © Ianko Tchoukanski

Polygon To Polyline

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a polygon data set to a polyline feature class

Inputs:

- A polygon feature layer

Outputs:

- New polyline feature class

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.
- If the input is of PolygonZ(M) type, the output will be of PolylineZ(M) type. The Z(M) values will be preserved.

ToolBox implementation

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

Command line syntax

ET_GPPolygonToPolyline <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPolygonToPolyline (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonToPolyline(plnFC As IFeatureClass, sOutFName As String) As IFeatureClass

Polygon To Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a polygon data set to a point feature class

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.
 - ☐ Deepest Point - a single point per polygon is created - the inside point farthest from the polygon boundary. The distance from the polygon boundary is stored in the ET_Depth field of the point attribute table
- 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)

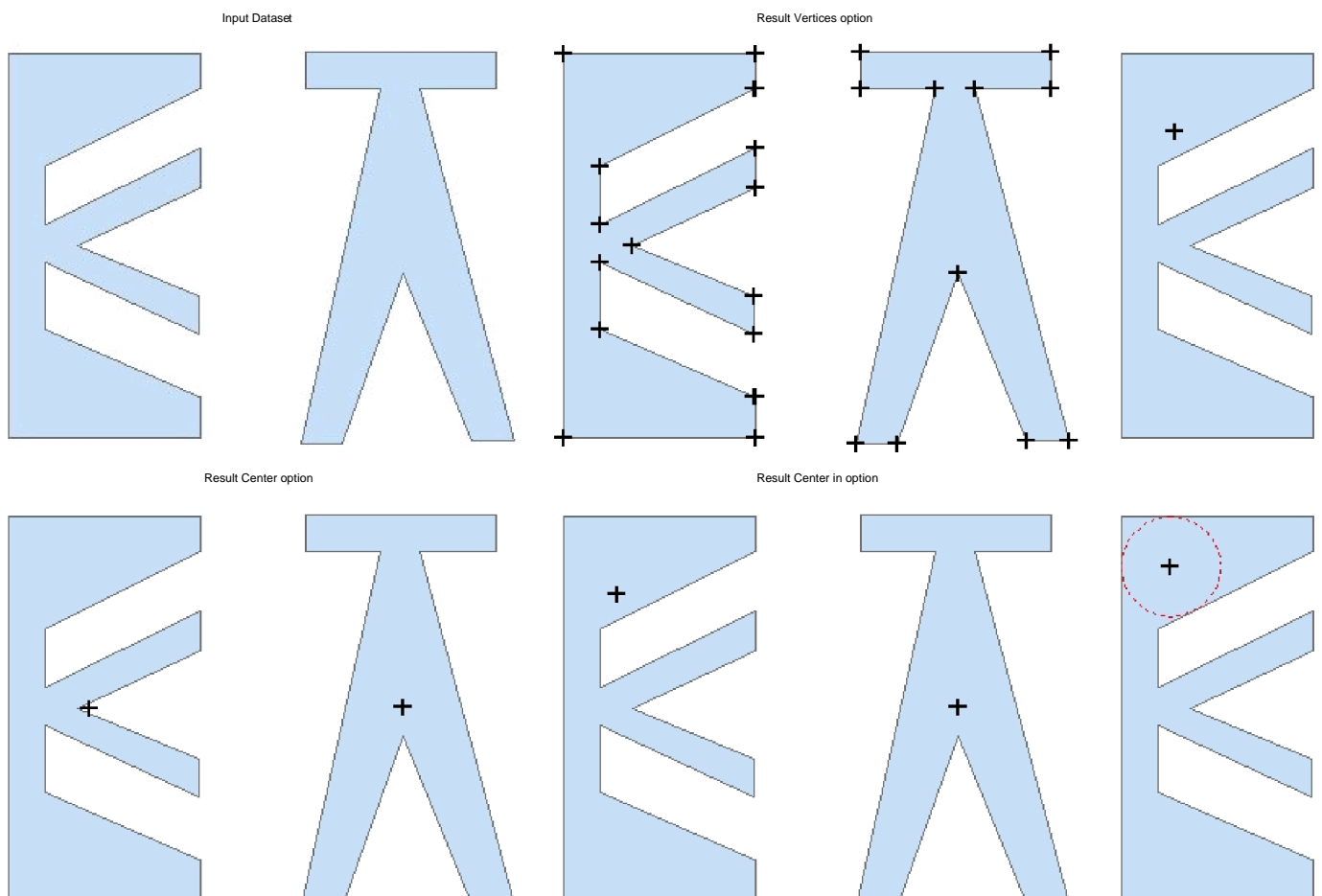
Outputs:

- New point feature class.
 - ☐ 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 loosing 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.
 - [ET_DEPTH] - the distance from the deepest point to the polygon boundary. - **only if "Deepest Point" conversion option is used**

Notes :

- See above for the use of the "Remove duplicate points" option
- The functionality of the PolygonZ(M) To Point function available in the pre 11.0 versions is entirely included in this function.

Examples:



ToolBox implementation

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

Command line syntax

ET_GPPolygonToPoints <input_dataset> <out_feature class> <Vertex | Center | Label> {remove_duplicates} {calc_point_pos} {keep_ZM}

Parameters

Part	Description
<input_dataset>	A Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Vertex Center Label CenterIn DeepestPoint>	Convert Option. A String - the export option to be used. The available options are (Case sensitive) : <ul style="list-style-type: none">● "Vertex" - the vertices of all polygons will be converted to points. If {remove_duplicates} is <i>True</i> the duplicate points created from the vertices of two adjacent polygons will be represented by one point.● "Label" - 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).● "Center" - the Center points represent the centroid of a polygon. Therefore sometimes they might be located outside of the polygon● "CenterIn" - points representing the centroids of the polygons. If the centroid occurs outside of the polygon, the point is moved to be in the polygon.● "DeepestPoint" - a single point per polygon - the inside point farthest from the polygon boundary.
{remove_duplicates}	A Boolean used only with Convert Option: "Vertex". If <i>True</i> the duplicate points representing coincident vertices of two or more adjacent polygons will be removed.
{calc_point_pos}	A Boolean indicating whether the position of the points along the polygon boundary to be calculated (only if the "Vertex option is used)
{keep_ZM}	A Boolean indicating whether the the output will be of Z(M) type (only if the input dataset is of Z(M) type)

Scripting syntax

ET_GPPolygonToPoints (input_dataset, out_feature class, convert_option, remove_duplicates,calc_point_pos, keep_ZM)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonToPoints(plnFC As IFeatureClass, sOutFName As String, sOption As String, Optional bDuplicates As Boolean = False, Optional bPos As Boolean = False, Optional bKeepZM As Boolean = False) As IFeatureClass

Polyline To Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a polyline data set to a point feature class

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.

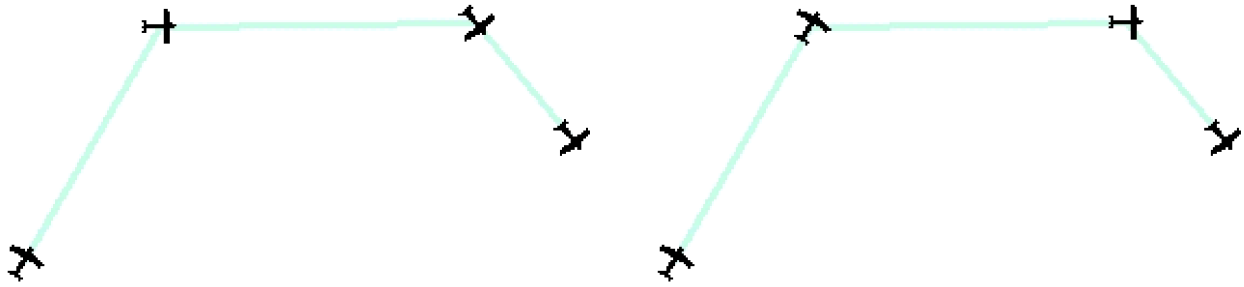
Notes :

- 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
- The functionality of the PolylineZ(M) To Point function available in the pre 11.0 versions is entirely included in this function.

Example:

Angle at the start of segments

Angle at the end of segments



ToolBox implementation

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

Command line syntax

```
ET_GPPolylineToPoints <input_dataset> <out_feature class> <Vertex | Middle | Node> {remove_duplicates} {calc_point_pos}
{point_angle} {angle_at_start} {keep_ZM}
```

Parameters

Part	Description
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Vertex Middle Node>	Convert Option. A String - the export option to be used. The available options are (Case sensitive): <ul style="list-style-type: none"> "Vertex" - - the vertices of all polylines will be converted to points. If {remove_duplicates} is <i>True</i> the duplicate points created from the nodes of two polylines sharing a common node will be represented by one point. "Middle" - only the middle point of each polyline will be exported. "Node" - only the nodes of each polyline will be exported. If {remove_duplicates} is <i>True</i> the duplicate points created from the nodes of two polylines sharing a common node will be represented by one point.
{remove_duplicates}	A Boolean used only with Convert Option= "Vertex" and Convert Option= "Node". If <i>True</i> the duplicate points representing coincident nodes will be removed.
{calc_point_pos}	A Boolean indicating whether the position of the points along the polylines to be calculated (only if the "Vertex option is used)
{point_angle}	A Boolean - indicates whether an angle attribute to be added to the point attribute table.
{angle_at_start}	A Boolean - indicates from which polyline segment to be calculated the angle. True - from the start segment, False3 from the segment end. See the main page of the function for an example.
{keep_ZM}	A Boolean indicating whether the the output will be of Z(M) type (only if the input dataset is of Z(M) type)

Scripting syntax

```
ET_GPPolylineToPoints (input_dataset, out_feature class, convert_option, remove_duplicates, calc_point_pos, point_angle,
angle_at_start, keep_ZM)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolylineToPoints(pInFC As IFeatureClass, sOutFName As String, sOption As String, Optional bAngle As Boolean = False, Optional sAng As String = "From", Optional bDuplicates As Boolean = False, Optional bPos As Boolean = False, Optional bKeepZM As Boolean = False) As IFeatureClass

Copyright © Ianko Tchoukanski

Closed Polylines To Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts closed polylines (and polyline chains) to polygons

Inputs:

- A polyline feature layer

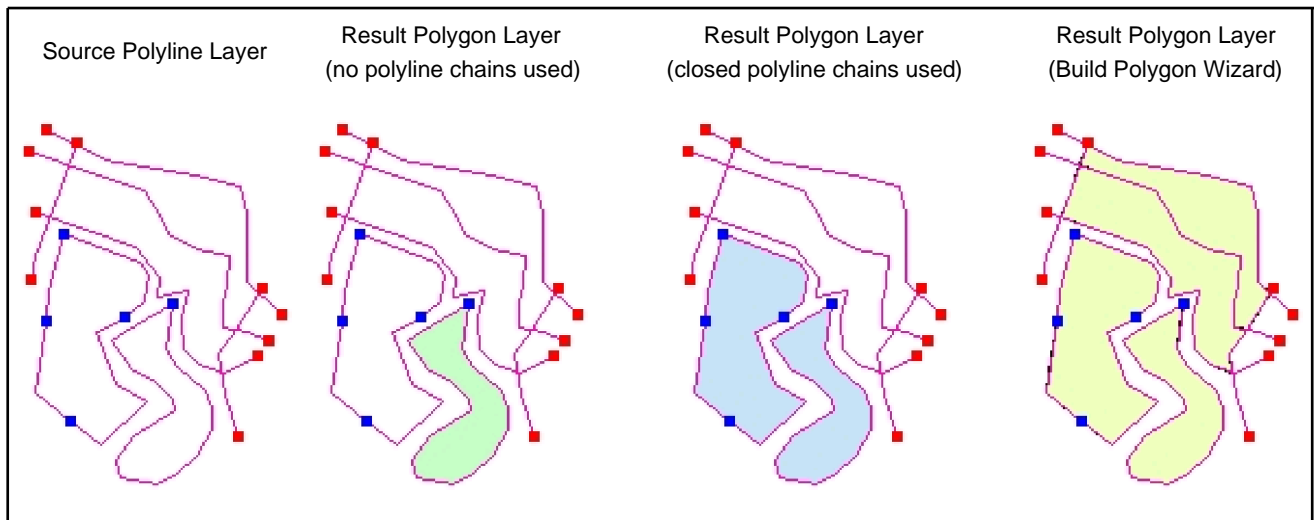
Outputs:

- New polygon feature class

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:



ToolBox implementation

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

Command line syntax

ET_GPPolylineToPolygon <input_dataset> <out_feature_class> {force_closure} {tolerance}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{force_closure}	A Boolean indicating whether the non closed polylines must be closed and converted to polygons.
{tolerance}	A Double setting the tolerance (in the units of the input dataset) to be used if the force_closure = True. If force_closure = False this parameter is ignored. Open polylines which end points are closer

to each other than this tolerance will be closed and converted to polygons

Scripting syntax

ET_GPPolylineToPolygon (input_dataset , out_feature_class, force_closure, tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolylinesToPolygons(pInFC As IFeatureClass, sOutFName As String, Optional bForce As Boolean = False, Optional dTol As Double = 0) As IFeatureClass

Copyright © Ianko Tchoukanski

Polyline To Multipoint

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a polyline data set to a Multipoint feature class

Inputs:

- A polyline feature layer

Outputs:

- New Multipoint feature class

Notes :

- The attributes of the original polylines are transferred to the resulting multipoints.

ToolBox implementation

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

Command line syntax

ET_GPPolylineToMultipoint <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPolylineToMultipoint (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolylineToMultipoint(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Point To Polyline

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a point data set to a polyline feature class. 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.

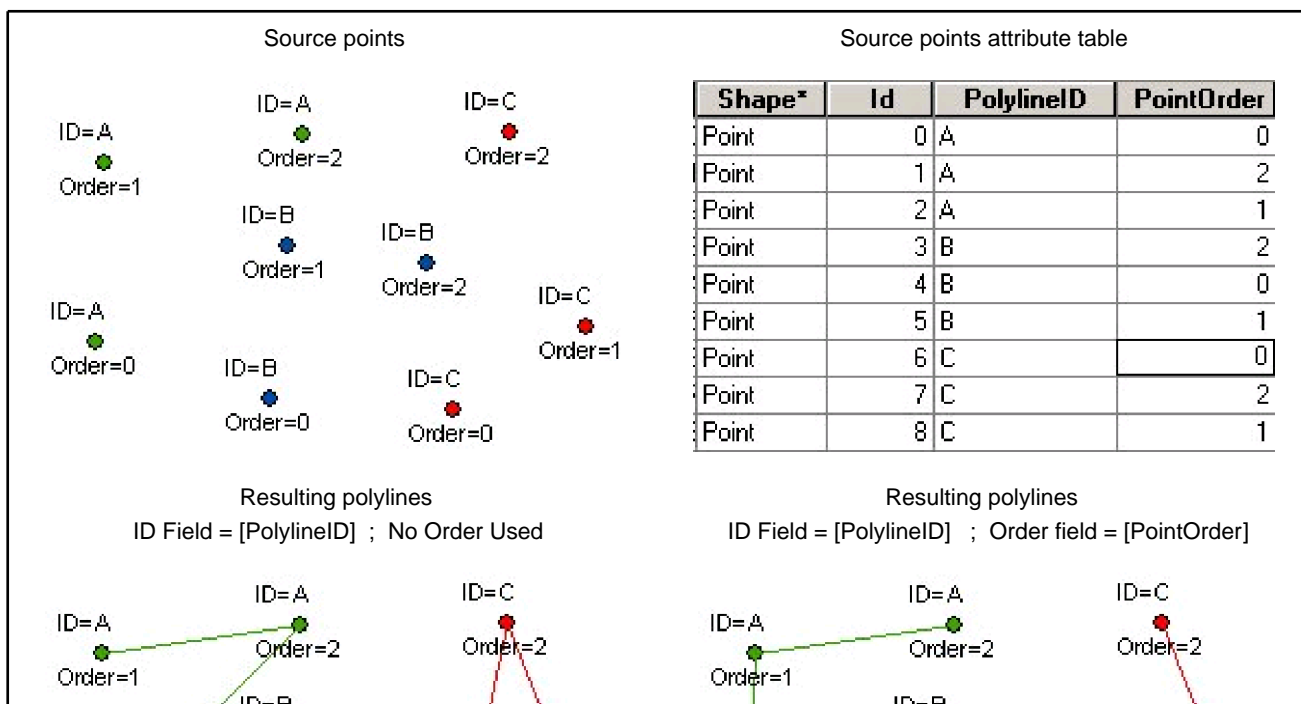
Outputs:

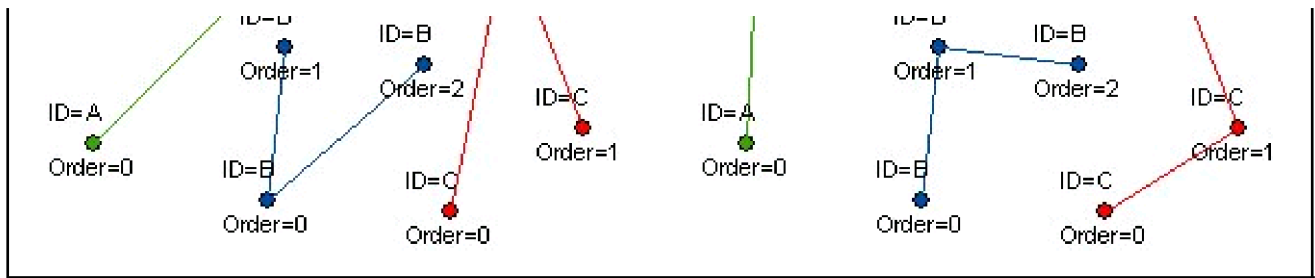
- New polyline feature class
- Fields to be added to the polyline attribute table
 - [ET_ID] - the field used as Polyline ID
 - [ET_FromAtt] - the values of the start point of the polyline in the Link field (if link field is used)
 - [ET_ToAtt] - the values of the end point of the polyline in the Link 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.
- The function entirely covers the functionality of the Point To PolylineZ(M) function available in pre 11.0 versions.

Example:





ToolBox implementation

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

Command line syntax

ET_GPPointToPolyline <input_dataset> <out_feature class> <polylineID_field> {Z_value_field} {M_value_field} {order_field}
{link_field}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<polylineID_field>	A String - the name (Case sensitive) of the field which values will indicate the points used to form a single polyline.
{Z_value_field}	A String - the name of the field which values will be used for Z values of the vertices. "Shape" can be used if the input points have Z.
{M_value_field}	A String - the name of the field which values will be used for M values of the vertices. "Shape" can be used if the input points have M.
{order_field}	A String - the name (Case sensitive) 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
{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.

Scripting syntax

ET_GPPointToPolyline (input_dataset, out_feature class, polylineID_field, Z_value_field, M_value_field, order_field, link_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointsToPolylines(pInFC As IFeatureClass, sOutFName As String, sIdField As String, Optional sOrderField As String = "",
Optional sZFieldName As String = "", Optional sMFieldName As String = "", Optional sLinkField As String = "") As
IFeatureClass

Point To Polygon

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a point data set to a polygon feature class. 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.
- The function entirely covers the functionality of the Point To PolygonZ(M) function available in pre 11.0 versions.

Example: [See the example for Point To Polyline function](#)

ToolBox implementation

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

Command line syntax

```
ET_GPPointsToPolygons <input_dataset> <out_feature class> <polygonID_field> {Z_value_field} {M_value_field}  
{order_field} {link_field}
```

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<polygonID_field>	A String - the name (Case sensitive) of the field which values will indicate the points used to form a single polygon.
{Z_value_field}	A String - the name of the field which values will be used for Z values of the vertices. "Shape" can be used if the input points have Z.

{M_value_field}	A String - the name of the field which values will be used for M values of the vertices. "Shape" can be used if the input points have M.
{order_field}	A String - the name (Case sensitive) 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
{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.

Scripting syntax

ET_GPPointsToPolygons (input_dataset, out_feature class, polygonID_field, Z_value_field, M_value_field, order_field, link_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

```
PointsToPolygons(pInFC As IFeatureClass, sOutFName As String, sIdField As String, Optional sOrderField As String = "",
Optional sZFieldName As String = "", Optional sMFieldName As String = "", Optional sLinkField As String = "") As
IFeatureClass
```

Point To Multipoint

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a point data set to a Multipoint feature class.

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 feature class
- Fields to be added to the attribute table
 - [ET_ID] - the field used as Multipoint ID

ToolBox implementation

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

Command line syntax

ET_GPPointToMultipoint <input_dataset> <out_feature class> <multipointID_field>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<multipointID_field>	A String - the name of the field which values will indicate the points used to form a single multipoint feature.
{Z_value_field}	A String - the name of the field which values will be used for Z values of the vertices. "Shape" can be used if the input points have Z.
{M_value_field}	A String - the name of the field which values will be used for M values of the vertices. "Shape" can be used if the input points have M.
{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
{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.

Scripting syntax

ET_GPPointToMultipoint (input_dataset, out_feature class, multipointID_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointsToMultipoints(pInFC As IFeatureClass, sOutFName As String, sIdField As String, Optional sOrderField As String = "", Optional sZFieldName As String = "", Optional sMFieldName As String = "", Optional sLinkField As String = "") As IFeatureClass

Point To Point Z(M)

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a point data set to a point Z (M) feature class

Inputs:

- A point feature layer
 - REQUIRED: a numeric field with Z (M) values that will be applied to the newly created Points Z (M)
- Type of the output point feature class - Z or M

Outputs:

- New point Z(M) feature class. All the original attributes are transferred.

ToolBox implementation

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

Command line syntax

ET_GPPointToPointZM <input_dataset> <out_feature class> <ZM_value_field> <Z | M>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<ZM_value_field>	A String - the name (Case sensitive) of a Numeric (integer or double) field which values will be used for assigning Z(M) values to the output points
<Z M>	Convert Option. A String - indicates the type of the output points. The available options are (Case sensitive): <ul style="list-style-type: none">● "Z" - the output will be PointZ● "M" - the output will be PointM

Scripting syntax

ET_GPPointToPointZM (input_dataset, out_feature class, ZM_value_field, convert_option)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointToPointZM(pInFC As IFeatureClass, sOutFName As String, Optional sZFieldName As String = "", Optional sMFieldName As String = "") As IFeatureClass

Multipoint To Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a multipoint data set to a point feature class

Inputs:

- A multipoint feature layer

Outputs:

- New point feature class
 - 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

ToolBox implementation

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

Command line syntax

ET_GPMultipointsToPoints <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Multipoint feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPMultipointsToPoints (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

MultipointsToPoints(pInFC As IFeatureClass, sOutFName As String, Optional bKeepZM As Boolean = False) As IFeatureClass

Multipoint To Polyline

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a Multipoint data set 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.

ToolBox implementation

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

Command line syntax

ET_GPMultipointToPolyline <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Multipoint feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPMultipointToPolyline (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

MultipointToPolyline(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Shape Z (M) To Shape

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a Z (M) data set to a plain (no Z or M) feature class

Inputs:

- A Z aware or M aware feature layer
 - Point
 - Polyline
 - Polygon

Outputs:

- New feature class
 - The original attributes are preserved

Notes:

- Multipoint is not supported. To convert Multipoint Z (M) to point use Multipoint To Point Wizard
- The Z or/and M values are dropped from the features. To preserve Z (M) values use:
 - Point Z (M) to Point Wizard
 - Polyline Z (M) to Point Wizard
 - Polygon Z (M) to Point Wizard

ToolBox implementation

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

Command line syntax

ET_GPDropZM<input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPDropZM(input_dataset out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

DropZM(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Shape To ShapeZ

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts the features of a data set to 3D features with constant Z value

Inputs:

- A point, 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, PolylineZ or PolygonZ (depending on the input) feature class. All the original attributes are transferred.

ToolBox implementation

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

Command line syntax

ET_GPShapeToShapeZ<input_dataset> <out_feature class> <Z_value_field> {Add_M}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Z_value_field>	A String - the name (Case sensitive) of a Numeric (integer or double) field which values will be used for assigning Z values to the output points
{Add_M}	A Boolean - if TRUE - M values will be added.

Scripting syntax

ET_GPShapeToShapeZ(input_dataset, out_feature_class, Z_value_field, Add_M)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ShapeToShapeZ(plnFC As IFeatureClass, sOutFName As String, sZFieldName As String, Optional bM As Boolean = False,)
As IFeatureClass

Build TIN

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Builds a Triangulated Irregular Network from a feature layer

Inputs:

- A feature layer (Point, Polyline, Polygon)
- An elevation field - numeric field that will be used

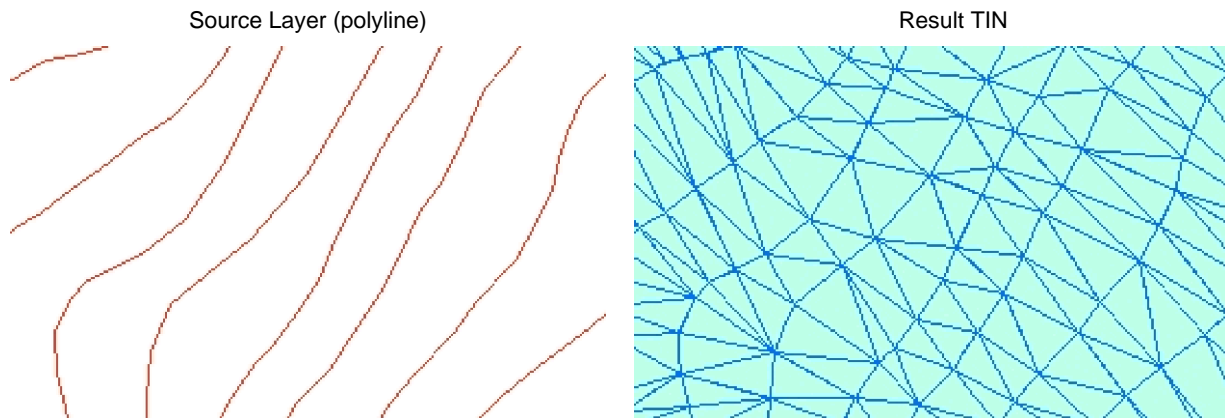
Outputs:

- New polygon Z feature class. All the polygons are triangles that comply with the Delaunay criteria. See [TIN notes](#) for more information about Triangulated Irregular Network

Notes :

- The process goes through several steps
 - Collecting the elevation points from the source layer. If the source is a polygon or polyline layer, all the vertices are used.
 - Removing duplicate points
 - Creating the TIN structure
 - Storing the polygons Z
- To achieve best results when creating TIN from a polyline layer use [Generalize Polylines function](#) or [Densify Polylines function](#) in order to remove unnecessary points or add points to the long straight segments
- The function should work with no problems on datasets with up to 2 million points.
- For advanced TIN and Raster surfaces creation, check ET Surface at <http://www.ian-ko.com>

Example:



ToolBox implementation

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

Command line syntax

ET_GPBuildTIN <input_dataset> <out_feature class> <elevation_field>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

<elevation_field> A String representing the field name (must be a numeric field) that will be used as a source for the elevation values

Scripting syntax

ET_GPBuildTIN (input_dataset, out_feature class, elevation_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

BuildPolygonZTIN(pInFC As IFeatureClass, sOutFName As String, sElevationField As String, Optional dLightAzimuth As Double = 315, Optional dLightAltitude As Double = 45) As IFeatureClass

Copyright © Ianko Tchoukanski

Analyze TIN

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Calculates several characteristics for each triangle of a TIN (polygonZ)

- Minimum elevation
- Maximum elevation
- Mean elevation
- Slope - identifies the slope, or maximum rate of elevation change for each triangle
- Aspect - the values of the output field represent the compass direction of the aspect; 0 is true north, a 90 degree aspect is to the east etc. For flat triangles (slope = 0) the value of -1 is assigned for the aspect
- Hill Shade - computes the brightness of each triangle based on a light source location.

Inputs:

- A TIN (polygonZ) feature layer
- Characteristics to be calculated
 - Parameters for Hill Shade (if Hill Shade option is selected)
 - azimuth - the azimuth angle of the light source. The azimuth is expressed in positive degrees from 0 to 360, measured clockwise from the north. The default is 315 degrees.
 - altitude - the altitude angle of the light source above the horizon. The altitude is expressed in positive degrees, with 0 degrees at the horizon and 90 degrees directly overhead. The default is 45 degrees.

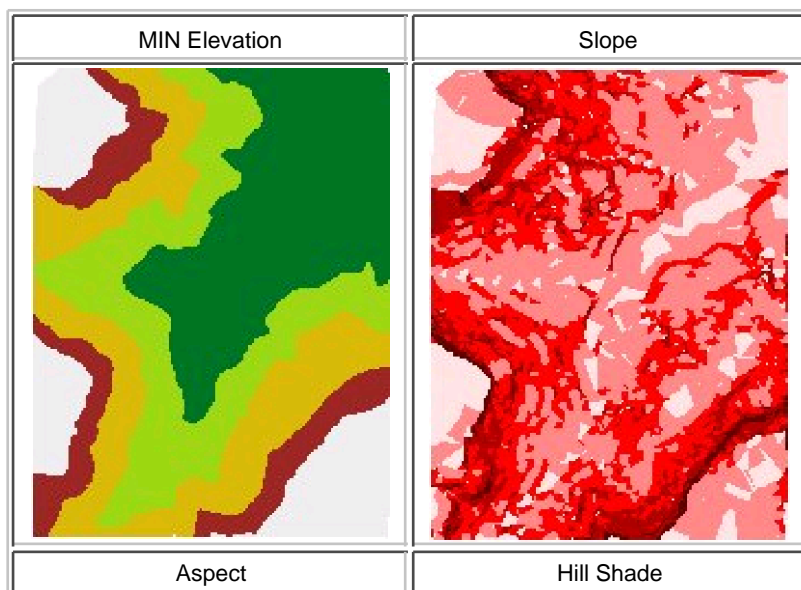
Outputs:

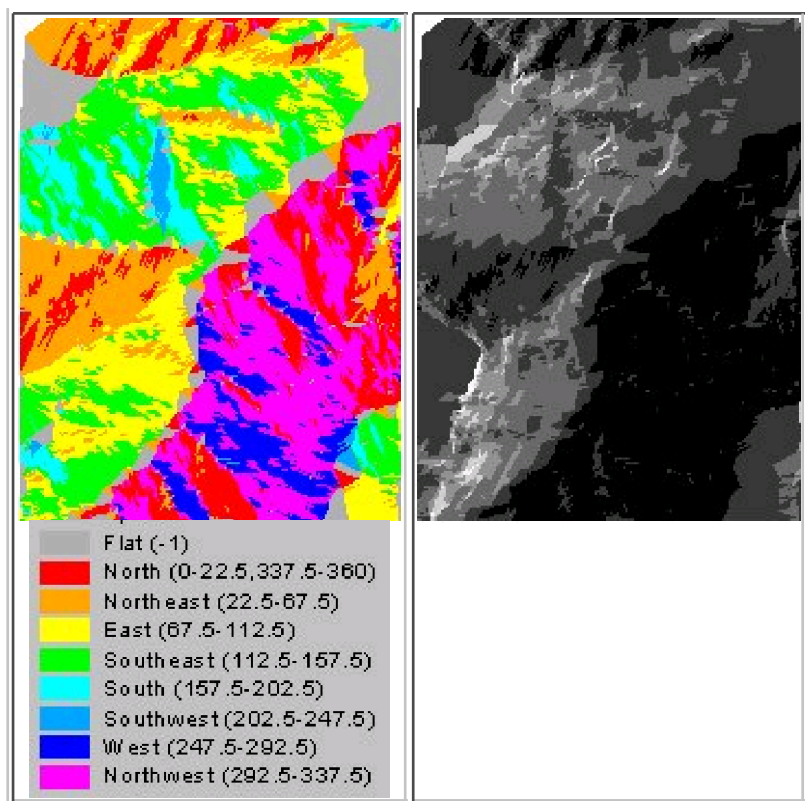
- New polygon Z feature class. Several fields are added to the attribute table, depending on the options selected
 - [ET_EIMin] - minimum elevation values for each triangle
 - [ET_EIMax] - maximum elevation values for each triangle
 - [ET_EIMean] - mean elevation values for each triangle
 - [ET_Slope] - the slope of each triangle
 - [ET_Aspect] - the aspect of each triangle
 - [ET_Hill] - a Hill Shade value for each triangle depending on the input values for Azimuth and Altitude of the light source

Notes:

- A flat triangle (Slope = 0) will have an Aspect of -1

Example: TIN (PolygonZ) classified by





ToolBox implementation

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

Calculates several characteristics (Slope, Aspect, Min & Max Elevation, Mean Elevation, Hillshade values) for each triangle of a TIN (polygonZ)

Command line syntax

ET_GPAnalyzeTIN <input_dataset> <out_feature_class> <Light_Azimuth><Light_Altitude>

Parameters

Expression	Explanation
<input_dataset>	A PolygonZ feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Light_Azimuth}	A Double representing the azimuth angle of the light source. The azimuth is expressed in positive degrees from 0 to 360, measured clockwise from the north. The default is 315 degrees.
{Light_Altitude}	A Double representing the altitude angle of the light source above the horizon. The altitude is expressed in positive degrees, with 0 degrees at the horizon and 90 degrees directly overhead. The default is 45 degrees.

Scripting syntax

ET_GPAnalyzeTIN (input_dataset out_feature_class Light_Azimuth Light_Altitude)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

AnalyzeTIN(pInFC As IFeatureClass, sOutFName As String,sElevationField As String, Optional dLightAzimuth As Double = 315, Optional dLightAltitude As Double = 45) As IFeatureClass

Copyright © Ianko Tchoukanski

ESRI TIN to PolygonZ

[Go to ToolBox Implementation](#)

Converts an ESRI TIN to PolygonZ feature class. Calculates several characteristics for each triangle.

- Minimum elevation
- Maximum elevation
- Mean elevation
- Slope - identifies the slope, or maximum rate of elevation change for each triangle
- Aspect - the values of the output field represent the compass direction of the aspect; 0 is true north, a 90 degree aspect is to the east etc. For flat triangles (slope = 0) the value of -1 is assigned for the aspect
- Hill Shade- computes the brightness of each triangle based on a light source location.

Inputs:

- A TIN in ESRI TIN format
- Characteristics to be calculated
 - Parameters for Hill Shade (if Hill Shade option is selected)
 - azimuth - the azimuth angle of the light source. The azimuth is expressed in positive degrees from 0 to 360, measured clockwise from the north. The default is 315 degrees.
 - altitude - the altitude angle of the light source above the horizon. The altitude is expressed in positive degrees, with 0 degrees at the horizon and 90 degrees directly overhead. The default is 45 degrees.

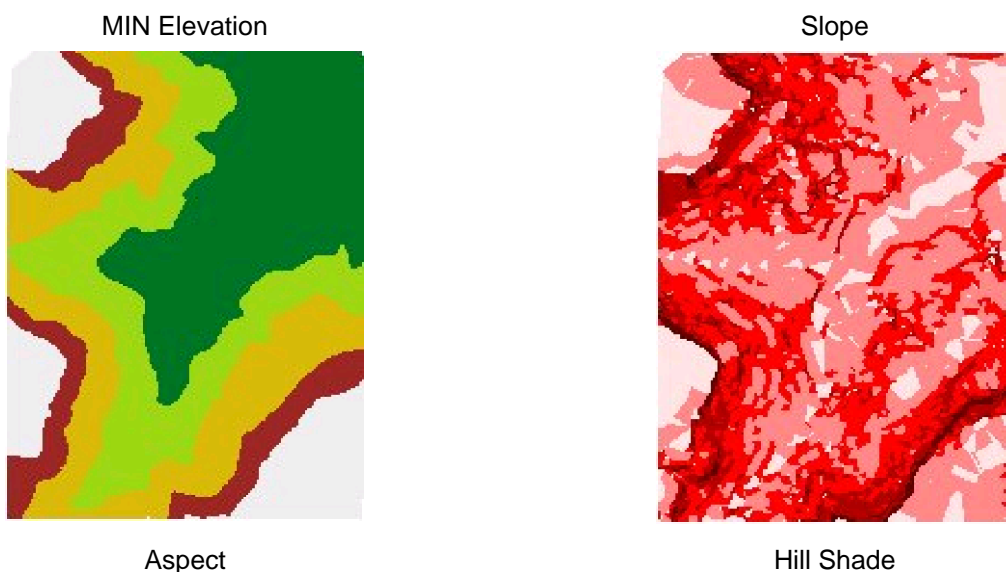
Outputs:

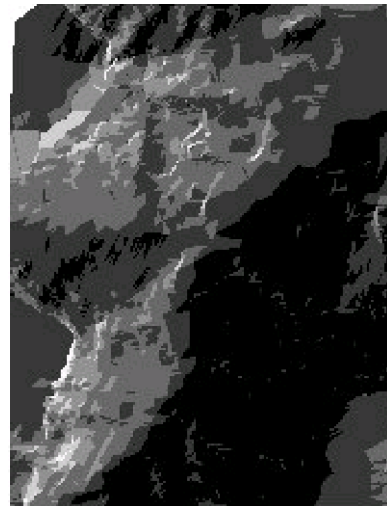
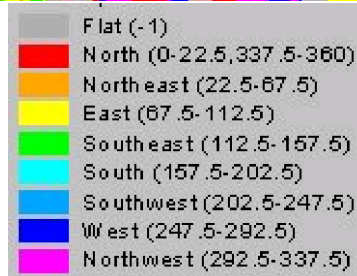
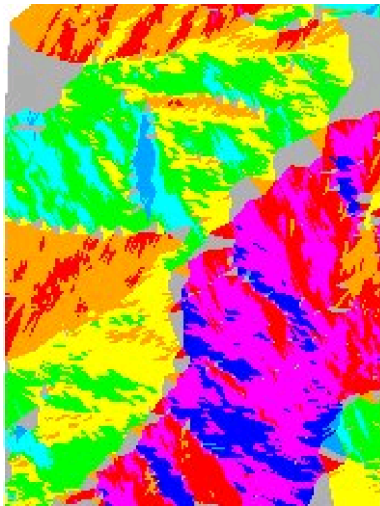
- New PolygonZ feature class. Several fields are added to the attribute table, depending on the options selected
 - [ET_EIMin] - minimum elevation values for each triangle
 - [ET_EIMax] - maximum elevation values for each triangle
 - [ET_EIMean] - mean elevation values for each triangle
 - [ET_Slope] - the slope of each triangle
 - [ET_Aspect] - the aspect of each triangle
 - [ET_Hill] - a Hill Shade value for each triangle depending on the input values for Azimuth and Altitude of the light source

Notes:

- A flat triangle (Slope = 0) will have an Aspect of -1

Example: TIN (PolygonZ) classified by





ToolBox implementation

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

Command line syntax

ET_GPESriTINToPolygonZ <input_TIN> <out_feature_class> <Light_Azimuth><Light_Altitude>

Parameters

Expression	Explanation
<input_TIN>	A TIN layer or dataset
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Light_Azimuth}	A Double representing the azimuth angle of the light source. The azimuth is expressed in positive degrees from 0 to 360, measured clockwise from the north. The default is 315 degrees.
{Light_Altitude}	A Double representing the altitude angle of the light source above the horizon. The altitude is expressed in positive degrees, with 0 degrees at the horizon and 90 degrees directly overhead. The default is 45 degrees.

Scripting syntax

ET_GPESriTINToPolygonZ (input_TIN out_feature_class Light_Azimuth Light_Altitude)

See the explanations above:

<> - required parameter

{ } - optional parameter

Features to 3D

Creates a 3D dataset from the features of existing 2D dataset (point, polyline or polygon) by deriving the values from a surface (Raster or TIN) layer.

No 3D Analyst needed!

Inputs:

- A feature layer (Point, Polygon, Polyline)
- A surface layer (Raster or TIN)

Outputs:

- New PointZ, PolylineZ or PolygonZ dataset (depending on the input).

Notes:

- The input feature layer and surface layer must have the same Spatial Reference
- The points/vertices that fall outside of the surface will get Z value of 999999 (NO DATA)
- For advanced surface functionality check ET Surface - <http://www.ian-ko.com>

Interpolate Contours

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Interpolates contours from a TIN (polygonZ)

Inputs:

- A TIN (polygonZ) feature layer
- Base value - the contour from which to begin generation of contours
- Contour interval - Z value difference between adjacent contours in map units.

Outputs:

- New polyline feature class. Several fields are added to the attribute table, depending on the options selected
 - [ET_Height] - the contour value

Notes:

- If flat triangles (see [TIN Notes](#)) are present, some small problems might occur in the contours. These problems are easy to identify using Export Nodes Wizard. Since the contour lines never intersect each other, there should not be Regular Nodes in the contour layer. A regular node in this case will indicate an error (most probably caused by a flat triangle). The excess polyline can be deleted.
- The [Smooth Polylines function](#) can be used to smooth the shape of interpolated contour
- The [Generalize Polylines function](#) can be used to remove the excess vertices

ToolBox implementation

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

Command line syntax

ET_GPInterpolateContours <input_dataset> <out_feature_class> <Base_Contour><Contour_Interval>

Parameters

Expression	Explanation
<input_dataset>	A PolygonZ feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Base_Contour>	A Double representing the the contour from which to begin generation of contours.
<Contour_Interval>	A Double representing the Z value difference between adjacent contours in the units of the TIN.

Scripting syntax

ET_GPInterpolateContours (input_dataset out_feature_class Base_Contour Contour_Interval)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

InterpolateContours(pInFC As IFeatureClass, sOutFName As String, dBase As Double, dInterval As Double, Optional bZContours As Boolean = False) As IFeatureClass

Interpolate Z for Points

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Interpolates Z values for the features from a point layer.

Inputs:

- A point feature class
- A TIN (PolygonZ) feature class. See [Interpolate TIN](#) function

Outputs:

- New point feature class. A field "ET_Z" is added to the attribute table and the interpolated Z values are stored in this field.

Notes:

- Source for the Z values should be a TIN (polygonZ) layer
- The Z values are stored in a new field - "ET_Z"
- Use Point To PointZ function to convert the points to a 3D feature class
- The spatial references of both input datasets must have the same Geographic Coordinate System

ToolBox implementation

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

Command line syntax

ET_GPInterpolateZForPoints <input_dataset> <Polygon_Z_TIN> <out_feature_class>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<Polygon_Z_TIN>	A PolygonZ TIN feature class
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPInterpolateZForPoints (input_dataset, Polygon_Z_TIN, out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

InterpolateZforPoints(pInFC As IFeatureClass, pTinFC As IFeatureClass, sOutFName As String) As IFeatureClass

Clip

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Clips a feature layer with the features of a polygon layer

Inputs:

- Layer to be clipped - a Point, Polyline or Polygon layer
- Clip layer - a polygon layer which features will be used for clipping

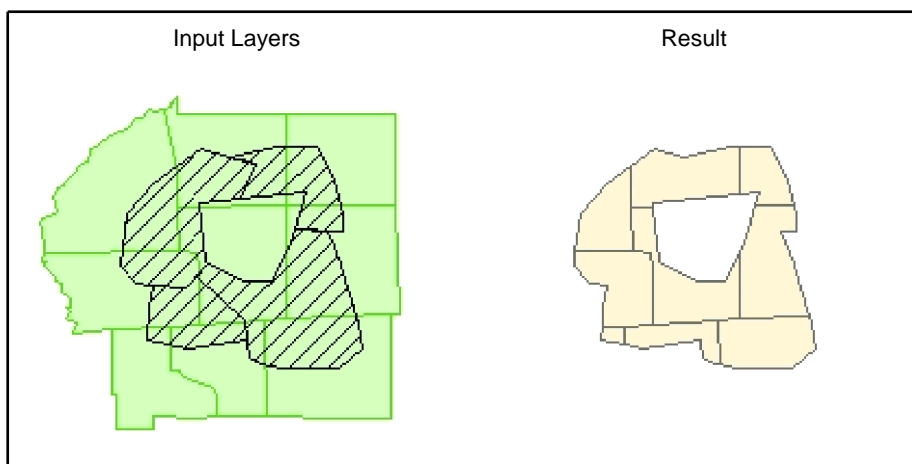
Outputs:

- New feature class (Point, 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

Notes:

- The function works very much like the Clip function of the Geo Processing Wizard, however it preserves the Spatial reference of the input data set. 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.

Examples:



ToolBox implementation

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

Command line syntax

ET_GPClip <input_dataset> <clip_dataset> <out_feature class> <fuzzy_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<clip_dataset>	A Polygon feature class or feature class. NOTE: The spatial references of <clip_dataset> and the <input_dataset> must have the same Geographic Coordinate System
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<fuzzy_tolerance>	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPClip (input_dataset, clip_dataset, out_feature class, fuzzy_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ClipSingle(pInFC As IFeatureClass, pClipFC As IFeatureClass, sOutFName As String, dFuzzy As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Batch Clip

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Clips a batch of feature layers with the features of a polygon layer

Inputs:

- Layers to be clipped - a Point, Polyline or Polygon layers
- Clip layer - a polygon layer which features will be used for clipping
- Workspace where the result feature classes will be stored

Outputs:

- New feature classes (Point, 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 feature classes will be named after the original layers with suffix "_clip". If the name exists the suffix will have a number at the end - "name_clip1", "name_clip2"...

ToolBox implementation

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

Command line syntax

ET_GPBatchClip <input_dataset;input_dataset;...> <clip_dataset> <out_workspace> {suffix} {fuzzy_tolerance}

Parameters

Expression	Explanation
<input_dataset;input_dataset;...>	A list of Point, Polyline or Polygon feature classes or feature layers
<clip_dataset>	A Polygon feature class or feature layer. NOTE: The spatial references of <erase_dataset> and all input_datasets must have the same Geographic Coordinate System
<out_workspace>	A String - the full name of the output feature workspace. Examples: <ul style="list-style-type: none">● "c:\00\test_pgdb.mdb" - for Personal Geodatabase● "c:\00\results" - for shapefiles
{suffix}	A String - used for generating the names of the output feature classes
{fuzzy_tolerance}	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPBatchClip (input_datasets, clip_dataset, out_workspace, suffix, fuzzy_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

BatchClip(pFCArray() As IFeatureClass, pClipFC As IFeatureClass, sOutWorkSpace As String, Optional sSuffix As String = "", Optional dFuzzy As Double = 0) As Boolean

Erase

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Erases a feature layer with the features of a polygon layer

Inputs:

- Layer to be erased - a Point, Polyline or Polygon layer
- Erase layer - a polygon layer which features will be used for erasing

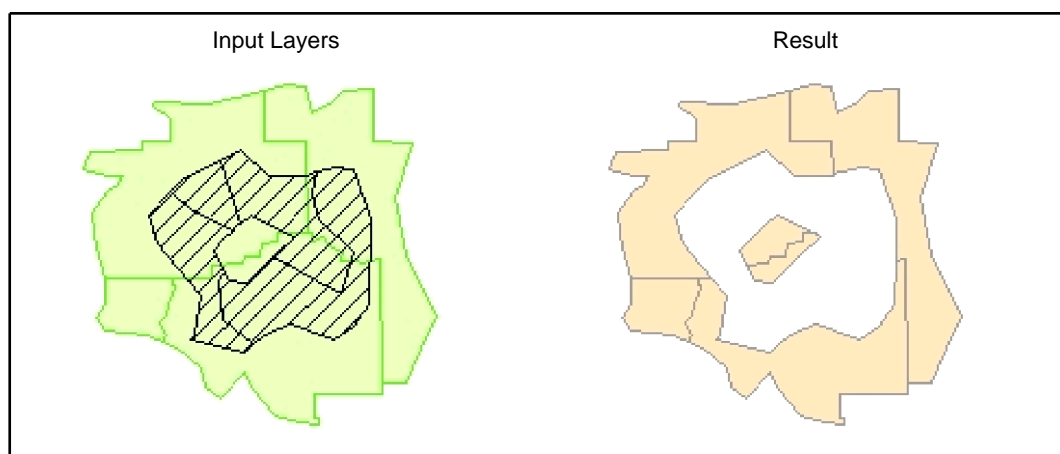
Outputs:

- New feature class (Point, 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

Notes:

- The function is the opposite of the Clip function.

Examples:



ToolBox implementation

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

Command line syntax

ET_GPERase <input_dataset> <erase_dataset> <out_feature class> <fuzzy_tolerance>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<erase_dataset>	A Polygon feature class or feature class. NOTE: The spatial references of <erase_dataset> and the <input_dataset> must have the same Geographic Coordinate System
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<fuzzy_tolerance>	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPErase (input_dataset, erase_dataset, out_feature class, fuzzy_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

EraseSingle(pInFC As IFeatureClass, pEraseFC As IFeatureClass, sOutFName As String, dFuzzy As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Batch Erase

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Erases a batch of feature layers with the features of a polygon layer

Inputs:

- Layers to be erased - a Point, Polyline or Polygon layers
- Erase layer - a polygon layer which features will be used for erasing
- Workspace where the result feature classes will be stored

Outputs:

- New feature classes (Point, 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 feature classes will be named after the original layers with suffix "_erase". If the name exists the suffix will have a number at the end - "name_erase1", "name_erase2"...

ToolBox implementation

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

Command line syntax

ET_GPBatchErase <input_dataset;input_dataset;...> <erase_dataset> <out_workspace> {suffix} {fuzzy_tolerance}

Parameters

Expression	Explanation
<input_dataset;input_dataset;...>	A list of Point, Polyline or Polygon feature classes or feature layers
<erase_dataset>	A Polygon feature class or feature layer. NOTE: The spatial references of <erase_dataset> and all input_datasets must have the same Geographic Coordinate System
<out_workspace>	A String - the full name of the output feature workspace. Examples: <ul style="list-style-type: none">● "c:\00\test_pgdb.mdb" - for Personal Geodatabase● "c:\00\results" - for shapefiles
{suffix}	A String - used for generating the names of the output feature classes
{fuzzy_tolerance}	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPBatchErase (input_datasets, erase_dataset, out_workspace, suffix, fuzzy_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

BatchErase(pFCArray() As IFeatureClass, pEraseFC As IFeatureClass, sOutWorkSpace As String, Optional sSuffix As String = "", Optional dFuzzy As Double = 0) As Boolean

Merge Feature Layers

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Merges feature layers from the same type together.

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)

Notes:

- The function works very much like the Merge Layers function in the standard GeoProcessing wizard, but preserves the Z/M dimension of the input layers (if the base layer has Z/M dimension.

ToolBox implementation

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

Command line syntax - Merge Multi

ET_GPMergeMulti <base_dataset> <Input_dataset ;Input_dataset...> <out_feature class>

Parameters - Merge Multi

Expression	Explanation
<base_dataset>	A feature class or feature layer. The fields of this dataset will be preserved in the output feature class.
<Input_dataset ;Input_dataset...>	A list of the datasets to be merged to the base dataset. Note: the feature classes should have the same type geometry and a spatial reference with the same geographic coordinate system as the base dataset.
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax - Merge Multi

ET_GPMergeMulti (base_dataset, input_datasets, out_feature class)

Command line syntax - Merge

ET_GPMerge<base_dataset> <merge_dataset > <out_feature class>

Parameters - Merge

Expression	Explanation
<base_dataset>	A feature class or feature layer. The fields of this dataset will be preserved in the output feature class.

<merge_dataset	A feature class or feature layer - the dataset to be merged to the base dataset - should have the same type geometry and a spatial reference with the same geographic coordinate system as the base dataset.
>	
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax - Merge

ET_GPMerge(base_dataset, merge_dataset , out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

MergeMulti(pInFC As IFeatureClass, pFCArray() As IFeatureClass, sOutFName As String) As IFeatureClass

MergeSingle(pInFC As IFeatureClass, pMergeFC As IFeatureClass, sOutFName As String) As IFeatureClass

Split By Location

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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, 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 feature classes
 - 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"

ToolBox implementation

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

Command line syntax

ET_GPSplitByLocation <input_dataset> <clip_dataset> <out_folder> <fuzzy_tolerance> <name_field> {prefix}

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<clip_dataset>	A Polygon feature class or feature layer. The features of this dataset will be used for clipping
<out_folder>	A String - the name of the output folder (must exist)
<fuzzy_tolerance>	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

<name_field>	A String - the name of a field from the clip dataset used for generating the names of the output names
{prefix}	A String - used for generating the names of the output feature classes

Scripting syntax

ET_GPSplitByLocation (input_dataset, clip_dataset, out_folder, fuzzy_tolerance, name_field, prefix)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SplitByLocation(pInFC As IFeatureClass, pClipFC As IFeatureClass, sOutWorkSpace As String, dFuzzy As Double, sField As String, Optional sPrefix As String = "") As Boolean

Copyright © Ianko Tchoukanski

Split By Attributes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Splits a layer into separate datasets based on an the attribute values in the selected field

Inputs:

- Layer to be split - a Point, Polyline or Polygon layer
- Output folder
- Name field - a field from the input 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 feature classes
 - The attributes are preserved
 - The spatial reference of the input data set is preserved

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"

ToolBox implementation

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

Command line syntax

ET_GPSplitByAttributes <input_dataset> <out_folder> <name_field> {prefix}

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_folder>	A String - the name of the output folder (must exist)
<name_field>	A String - the name of a field from the input layer used for splitting and for generating the names of the output names
{prefix}	A String - used for generating the names of the output feature classes

Scripting syntax

ET_GPSplitByAttributes (input_dataset, out_folder, name_field, prefix)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SplitByAttributes(pInFC As IFeatureClass, sOutWorkSpace As String, sField As String, Optional sPrefix As String = "") As Boolean

Copyright © Ianko Tchoukanski

Transfer Polygon Attributes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

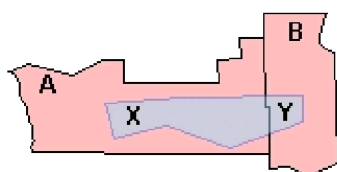
Inputs:

- 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 feature class.
 - 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.)

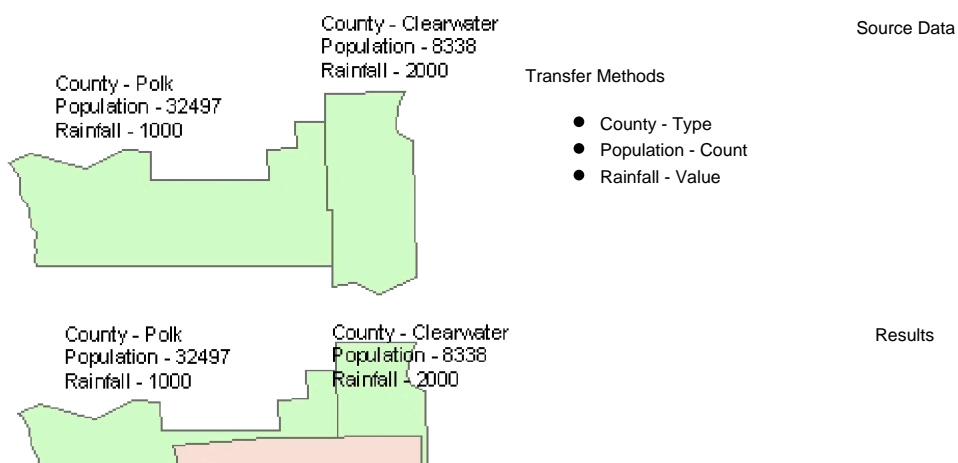
IF $\text{area_X} / \text{area_Z} > \text{area_Y} / \text{area_Z}$ THEN $\text{soiltype_Z} = \text{soiltype_A}$

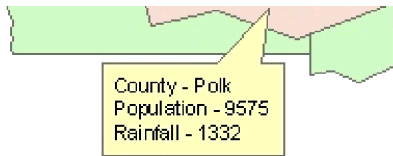
IF $\text{area_X} / \text{area_Z} < \text{area_Y} / \text{area_Z}$ THEN $\text{soiltype_Z} = \text{soiltype_B}$

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:





ToolBox implementation

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

Command line syntax

ET_GPTTransferAttributes <target_dataset> <source_dataset> <out_feature_class> <Field {Transfer Method};Field {Transfer Method}...>

Parameters

Expression	Explanation
<target_dataset>	A Polygon feature class or feature layer
<source_dataset>	A Polygon feature class or feature layer.
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Field {Transfer Method};Field {Transfer Method}...>	A list of the fields to be transferred and the transfer method for each field.

Scripting syntax

ET_GPTTransferAttributes (target_dataset, source_dataset, out_feature_class, fields to transfer)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

TransferAttributes(pTargetFC As IFeatureClass, pSourceFC As IFeatureClass, sOutFName As String, transferDic As Dictionary(Of String, String)) As IFeatureClass

Remove Exact Duplicates

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Removes duplicates with exactly the same shape from a Point, Polyline or Polygon dataset.

Inputs:

- A feature layer (Point, Polygon, Polyline)
- OPTIONAL: A reference layer (Should have the same shape type as the input layer)

Outputs:

- New Point, 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

ToolBox implementation

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

Command line syntax

ET_GPRemoveDuplicates<input_dataset> <Reference_dataset> <out_feature_class>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
{Reference_dataset}	A Polyline or Polygon feature class or feature class. NOTE: The shape type of {Reference_dataset} and the <input_dataset> must be the same.
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPRemoveDuplicates (input_dataset, Reference_dataset, out_feature_ class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

RemoveDuplicates(plnFC As IFeatureClass, sOutFName As String, Optional pRefFc As IFeatureClass = Nothing) As IFeatureClass

Symmetrical Difference

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

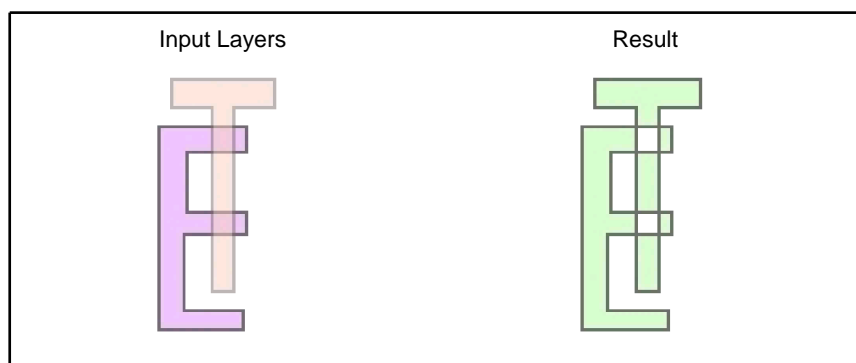
Inputs:

- Two polygon feature classes
- Fuzzy tolerance. Keep the fuzzy tolerance as small as possible to avoid unwanted approximation of the shapes.

Outputs:

- New polygon feature class

Examples:



ToolBox implementation

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

ET_GPSymmetricalDifference<first_dataset> <second_dataset> <out_feature class> <fuzzy_tolerance>

Parameters

Expression	Explanation
<first_dataset>	A Polygon feature class or feature layer
<second_dataset>	A Polygon feature class or feature class. NOTE: The spatial references of <first_dataset> and the <second_dataset> must have the same Geographic Coordinate System
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<fuzzy_tolerance>	A Double representing the Fuzzy tolerance (in the units of the input dataset) to be used

Scripting syntax

ET_GPSymmetricalDifference (first_dataset, second_dataset, out_feature_class, fuzzy_tolerance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SymmetricalDifference(pInFC As IFeatureClass, pRefFc As IFeatureClass, sOutFName As String, dFuzzy As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Spatial Join

This function is not available via the GUI of ET GeoWizards.

Joins the attributes of feature classes based on the spatial relationships of the features.

Command line syntax

```
ET_GPSSpatialJoin <input_dataset> <join_dataset> <out_feature class> <Nearest | Within | Intersects | Contains> {left_outer_join} {search_tolerance}
```

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<join_dataset>	A Point, Polyline or Polygon feature class or feature layer. The features of this dataset will be used for clipping
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Nearest Within Intersects Contains>	Join Type. A String - the join type to be used. The available options are: <ul style="list-style-type: none">● "Nearest" - Joins with the nearest feature in the join feature class. Only features within a distance of {search_tolerance} will be joined. A {search_tolerance} of -1 means infinity.● "Within" - Joins if a feature from the input dataset is within a feature of the join dataset.● "Intersects" - Joins if a feature from the input dataset intersects a feature of the join dataset.● "Contains" Joins if a feature from the input dataset contains a feature of the join dataset.
{left_outer_join}	A Boolean - Indicates whether a match is required before adding a record from the source feature class to the result. If True, all records in the Source feature class are added regardless of whether there is a match.
{search_tolerance}	A Double representing the search tolerance

Scripting syntax

```
ET_GPSSpatialJoin (input_dataset, join_dataset, out_feature class, join_type, left_outer_join, search_tolerance)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

SpatialJoin(pInFC As IFeatureClass, pJoinFC As IFeatureClass, sOutFName As String, sJoinType As String,
bLeftOuter As Boolean, dSearchTol As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Create feature class

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates new feature class with user defined type, spatial reference and fields.

Inputs:

- Type of the feature class to be created
 - Point
 - Polyline
 - Polygon
- Options for additional dimension
 - Z - if the feature class will contain shapes with Z values
 - M - if the feature class will contain shapes with M values
- Spatial Reference (can be copied from the Data Frame or from any feature layer loaded in the map)
- Attribute fields definitions (added in ET GeoWizards 9.1).

Outputs:

- An empty feature class (Point, Polyline or Polygon depending on the selected type)

Notes:

- The function works very much like the ArcCatalog function New ==> feature class , but from ArcMap

ToolBox implementation

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

Command line syntax

ET_GPCreateFeatureClass<geometry_type> <out_feature_class>{out_spatial_reference}{Z} {M}

Parameters

Expression	Explanation
<geometry_type>	A String defining the type of geometry of the new feature class - "Point", "Polyline", "Polygon", "Multipoint"
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{out_spatial_reference}	A String - the spatial reference of the dataset to be created
{Z}	A Boolean - indicating whether the output will be Z enabled
{M}	A Boolean - indicating whether the output will be M enabled

Scripting syntax

ET_GPCreateFeatureClass(geometry_type, out_feature_class,out_spatial_reference, Z, M)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CreateFeatureClass(sOutFName As String, sGeometryType As String, pOutputSRef As ISpatialReference, Optional bZ As

Boolean = False, Optional bM As Boolean = False) As IFeatureClass

Copyright © Ianko Tchoukanski

Sort Shapes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Sorts the features of a feature layer according to user specified fields and order methods.

Inputs:

- A feature layer
 - Point
 - 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

ToolBox implementation

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

Command line syntax

ET_GPSortShapes <input_dataset> <out_feature class> <Field {Sort Method};Field {Sort Method}...>

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Field {Sort Method};Field {Sort Method}...>	A list of the fields to be transferred and the sort method for each field. Note: If the Sort Method string is not "Ascending" or "Descending" (case sensitive) a descending method will be used.

Scripting syntax

ET_GPSSortShapes (input_dataset, out_feature class, sort fields)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

SortShapes(pInFC As IFeatureClass, sOutFName As String, sortDictionary As Dictionary(Of String, String)) As IFeatureClass

Copyright © Ianko Tchoukanski

Move Shapes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Moves the features of a feature layer according to user specified translation vector .

-

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
- Input type - a way the translation vector will be defined
 - From X,Y & To X,Y
 - dX & dY
 - From Point & To Point from a point layer
- Move parameters - parameters defining the translation vector depending on the selected input type
 - If the selected input type is "From & To Points from a layer" a point layer with only two points in it have to be selected

Outputs:

- New feature class
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Notes:

- There is a standard ArcGIS Move function, but the Move Shapes Wizard gives more control over the operation, especially if multiple layers are to be moved.
- The standard ArcGIS Move function may be convenient for moving individual features, but it is very slow when many features (data set) are to be moved.
- If the From & To Points from a layer" is to be used
 - Use [New Feature Class function](#) to create new point feature class
 - Start editing
 - Input From Point (snap can be used to place the point exactly at desired location)
 - Input To Point
 - Stop editing
 - Use this layer in the Move Shapes Wizard

ToolBox implementation

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

Command line syntax

ET_GPMoveShapes<input_dataset> <out_feature_class> {from_X} {from_Y} {to_X} {to_Y} {from_to_point_dataset} {Delta_X} {Delta_Y}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{from_X}	A Double - X coordinate of the origin point.

{from_Y}	A Double - Y coordinate of the origin point.
{to_X}	A Double - X coordinate of the destination point.
{to_Y}	A Double - Y coordinate of the destination point.
{from_to_point_dataset}	A Point feature class or feature layer. It must have at least two points - the first point will be used as origin and the second point will be used as destination.
{Delta_X}	A Double - Movement in X direction
{Delta_Y}	A Double - Movement in Y direction

Scripting syntax

ET_GPMoveShapes(input_dataset, out_feature_class,from_X,from_Y, to_X, to_Y)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\moved"
arcpy.ET_GPMoveShapes(input_dataset, result, 0.00, 0.00,1.00,1.00)
```

.NET implementation

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

MoveShapes(pInFC As IFeatureClass, sOutFName As String, dDeltaX As Double, dDeltaY As Double) As IFeatureClass

Rotate Shapes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Rotates the features of a feature layer according to user specified rotation point and angle .

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
- Input type - a way the rotation point will be defined
 - input X,Y
 - point from a feature layer
- Rotation angle - angle in decimal degrees, positive values will rotate the features counterclockwise and negative - clockwise

Outputs:

- New feature class
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Notes:

- There is a standard ArcGIS Rotate tool, but the Rotate Shapes Wizard gives more control over the operation, especially if multiple layers are to be moved
- If rotation point from a point layer is to be used
 - Use [New Feature Class function](#) to create new point feature class
 - Start editing
 - Input Origin Point (snap can be used to place the point exactly at desired location)
 - Stop editing
 - Use this layer in the Rotate Shapes Wizard

ToolBox implementation

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

Command line syntax

```
ET_GPRotateShapes<input_dataset> <out_feature_class><input_dataset> <out_feature_class> {rotation_angle}  
{origin_point_dataset} {Origin_X} {Origin_Y}
```

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{rotation_angle}	A Double - Rotation angle in Degrees - positive values will rotate the features counterclockwise and negative - clockwise
{origin_point_dataset}	A Point feature class or feature layer. It must have at least one point to be used as origin of rotation.
{Origin_X}	A Double - X of the origin point to be used for rotation
{Origin_Y}	A Double - Y of the origin point to be used for rotation

Scripting syntax

ET_GPRotateShapes(input_dataset, out_feature_class, rotation_angle, "", Origin_X, Origin_Y)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\rotated"
arcpy.ET_GPRotateShapes(input_dataset, result, 45, "", 0.00, 0.00)
```

.NET implementation

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

RotateShapes(pInFC As IFeatureClass, sOutFName As String, dOriginX As Double, dOriginY As Double, dRotationAngle As Double) As IFeatureClass

Scale Shapes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Scales the features of a feature layer according to user specified anchor point X and Y scale factors .

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
- Input type - a way the anchor point will be defined
 - input X,Y
 - point from a feature layer
- X scale factor
- Y scale factor

Outputs:

- New feature class
 - The attributes are preserved
 - The spatial reference of the input layer is preserved

Notes:

- There is a standard ArcGIS Scale tool, but the Scale Shapes Wizard gives more control over the operation, especially if multiple layers are to be moved
- If anchor point from a point layer is to be used
 - Use the [New Feature Class function](#) to create new point feature class
 - Start editing
 - Input Origin Point (snap can be used to place the point exactly at desired location)
 - Stop editing
 - Use this layer in the Scale Shapes Wizard

ToolBox implementation

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

Command line syntax

ET_GPSScaleShapes<input_dataset> <out_feature_class> {Scale_X} {Scale_Y} {origin_point_dataset} {Origin_X} {Origin_Y}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Scale_X}	A Double - defining the scale factor in X direction
{Scale_Y}	A Double - defining the scale factor in Y direction
{origin_point_dataset}	A Point feature class or feature layer. It must have at least one point to be used as origin for scaling.
{Origin_X}	A Double - X of the origin point to be used for rotation
{Origin_Y}	A Double - Y of the origin point to be used for rotation

Scripting syntax

ET_GPSScaleShapes(input_dataset, out_feature_class, Scale_X, Scale_Y, "", Origin_X, Origin_Y)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
input_dataset = "C:\\data\\pg1.shp"
result = "C:\\data\\fgdb_test.gdb\\scaled"
arcpy.ET_GPSScaleShapes(input_dataset, result, 2.0, 1.0, "", 0.00, 0.00)
```

.NET implementation

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

ScaleShapes(pInFC As IFeatureClass, sOutFName As String, dOriginX As Double, dOriginY As Double, dScaleX As Double, dScaleY As Double) As IFeatureClass

Explode

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
- Update rules for the numeric fields to be transferred.

Outputs:

- A feature class with no multipart shapes present.
- The numeric attributes will be transferred according the user specified rules. The rest of the attributes will be copied over

Notes:

- To select an update rule for each numeric field - click on the appropriate cell in the field list

ToolBox implementation

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

Command line syntax

ET_GPExplode <input_dataset> <out_feature class> {Update_Rules_List}

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Update_Rules_List}	A String - a list of fields with their update rules.

Scripting syntax

ET_GPExplode (input_dataset, out_feature class, Update_Rules_List)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET GeoWizards.tbx"
input_dataset = "C:\\data\\suburbs.shp"
result = "C:\\data\\fgdb_test.gdb\\single_part"
arcpy.gp.ET_GPExplode (input_dataset, result, "Population Proportion; City Copy")
```

.NET implementation

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

ExplodeMultiPart(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Copyright © Ianko Tchoukanski

Closest Feature Distance

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 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 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.

ToolBox implementation

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

Command line syntax

ET_GPClosestFeatureDistance<input_dataset> <out_feature_class> <search_distance>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<search_distance>	A Double representing the maximum distance between the points within a cluster - in the units of the spatial reference of the input dataset

Scripting syntax

ET_GPClosestFeatureDistance(input_dataset,out_feature class, search_distance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

ClosestFeatureDistance(pInFC As IFeatureClass, sOutFName As String, dSearchTol As Double) As IFeatureClass

Copyright © Ianko Tchoukanski

Fix Geometry

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Exports the features of the input feature dataset to a new feature class. Removes the null and empty shapes. Fixes some geometric inconsistencies of the geometries.

Inputs:

- A Point, Polyline or Polygon feature layer

Outputs:

- New feature class - no null or empty shapes present

ToolBox implementation

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

Command line syntax

ET_GPFixGeometry <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Point, Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPFixGeometry (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FixGeometry(plnFC As IFeatureClass, sOutFName As String) As IFeatureClass

Delete Multiple Fields

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Deletes multiple fields from a layer. If the source of the layer is a feature class, the operation can be performed directly on the source. If the source is a coverage or geodatabase, new feature class is the only option

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
 - Multipoint
- Option to create a new feature class or to delete the fields in the input feature class.
- Fields to be deleted

Outputs:

- New feature class - if the option is selected and output specified.

ToolBox implementation

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

Command line syntax

ET_GPDeleteMultipleFields <input_dataset> {out_feature class} <Field;Field...>

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer
{out_feature class}	A String - the full name of the output feature class (A feature class with the same full name should not exist). If not specified the fields will be deleted from the input dataset.
<Field ;Field...>	A list of the fields to be be deleted.

Scripting syntax

ET_GPDeleteMultipleFields (input_dataset, out_feature_class, fields)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

DeleteMultipleFields(pInFC As IFeatureClass, fieldList As List(Of String), Optional sOutFName As String = "") As IFeatureClass

Order Fields

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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
 - 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

ToolBox implementation

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

Command line syntax

ET_GPOrderFields <input_dataset> <out_feature class> <Field;Field...>

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<Field ;Field...>	A list of the fields to be used. The order of the fields in the list will be the order of the fields in the attribute table of the output feature class. Note: only the fields in the list will be available in the output feature class

Scripting syntax

ET_GPOrderFields (input_dataset, out_feature class, ordered fields)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

OrderFields(pInFC As IFeatureClass, sOutFName As String, ByVal fieldList As List(Of String)) As IFeatureClass

Redefine Fields

[Go to ToolBox Implementation](#)

Change field names and definitions.

Inputs:

- A feature layer
 - Point
 - Polyline
 - Polygon
- New field names, length, precision, scale

Outputs:

- A new feature class.

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't be changed
- The new field names should be max 10 characters long
- If the source layer is a Personal Geodatabase layer the numeric fields will be reported incorrectly in the grid with Precision = 0 and Scale = 0.
- If the Precision and Scale are incorrectly set to 0, the original field definition will be used.

ToolBox implementation

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

Command line syntax

ET_GPRedefineFields<input_dataset> <out_feature_class> <field_to_redefine>

Parameters

Expression	Explanation
<input_dataset>	A Polyline feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<field_to_redefine>	A String - a list of field names and new field definitions "OldName NewName Length/Precision Scale"

Scripting syntax

ET_GPRedefineFields(input_dataset, out_feature_class, field_to_redefine)

See the explanations above:

<> - required parameter

{ } - optional parameter

Example Python script:

```
import arcpy
arcpy.ImportToolbox("C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET
GeoWizards.tbx")
arcpy.gp.toolbox = "C:/Program Files/ET SpatialTechniques/ET GeoWizards 11.0 for ArcGIS 10.1/ET
GeoWizards.tbx"
input_dataset = "C:\\data\\suburbs.shp"
result = "C:\\data\\fgdb_test.gdb\\redefined"
arcpy.gp.ET_GPRedefineFields(input_dataset, result, "Population Pop 12 0; City CityNew 6 0; Meters
Length 12 2")
```

Copyright © Ianko Tchoukanski

Copy Fields

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Copies fields from one dataset to another.

Inputs:

- Target feature layer
- Source feature layer - the layer that will be used as a source for the field definitions

Outputs:

- The fields selected will be copied to the attribute table of the target layer

How to use:

- Select a target and source 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 already existing in the target attribute table
- If the target layer has a feature class source and the fields with long names (if present) will not be added to the target
- If the target attribute table has a field with the same name as a field selected to be copied, the field will not be added to the target

ToolBox implementation

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

Command line syntax

ET_GPCopyFields <target_dataset> <source_dataset> <Field;Field...>

Parameters

Expression	Explanation
<target_dataset>	A feature class or feature layer
<source_dataset>	A feature class or feature layer
<Field ;Field...>	A list of the fields to be copied from the source to the target dataset.

Scripting syntax

ET_GPCopyFields (input_dataset, source_dataset, fields_to_copy)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CopyFields(pTargetFC As IFeatureClass, pSourceFC As IFeatureClass, fieldList As List(Of String)) As Boolean

Rename Field

Changes the name of an existing field in the attribute table.

Command line syntax

ET_GPRenameField <input_dataset> <old_name> <new_name>

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer.
<old_name>	A String - the name of the field to be changed. Note. Some fields can not be renamed (Shape, OID, FID etc.)
<new_name>	A String - the new name of the field. The new field name should not duplicate the name of an existing field. It should not contain special charecters (space, "%", "&" etc.)

Scripting syntax

ET_GPRenameField (input_dataset,old_name, new_name)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

RenameField(pInFC As IFeatureClass, sOldName As String, sNewName As String) As Boolean

Add Attribute Index

This function is not available via the GUI of ET GeoWizards.

Adds attribute index to a feature class. An index can improve the performance of queries that evaluate an attribute's values.

Note: Can be used only on feature classes!

Command line syntax

ET_GPAttributeIndex <input_dataset> <field_to_index> {unique} {ascending}

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer (the source should be a feature class)
<field_to_index>	A String - the name of the field that will be indexed
{unique}	A Boolean - Indicates if the index is unique.
{ascending}	A Boolean - Indicates if the index is based on ascending order

Scripting syntax

ET_GPAttributeIndex (input_dataset, field_to_index, unique, ascending)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

Public Function AddAttributeIndex(ByVal pInFC As IFeatureClass, ByVal sFieldToIndex As String, Optional ByVal bUnique As Boolean = True, Optional ByVal bAscending As Boolean = True) As Boolean

Add Spatial Index

This function is not available via the GUI of ET GeoWizards.

Adds a spatial index to a feature class. Having a current spatial index ensures that a high level of performance is maintained when drawing and working with the feature class's features.

Note: Can be used only on feature classes!

Command line syntax

ET_GPSSpatialIndex <input_dataset>

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer (the source should be a feature class)

Scripting syntax

ET_GPSSpatialIndex (input_dataset)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

AddSpatialIndex(pInFC As IFeatureClass) As Boolean

Calculate Area

This function is not available via the GUI of ET GeoWizards.

Calculates the area of polygons. Please see the explanations below

Command line syntax

ET_GPCalculateArea <input_dataset> {calc_spatial_reference} { | Square Meters | Square Kilometers | Acres | Square Miles | Hectares | Square Yards | Square Feet} {Area_field}

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer.
{calc_spatial_reference}	The spatial reference in which the calculations will be performed. If not specified the spatial reference of the input dataset will be used. NOTE: The spatial reference of the input dataset and the Calculations spatial reference must have the same Geographic Coordinate System
{ Square Meters Square Kilometers Acres Square Miles Hectares Square Yards Square Feet}	Output Units. A String that defines the units in which the area will be calculated. NOTE: If the the Calculations spatial reference is a Geographic Coordinate System, this parameter will be ignored and the results will be calculated in decimal degrees!!!
{Area_field}	Area Field. A String indicating an existing field in which the results for the area will be stored. If not specified a new field "ET_Area" will be created

Scripting syntax

ET_GPCalculateArea (input_dataset, calc_spatial_reference, output_units, Area_field)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

CalculateArea(pInFC As IFeatureClass, Optional pSpatialReference As ISpatialReference = Nothing, Optional sAreaUnits As String = "", Optional sAreaField As String = "") As Boolean

Calculate Length

This function is not available via the GUI of ET GeoWizards.

Calculates the length of polylines or the perimeter of polygons. Please see the explanations below

Command line syntax

```
ET_GPCalculateLength <input_dataset> {calc_spatial_reference} { | Meters | Kilometers | Feet | Miles | Yards  
| Inches | Centimeters | Millimeters} {Length_field}
```

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer.
{calc_spatial_reference}	The spatial reference in which the calculations will be performed. If not specified the spatial reference of the input dataset will be used. NOTE: The spatial reference of the input dataset and the Calculations spatial reference must have the same Geographic Coordinate System
{ Meters Kilometers Feet Miles Yards Inches Centimeters Millimeters}	Output Units. A String that defines the units in which the length will be calculated. NOTE: If the the Calculations spatial reference is a Geographic Coordinate System, this parameter will be ignored and the results will be calculated in decimal degrees!!!
{Length_field}	Length Field. A String indicating an existing field in which the results for the length will be stored. If not specified a new field "ET_Length" will be created

Scripting syntax

```
ET_GPCalculateLength (input_dataset, calc_spatial_reference, output_units, Length_field)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

```
CalculateLength(pInFC As IFeatureClass, Optional pSpatialReference As ISpatialReference = Nothing,  
Optional sDistanceUnits As String = "", Optional sLengthField As String = "") As Boolean
```

Calculate

This function is not available via the GUI of ET GeoWizards.

Calculates some spatial characteristics of the shapes. The results are added in new fields in the attribute table of the original dataset.

Note: For more flexible Area or Length calculations see Calculate Area and Calculate Length tools

Command line syntax

```
ET_GPCalculate <input_dataset> <Area | Length | XYStart | XYEnd | XYMiddle | XYCenter | XYLabel | XY | Z  
| M>
```

Parameters

Expression	Explanation
<input_dataset>	A feature class or feature layer.
<Area Length XYStart XYEnd XYMiddle XYCenter XYLabel XY Z M>	<p>Calculate Task. A String - defines the calculation to be performed</p> <ul style="list-style-type: none">● Area - the area of polygons in the units of the spatial reference of the dataset - Polygon datasets● Length - the length of polylines or the perimeter of polygons in the units of the spatial reference of the dataset - Polygon or Polyline datasets/● XYStart - the coordinates of the start point of polylines or polygons - Polygon or Polyline datasets.● XYEnd - the coordinates of the end point of polylines or polygons - Polygon or Polyline datasets.● XYMiddle - the coordinates of the middle point of polylines - Polyline datasets.● XYCenter - the coordinates of the centroid of polygons - Polygon datasets.● XYLabel - the coordinates of the label points of polygons - Polygon datasets.● XY - the coordinates of points - Point dataset● Z - the Z value of points - PointZ dataset● M - the M value of points - PointZ dataset

Scripting syntax

```
ET_GPCalculate (input_dataset, calc_task)
```

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

```
CalculateValues(plnFC As IFeatureClass, sTask As String) As Boolean
```


Point Coordinates

This function is not available via the GUI of ET GeoWizards.

Calculates the coordinates of the points from a point dataset in the units of the spatial reference of the dataset, Decimal Degrees or Degrees Minutes Seconds. Two new fields are added to the attribute table. The names depending on the output units are:

- If the calculation is in the units of the spatial reference of the dataset
 - ET_X
 - ET_Y
- If the calculation is in Decimal Degrees or Degrees Minutes Seconds
 - ET_Lat
 - ET_Lon

If the input dataset is projected and the selected output units are Decimal Degrees or Degrees Minutes Seconds, the shapes are projected on the fly to the Geographic Coordinate System of the projection of the input dataset before calculating the coordinates.

Command line syntax

ET_GPPointCoordinates <input_dataset> < | Dataset Units | Decimal Degrees | Degrees Minutes Seconds> {precision} {use_direction}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer.
< Dataset Units Decimal Degrees Degrees Minutes Seconds>	Output Units. The units to be used for the calculations.
{precision}	An Integer between 0 and 8 representing the number of places after the decimal point to be used.
{use_direction}	A Boolean. If True - a direction for coordinates in Degrees Minutes Seconds (35°25'34.23"W), If False - (-35°25'34.23")

Scripting syntax

ET_GPPointCoordinates (input_dataset, output_units, precision, use_direction)

See the explanations above:

<> - required parameter

{ } - optional parameter

Polygon Coordinates

This function is not available via the GUI of ET GeoWizards.

Calculates the coordinates of the centroid or the label points of the polygons from a polygon dataset in the units of the spatial reference of the dataset, Decimal Degrees or Degrees Minutes Seconds.. Two new fields are added to the attribute table. The names depending on the Calculate option and the output units and the are:

- If the calculation is in the units of the spatial reference of the dataset
 - Center_X or Label_X
 - Center_Y or Label_Y
- If the calculation is in Decimal Degrees or Degrees Minutes Seconds
 - Center_Lat or Label_Lat
 - Center_Lon or Label_Lon

If the input dataset is projected and the selected output units are Decimal Degrees or Degrees Minutes Seconds, the shapes are projected on the fly to the Geographic Coordinate System of the projection of the input dataset before calculating the coordinates.

Command line syntax

ET_GPPolygonCoordinates <input_dataset> < | Center | Label> < | Dataset Units | Decimal Degrees | Degrees Minutes Seconds> {precision} {use_direction}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer.
< Center Label>	Calculate Option. A String indicating what coordinates will be calculated - the Centroid or the Label point
< Dataset Units Decimal Degrees Degrees Minutes Seconds>	Output Units. The units to be used for the calculations.
{precision}	An Integer between 0 and 8 representing the number of places after the decimal point to be used.
{use_direction}	A Boolean. If True - a direction for coordinates in Degrees Minutes Seconds (35°25'34.23"W), If False - (-35°25'34.23")

Scripting syntax

ET_GPPolygonCoordinates (input_dataset, calculate_option, output_units, precision, use_direction)

See the explanations above:

<> - required parameter

{ } - optional parameter

Copyright © Ianko Tchoukanski

Polyline Coordinates

This function is not available via the GUI of ET GeoWizards.

Calculates the coordinates of points at user specified position along the polylines of a polyline dataset in the units of the spatial reference of the dataset, Decimal Degrees or Degrees Minutes Seconds. Two new fields are added to the attribute table. The names depending on the Point Position and the output units and the are:

- If the calculation is in the units of the spatial reference of the dataset
 - Point0_X, Point50_X, ...
 - Point0_Y or Point50_Y
- If the calculation is in Decimal Degrees or Degrees Minutes Seconds
 - Point0_Lat or Point50_Lat
 - Point0_Lon or Point50_Lon

If the input dataset is projected and the selected output units are Decimal Degrees or Degrees Minutes Seconds, the shapes are projected on the fly to the Geographic Coordinate System of the projection of the input dataset before calculating the coordinates.

Command line syntax

ET_GPPolylineCoordinates <input_dataset> <point_position> < | Dataset Units | Decimal Degrees | Degrees Minutes Seconds> {precision} {use_direction}

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer.
<point_position>	A Double between 0 and 1 indicating the position of the point along the polyline which coordinates will be calculated. 0 indicates the Start Point, 1 - End Point, 0.5 - Middle point, 0.1 - a point for which the distance from the start point along the polyline is 10% of the total length of the polyline
< Dataset Units Decimal Degrees Degrees Minutes Seconds>	Output Units. The units to be used for the calculations.
{precision}	An Integer between 0 and 8 representing the number of places after the decimal point to be used.
{use_direction}	A Boolean. If True - a direction for coordinates in Degrees Minutes Seconds (35°25'34.23"W), If False - (-35°25'34.23")

Scripting syntax

ET_GPPolylineCoordinates (input_dataset, point_position, output_units, precision, use_direction)

See the explanations above:

<> - required parameter

{ } - optional parameter

Copyright © Ianko Tchoukanski

ET GeoWizards Linear Referencing

Linear Referencing (or Dynamic Segmentation) is a very important GIS feature. In many cases (road, river management systems etc.) the data is not represented with X & Y coordinate pairs, but rather in one dimensional linear referencing system. ArcGIS has a comprehensive set of tools for creating, displaying and analyzing such data. Most of these tools however are available only to the users with ArcEditor or ArcInfo licenses.

ET GeoWizards 9.3 introduces a new set of functions that enables the ArcView license holders to perform tasks standard only in the top licensing options of ArcGIS

Functions Available:

ET GeoWizards Functions	Standard ArcGIS Tools
available with any ArcGIS license	available in
Create routes from existing polylines	ArcEditor and ArcInfo
Calibrate Routes with points	ArcEditor and ArcInfo
Locate point features along routes	ArcInfo
Locate polygon features along routes	ArcInfo
Dissolve Route Events	ArcInfo
Concatenate Route Events	ArcInfo
Intersect Route Events	ArcInfo
Union Route Events	ArcInfo

Since the ET GeoWizards functions work in a very similar fashion to the standard tools we strongly recommend our users to read the "Linear Referencing in ArcGIS" book , provided by ESRI on the ArcGIS instillation media in pdf format. The book gives a good overview of the Linear Referencing and discusses in detail the options available.

Create routes from existing polylines

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)

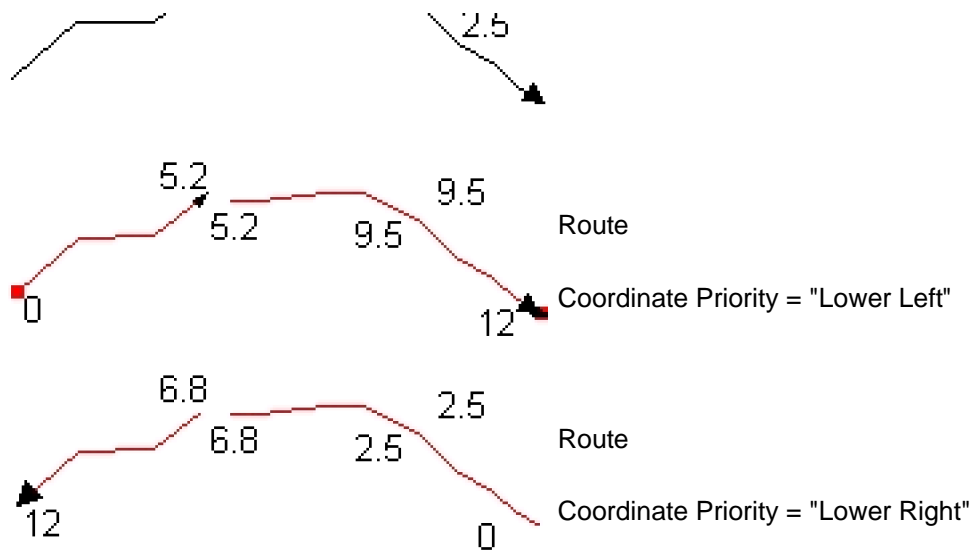
Measuring Methods

- 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

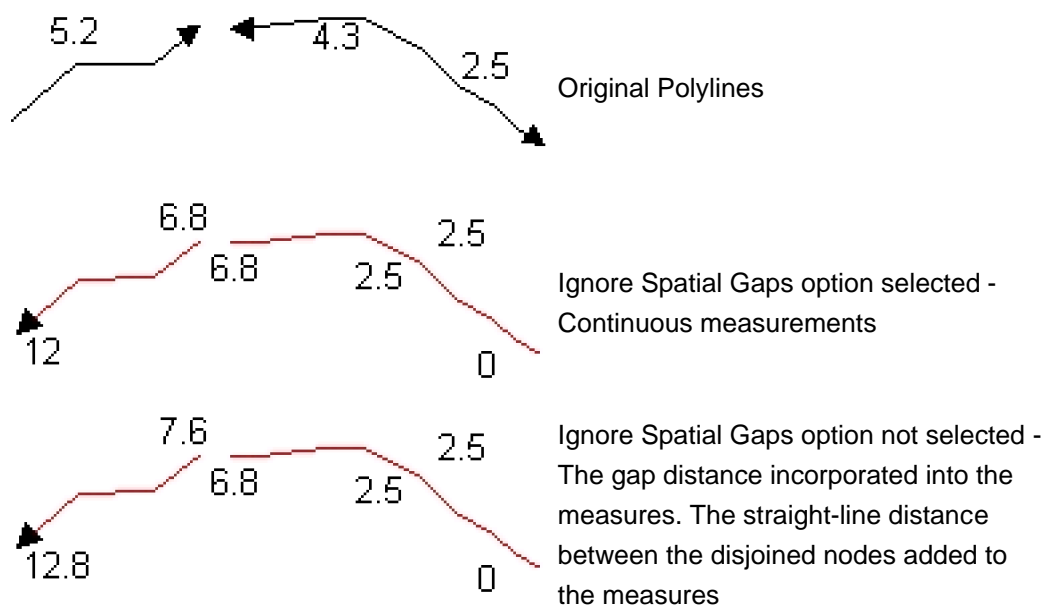
Notes:

- 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.



- 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.



- 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

Calibrate routes with points

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

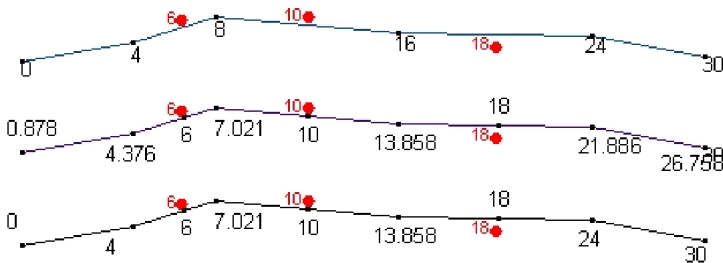
Output:

- 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

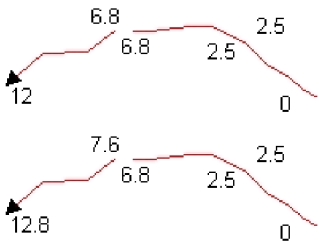
Only "Interpolate between" option used. The vertices before and after calibration points preserve their original measures

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 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.



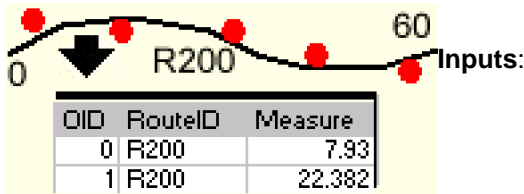
Ignore Spatial Gaps option selected - Continuous measurements

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 and the calibration points dataset

Locate point features along routes

Finds the route and measure information for the points from a Point layer and creates a point 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. The wizard allows the resulting data to be added to the map as an event layer or a standalone table.



- 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 feature layer which points will be located on the routes
- Search tolerance - only the points that are closer to the route than this tolerance will be written to the output event table

Output:


- A dbf table with event record for each point located.
- The dbf table will contain all the original fields of the point dataset
- Three new fields will be added
 - [ET_KYE] - the route identifier field. The values will correspond to the route on which each point was located
 - [ET_M] - the measure of each point
 - [ET_Dist] - the distance of the points from the routes

Notes:

- If the option to add the result as an event layer, two additional fields will be available
 - [ET_Angle] - the normal angle of the route where the event is placed
 - [ET_Error] - indicates whether the event was properly located. If the value in this field indicates some kind of error the shape of the event point will be empty.

Locate polygon features along routes

Finds the route and measure information at the geometric intersection of the input polygon layer and the route layer and creates a line 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. The wizard allows the resulting data to be added to the map as a line event layer or a standalone table.



OID	RouteID	From_M	To_M
0	R200	17.737	28.212
1	R200	28.212	40.933

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 Polygon feature layer which features will be located on the routes

Output:

- A dbf table with event record for each intersection of a polygon with a route.
- The dbf table will contain all the original fields of the polygon dataset
- Three new fields will be added
 - [ET_KYE] - the route identifier field. The values will correspond to the route on which each polygon was located
 - [ET_FromM] - the FROM measure of each intersection
 - [ET_ToM] - the TO measure of each intersection

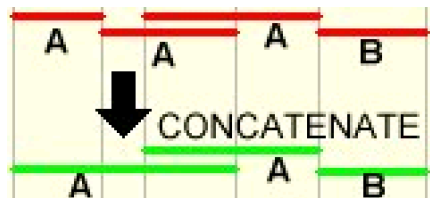
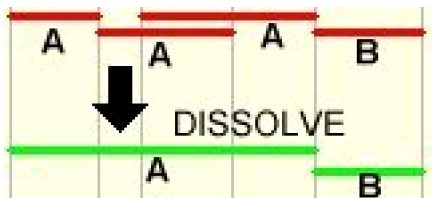
Notes:

- If the option to add the result as an event layer, an additional field will be available
 - [ET_Error] - indicates whether the event was properly located. If the value in this field indicates some kind of error the shape of the event point will be empty.

Dissolve/Concatenate Event Layers

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 dbf table with the aggregated events
- The dbf table will contain all the original fields of the table of the input event layer

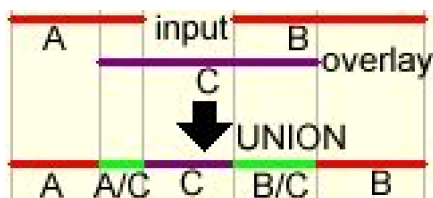
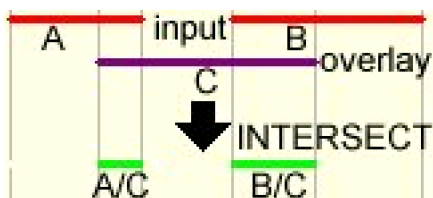
Notes:

- If the option to add the result as an event layer, an additional field will be available
 - [ET_Error] - indicates whether the event was properly located. If the value in this field indicates some kind of error the shape of the event point will be empty.

Intersect/Union Event Layers

Both Union and Intersect functions combine two line event layers in a single line event table.

- Union will split the input linear events at their intersections and writes them to the new event table
- Intersect will write in the output event table only the overlapping events from the input event layers



Input:

- An input line event layer
- An overlay line event layer

Output:

- A new dbf table with the events that result from the overlaying the input layers
- The dbf table will contain all the original fields of the input and overlay tables

Notes:

- If the option to add the result as an event layer, an additional field will be available
 - [ET_Error] - indicates whether the event was properly located. If the value in this field indicates some kind of error the shape of the event point will be empty.

COGO Inverse

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

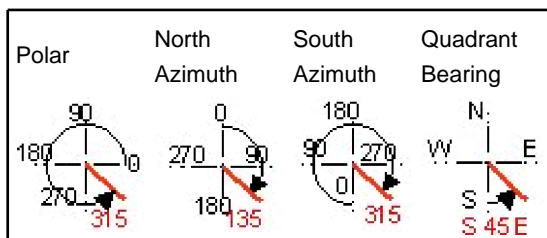
Converts a polyline or a polygon data set to a feature class containing only single segmented polylines. For each segment the COGO attributes are calculated and added to the resulting attribute table. The attributes of the original features are copied to the output features.

Inputs:

- A Polyline, PolylineZ(M), Polygon, PolygonZ(M) 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
- Since the feature class format cannot handle true arcs, if the source dataset is Geodatabase (Personal or SDE) and contains true arcs, these arcs will be represented by their linear approximation. One record will be created for each arc and all the COGO attributes of the original arc will be calculated

ToolBox implementation

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

Command line syntax

ET_GPCogolInverse <input_dataset> <out_feature class> <NorthAzimuth | SouthAzimuth | Polar | QuadrantBearing> <DD | DMS | Radians | Gradians | Gons> {linear_precision} {angular_precision} {add_start_end_coordinates} {add_ZM_attributes}

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<NorthAzimuth SouthAzimuth Polar QuadrantBearing>	Direction Type. A String - the type of the output angle for the direction of the segments.
<DD DMS Radians Gradians Gons>	Direction Unit. A String - the type of the output angle for the direction of the segments.
{linear_precision}	A Number representing the number of places after the decimal point for the output linear measurements
{angular_precision}	A Number representing the number of places after the decimal point for the output angular measurements
{add_start_end_coordinates}	A Boolean indicating whether the Start & End point coordinates will be recorded for the segments
{add_ZM_attributes}	A Boolean indicating whether the Z/M values of the segments will be recorded in the output attribute table. If the input features do not have Z/M values <i>this parameter</i> is set to False automatically

Scripting syntax

ET_GPCogolInverse (input_dataset, out_feature class, direction_type, direction_unit, linear_precision, angular_precision, add_start_end_coordinates, add_ZM_attributes)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CogolInverse(plnFC As IFeatureClass, sOutFName As String, bCoords As Boolean, bZattrib As Boolean, sDirType As String, sDirUnits As String, iLinPrec As Integer, iAngPrec As Integer) As IFeatureClass

Features To Envelopes

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

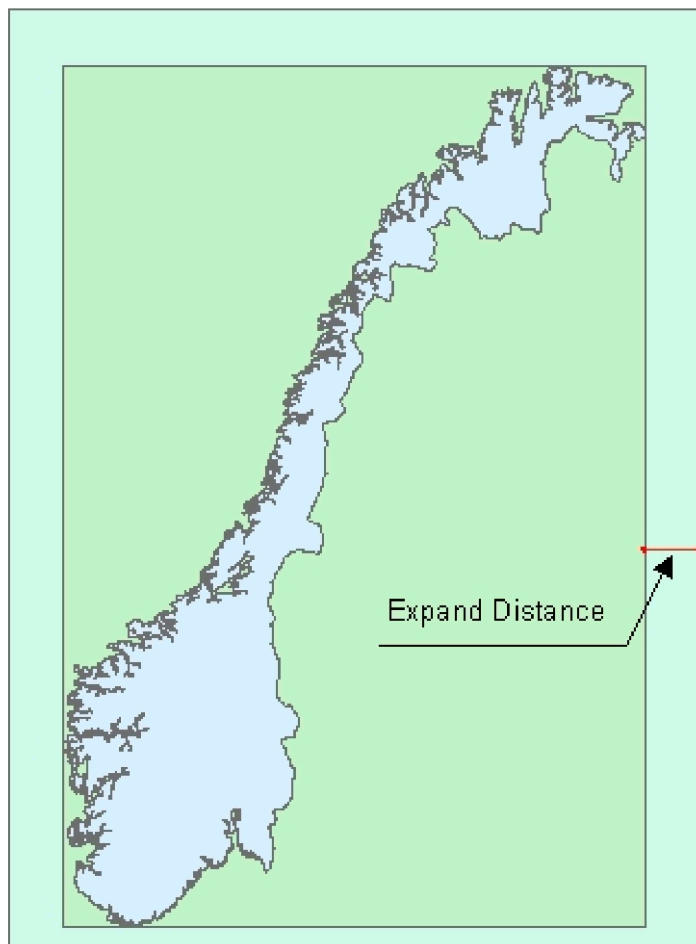
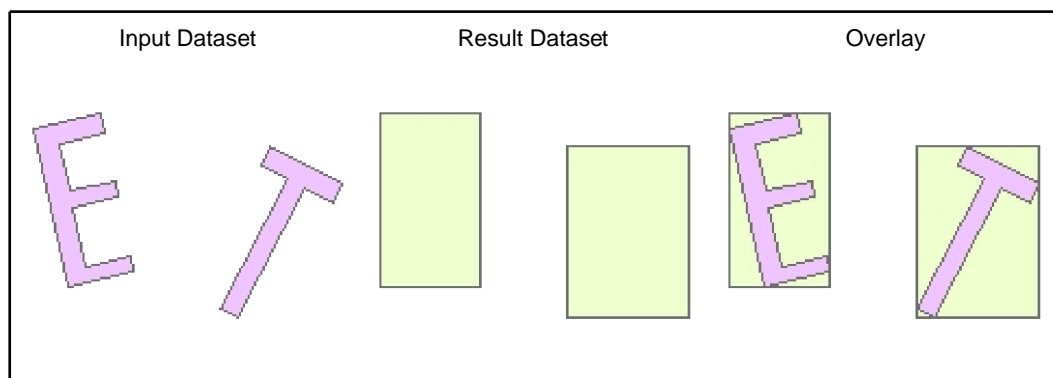
Inputs:

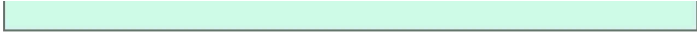
- A Polyline, Polygon or Multipoint feature class
- Expand distance. A distance in the units of the spatial reference of the input dataset with which the envelope of each feature will be expanded. The parameter is optional. The default value is 0.

Outputs:

- A polygon feature class. All attributes of the original features are preserved

Examples:





ToolBox implementation

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

Command line syntax

ET_GPFeaturesToEnvelopes<input_dataset> <out_feature_class> {Expand_Distance}

Parameters

Expression	Explanation
<input_dataset>	A Polyline, Polygon or Multipoint feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Expand_Distance}	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

Scripting syntax

ET_GPFeaturesToEnvelopes (input_dataset, out_feature_class, Expand_Distance)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FeaturesToEnvelopes(plnFC As IFeatureClass, sOutFName As String, Optional dExpandDistance As Double = 0) As IFeatureClass

Features To Minimum Bounding Circles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates a circular bounding polygon from each feature in the input feature class. Attributes of the original features are transferred to the resulting polygons.

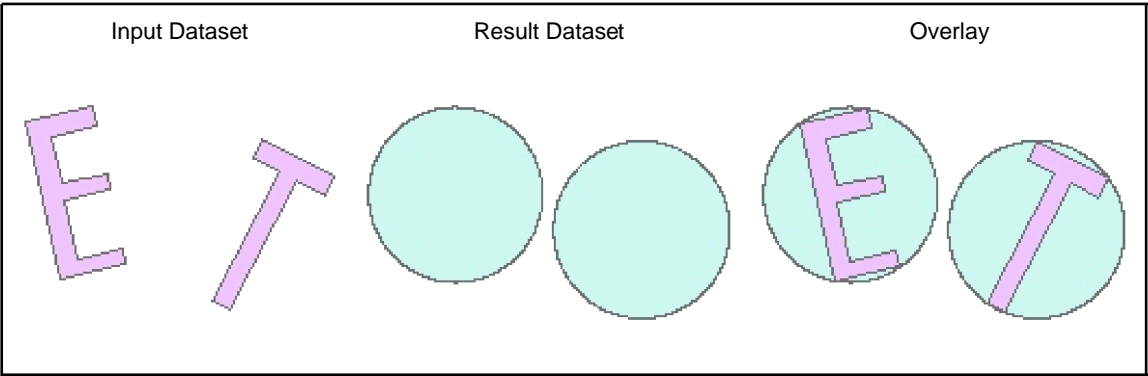
Inputs:

- A Polyline, Polygon or Multipoint feature class

Outputs:

- A polygon feature class. All attributes of the original features are preserved

Examples:



ToolBox implementation

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

Command line syntax

ET_GPFeaturesToCircles <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polyline, Polygon or Multipoint feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPFeaturesToCircles (input_dataset, out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FeaturesToCircles(plnFC As IFeatureClass, sOutFName As String) As IFeatureClass

Features To Convex Polygons

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Creates a convex polygon from each feature in the input feature class. Attributes of the original features are transferred to the convex polygons.

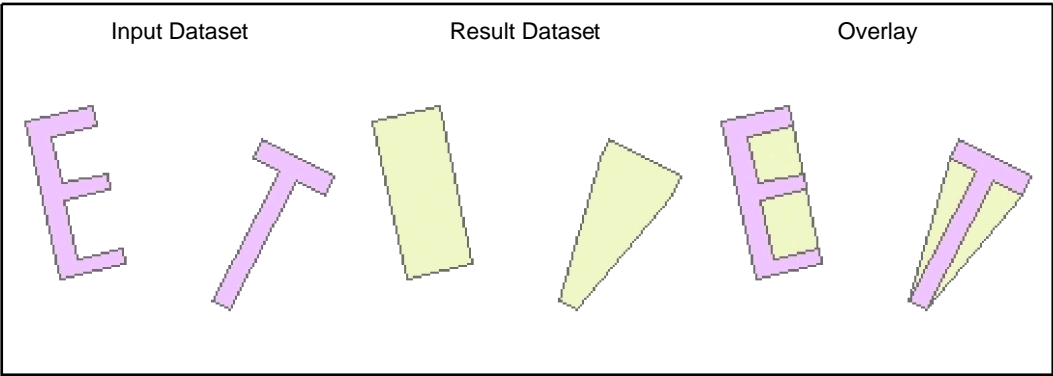
Inputs:

- A Polyline, Polygon or Multipoint feature class

Outputs:

- A polygon feature class. All attributes of the original features are preserved

Examples:



ToolBox implementation

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

Command line syntax

ET_GPFeaturesToConvexPolygons<input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A Polyline, Polygon or Multipoint feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPFeaturesToConvexPolygons(input_dataset, out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FeaturesToConvexPolygons(plnFC As IFeatureClass, sOutFName As String) As IFeatureClass

Features To Minimum Bounding Rectangles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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:

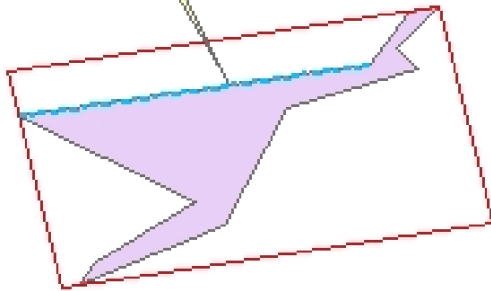
- A Polyline or Polygon feature class
- The orientation of the rectangles to be created
 - Along the longest segment of the polygon boundary
 - Along the longest axis of the original polygons
 - Minimum area rectangle

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

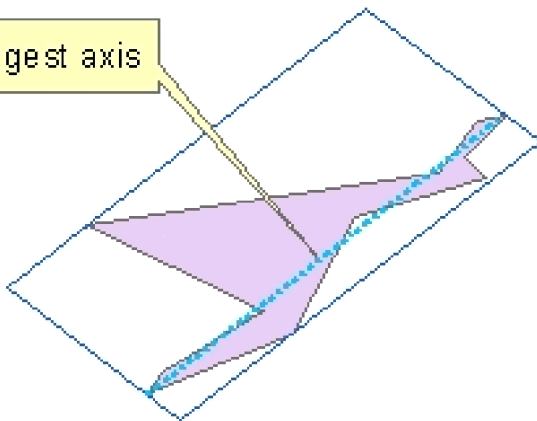
Examples:

Longest segment



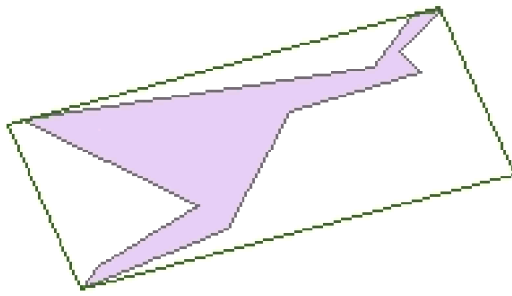
Bounding rectangle aligned with the longest segment of the boundary of the input polygon

Longest axis



Bounding rectangle aligned with the longest axis of the boundary of the input polygon

Minimum area bounding rectangle



ToolBox implementation

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

Command line syntax

ET_GPFeaturesToRectangles <input_dataset> <out_feature_class><Alignment>

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
{Alignment}	<div>A string defining the orientation of the rectangles to be created</div> <ul style="list-style-type: none">• "Longest_Segment" - aligns the rectangle along the longest segment of the polygon boundary• "Longest_Axis" - aligns the rectangle along the longest axis of the original polygons• "Min_Area" - Minimum area rectangle

Scripting syntax

ET_GPFeaturesToRectangles (input_dataset, out_feature_class, Alignment)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

FeaturesToRectangles(pInFC As IFeatureClass, sOutFName As String, Optional sAlignment As String = "")
As IFeatureClass

Lines from Points Direction and Distance

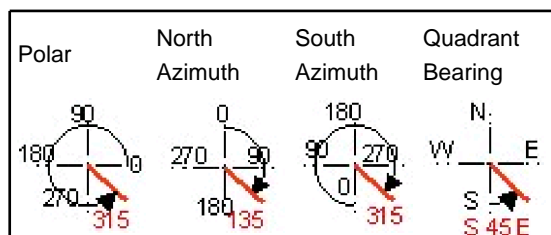
[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 feature class
- The attributes of the input point features are preserved

Example :

Input Points	Point Attribute Table	Resulting Polylines																												
	<table><tr><th>FI</th><th>Shap</th><th>Direction</th><th>Distance</th></tr><tr><td>0</td><td>Point</td><td>N 77-45-13.20 E</td><td>1064.4766</td></tr><tr><td>1</td><td>Point</td><td>S 35-46-12.41 E</td><td>924.303</td></tr><tr><td>2</td><td>Point</td><td>S 68-17-9.83 W</td><td>980.8273</td></tr><tr><td>3</td><td>Point</td><td>N 37-2-10.37 W</td><td>1111.2219</td></tr><tr><td>4</td><td>Point</td><td>S 12-34-11.25 E</td><td>1222.7778</td></tr><tr><td>5</td><td>Point</td><td>N 73-32-2.71 W</td><td>1706.9954</td></tr></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	
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																											

ToolBox implementation

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

Command line syntax

ET_GPLinesFromPointDirDist <input_dataset> <out_feature class><direction_field><distance_field> <NorthAzimuth | SouthAzimuth | Polar | QuadrantBearing> <DD | DMS | Radians | Gradians | Gons>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer

<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<direction_field>	A String representing the name of a field 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 attribute table of the input dataset field name. The field has the values for the distances of the lines to be created.
<NorthAzimuth SouthAzimuth Polar QuadrantBearing>	Direction Type. A String - the type of the output angle for the direction of the segments.
<DD DMS Radians Gradians Gons>	Direction Unit. A String - the type of the output angle for the direction of the segments.

Scripting syntax

ET_GPLinesFromPointDirDist (input_dataset, out_feature class,direction_field,distance_field, direction_type, direction_unit)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

LinesFromPointDirDist(pInFC As IFeatureClass, sOutFName As String, sDirField As String, sDistField As String, sDirType As String, sDirUnits As String) As IFeatureClass

Points Along Polylines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

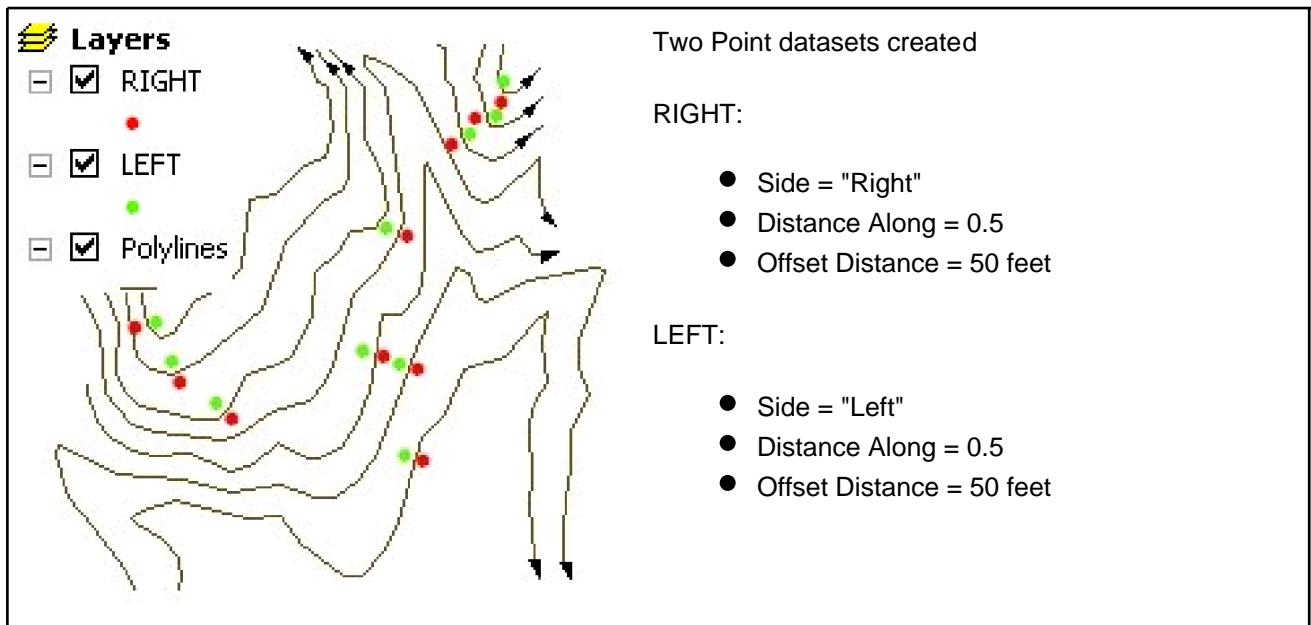
Outputs:

- 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_Alone] - 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:



ToolBox implementation

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

Command line syntax

ET_GPLinesFromPointDirDist <input_dataset> <out_feature_class> <relative_distance> <Both | Left | Right> {offset}

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<relative_distance>	<p>A Double representing the relative distance along the polyline. A value between 0 and 1 indicating the distance from the from point as a ratio.</p> <ul style="list-style-type: none"> ● 0 indicates the start point ● 0.5 indicates a point in the middle of the polyline ● 1 indicates the end point of the polyline
<Both Left Right>	<p>A String - On which side of the polyline will be placed the point if the offset distance is used - three options are available</p> <ul style="list-style-type: none"> ● "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
{offset}	A Double representing the distance from the polyline for the points to be created. If not specified, the points will be on the polylines

Scripting syntax

ET_GPLinesFromPointDirDist (input_dataset, out_feature_class,relative_distance, offset_side, offset)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointsAlongPolylines(plnFC As IFeatureClass, sOutFName As String, dAlong As Double, sSide As String,
Optional dOffset As Double = 0) As IFeatureClass

Copyright © Ianko Tchoukanski

Create Station Lines

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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.

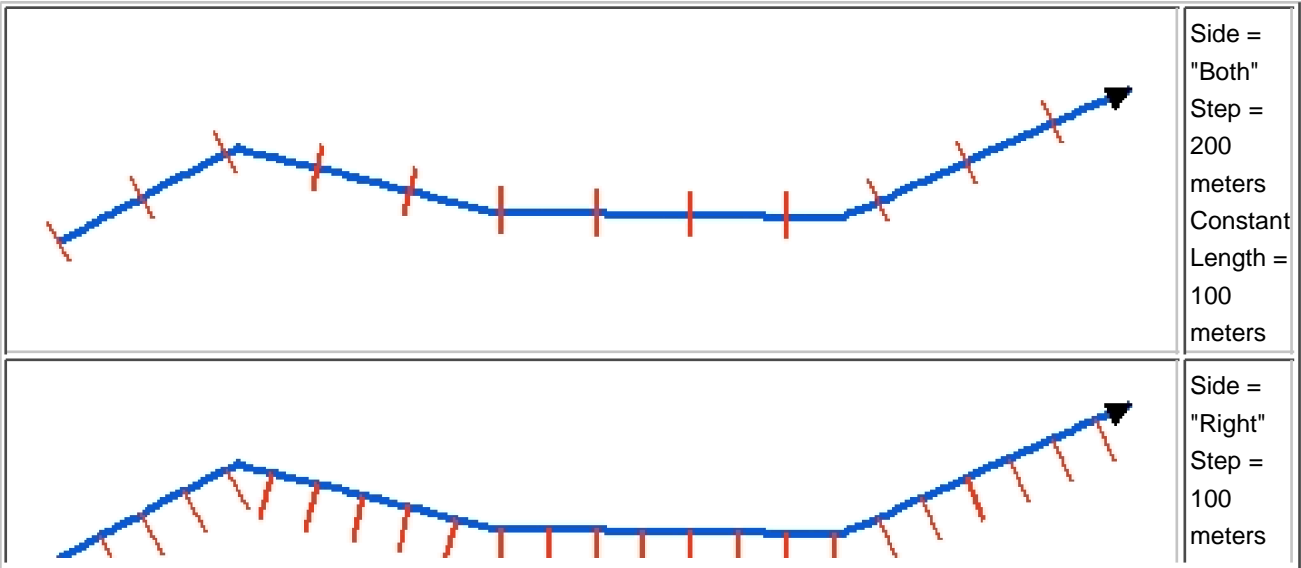
Outputs:

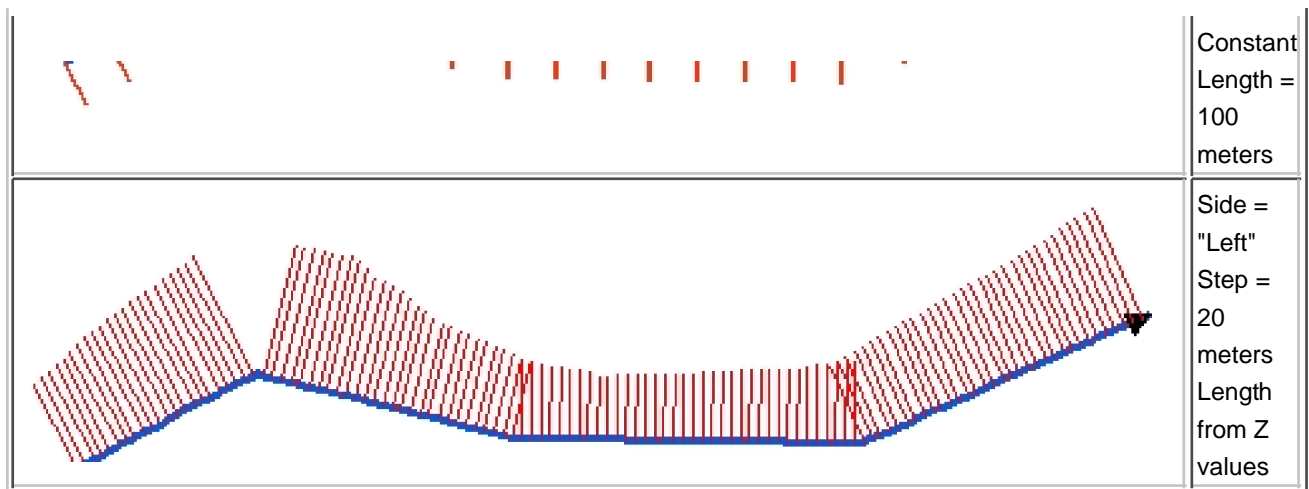
- New Polyline feature class with single segmented polylines perpendicular to the input polylines, distributed along the input polylines based on the user selected options.
- The attributes of the original polylines are preserved
- The following fields are added to the point attribute table
 - [ET_ID] - 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:





ToolBox implementation

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

Command line syntax

ET_GPStationLines<input_dataset> <out_feature class> <station_distance> <Both | Left | Right> <Constant | M Values | Z Values> {lines_length}

Parameters

Expression	Explanation
<input_dataset>	A Polyline or Polygon feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<station_distance>	A Double representing the distance (in the units of the spatial reference of the input feature class) between the station lines along the input polylines
<Both Left Right>	A String representing the side of the station lines in relation to the original polylines.
<Constant M Values Z Values>	A String indicating where the length of the station lines will be taken from
{lines_length}	A Double representing the length (in the units of the spatial reference of the input feature class) of the station lines. Used only with "Constant" option.

Scripting syntax

ET_GPStationLines(input_dataset, out_feature class, station_distance, side, length_from, lines_length)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

CreateStationLines(plnFC As IFeatureClass, sOutFName As String, dStationDistance As Double, sSide As String, sLengthFrom As String, Optional dLength As Double = 0) As IFeatureClass

Copyright © Ianko Tchoukanski

Points To Pie Segments

<

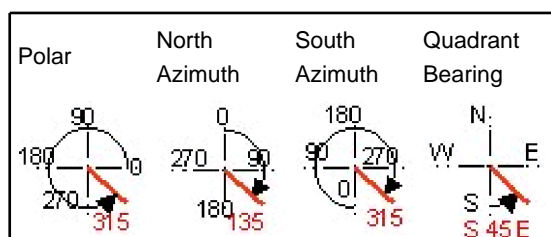
[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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

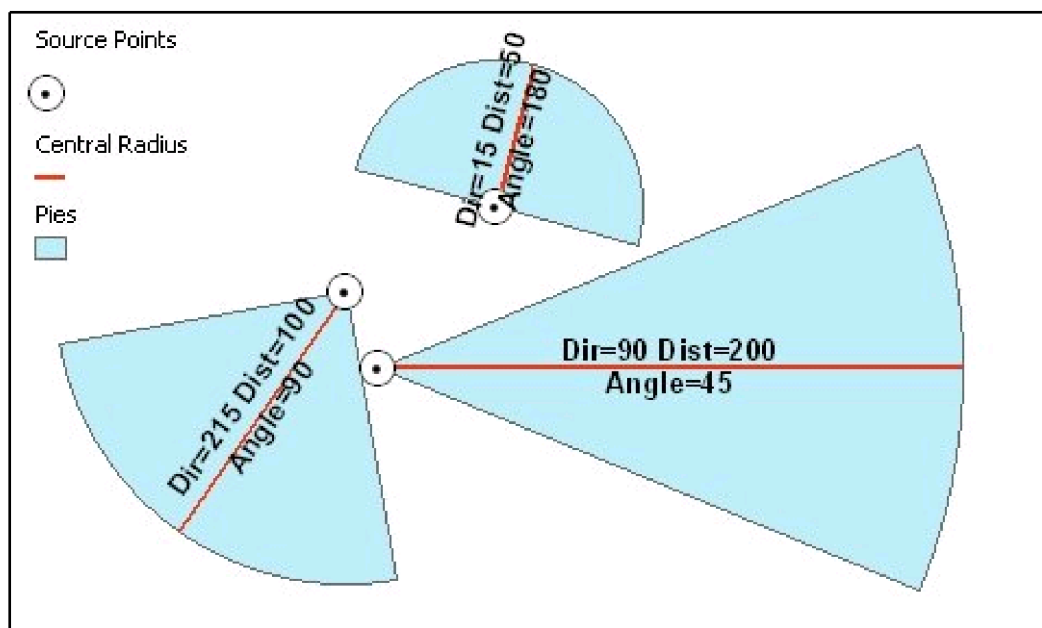


- Direction Angle Units
 - DD - Decimal Degrees
 - DMS - Degrees Minutes Seconds
 - Rad - Radians
 - Grad - Gradians (One gradian is equal to 1/400 circle)
 - Gon - Gons - used in some European countries (One gon is equal to 1/400 circle)

Outputs:

- New polyline feature class
- The attributes of the input point features are preserved

Example :



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

Command line syntax

ET_GPPointsToPieSegments <input_dataset> <out_feature class> <distance_field> <direction_field> <angle_field>
<NorthAzimuth | SouthAzimuth | Polar | QuadrantBearing> <DD | DMS | Radians | Gradians | Gons>

Parameters

Expression	Explanation
<input_dataset>	A Point feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)
<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 radius of the circular segments to be created
<direction_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 directions of the circular segments to be created. The direction defines the central radius of the segment.
<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 pie circular segments to be created.
<NorthAzimuth SouthAzimuth Polar QuadrantBearing>	Direction Type. A String - the type of the output angle for the direction of the segments. Go to the main page of the function for a description
<DD DMS Radians Gradians Gons>	Direction Unit. A String - the type of the output angle for the direction of the segments. Go to the main page of the function for a description

Scripting syntax

ET_GPPointsToPieSegments (input_dataset, out_feature class,direction_field,distance_field, angle_field, direction_type, direction_unit)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PointsToPieSegments(pInFC As IFeatureClass, sOutFName As String, sDistField As String, sDirField As String, sAngleField As String, sDirType As String, sDirUnits As String) As IFeatureClass

Polygons to Equal Area Circles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

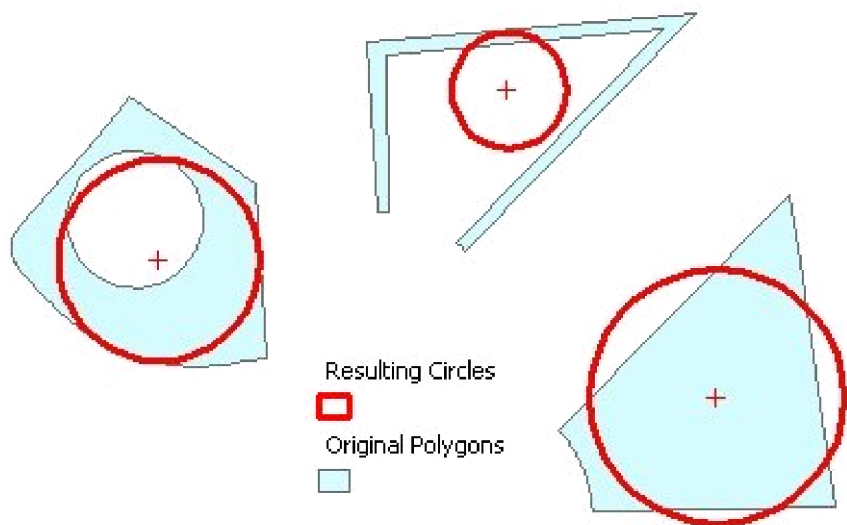
Creates a circular polygon with equal area for each polygon from the input feature class. The center of the circle is located in the centroid of the original polygon. The attributes of the original features are transferred to the resulting polygons.

Inputs:

- A Polygon feature layer or feature class.

Outputs:

- New polygon feature class.



ToolBox implementation

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

Command line syntax

ET_GPPolygonsToEqualAreaCircles <input_dataset> <out_feature_class>

Parameters

Expression	Explanation
<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPolygonsToEqualAreaCircles (input_dataset,out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonsToEqualAreaCircles(plnFC As IFeatureClass, sOutFName As String) As IFeatureClass

Copyright © Ianko Tchoukanski

Polygons to Maximum Inscribed Circles

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

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 Polyline, Polygon or Multipoint feature class

Outputs:

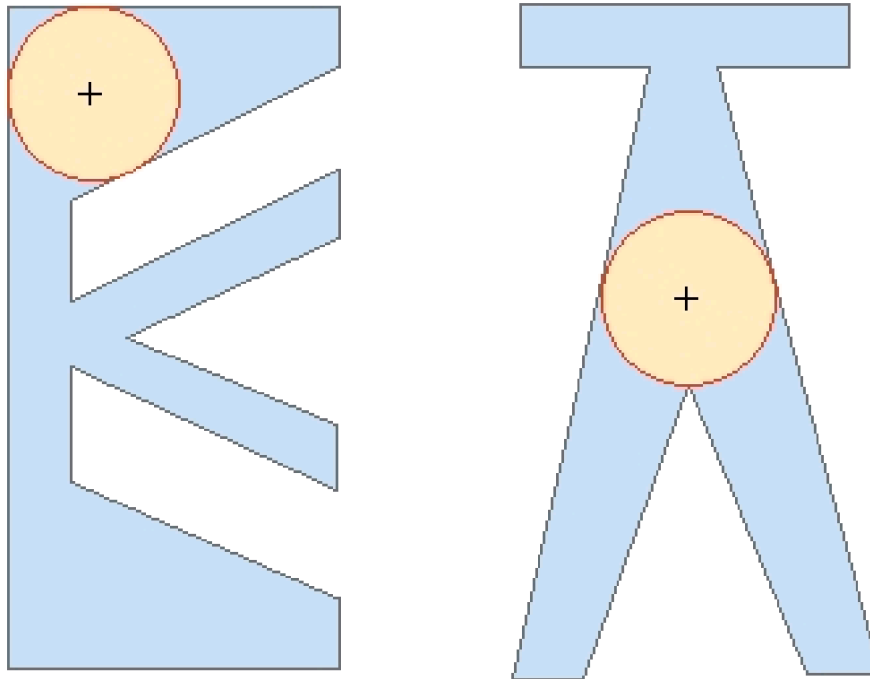
- A polygon feature class. All attributes of the original features are preserved
- A new field [ET_Radius] is added and populated for each polygon.

Note:

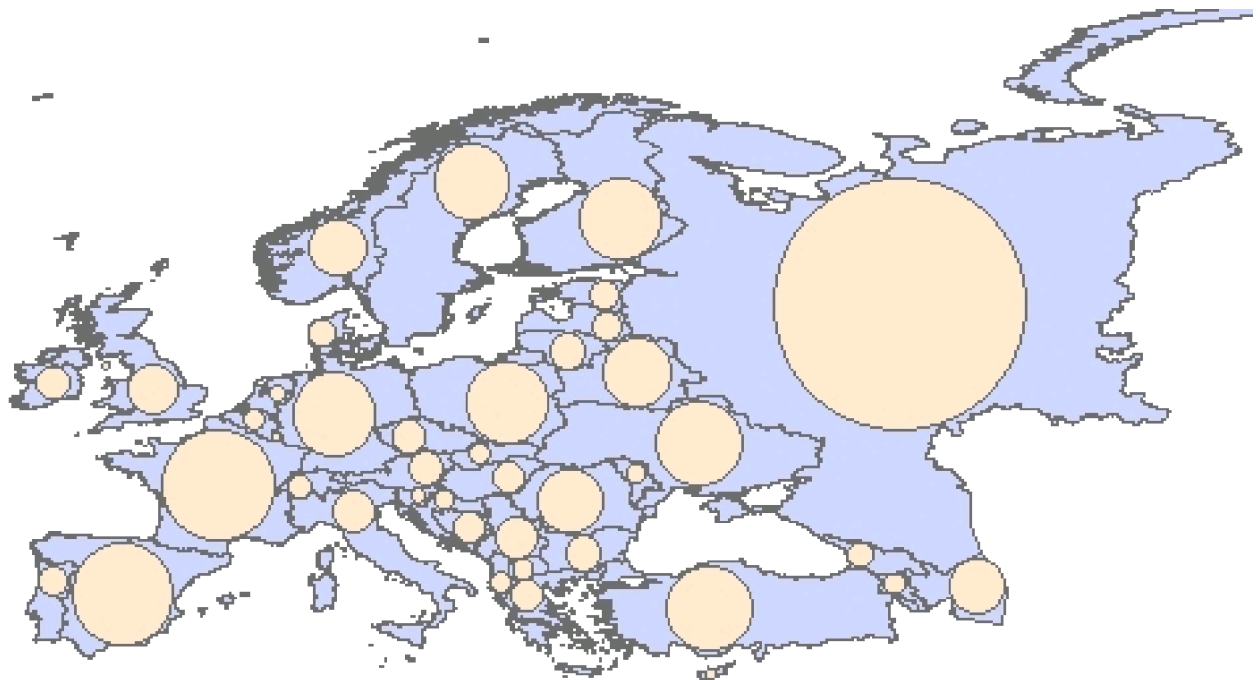
The function uses an interpolation algorithm and the precision of the calculation might not be 100%

Examples:

Example 1



Example 2 - Europe



ToolBox implementation

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

Command line syntax

ET_GPPolygonToMaxInscribedCircles <input_dataset> <out_feature class>

Parameters

Expression**Explanation**

<input_dataset>	A Polygon feature class or feature layer
<out_feature_class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPolygonToMaxInscribedCircles (input_dataset, out_feature_class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

PolygonsToMaxInscribedCircles(pInFC As IFeatureClass, sOutFName As String) As IFeatureClass

Copyright © Ianko Tchoukanski

Utility functions for .NET

Some functions available in the .Net implementation of ET GeoWizards that will help with handling the input parameters.

- Feature Class from Path - gets a feature class from a string representing the full name of the dataset

FeatureClassFromPath(sOutFName As String) As IFeatureClass

- Does Feature class exist

DoesFCExist(ByVal pInFC As IFeatureClass) As Boolean

- Delete feature class

DeleteFC(ByVal pInFC As IFeatureClass) As Boolean

- Get temp feature class name. Derives a name of a feature class in the temp folder of ET GeoWizards that does not exist

GetTempFCName(ByVal sOutType As String) As String

- Get Default Fuzzy tolerance

GetDefaultFuzzy(ByVal pInFC As IFeatureClass) As Double

- Release Concurrent License - for the concurrent versions only

ReleaseConcurrentLicense() As Boolean

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.

In the User Interface

- Basic functions
 - Create New feature class
 - Delete Multiple Fields
 - Sort Shapes
 - Move Shapes
 - Rotate Shapes
 - Scale shapes
 - Generate
 - Ungenerate
 - Explode multi-part features
 - Vector Grid
 - Closest Feature Distance
 - Order Fields
 - Redefine Fields
 - Copy Fields
- Conversion functions
 - Polygon To Polyline
 - Polygon To Point
 - Polyline To Point
 - Polyline To Polygon
 - Point To Polyline
 - Polyline To Polygon
 - Multipoint To Point
 - Shape Z (M) To Shape
 - Polygon Z (M) To Point
 - Polyline Z (M) To Point
 - Point Z (M) To Point
 - Point To Polygon Z (M)
 - Point To Polyline Z (M)
 - Point To Point Z (M)
- Geoprocessing functions
 - Clip layer
 - Erase layer
 - Merge Layers
- Surface functions
 - Convex Hull
 - Calculate True Surface Area
- Polyline functions
 - Generalize polyline layer
 - Densify polyline layer
 - Get PolylineZ characteristics
 - Flip Polylines
- Point functions
 - Create Point Grid
 - Point Distance
 - Station Points

In Arc Toolbox , Model Builder,Python Scripts

- Calculate
- Calculate Area
- Calculate Length
- Rename Field
- Add Attribute Index
- Add Spatial Index
- Get Point Coordinates
- Get Polygon Coordinates
- Get Polyline Coordinates

TRIANGULATED IRREGULAR NETWORK

The TIN model represents a surface as a set of contiguous, non-overlapping triangles. Within each triangle the surface is represented by a plane. The triangles are made from a set of points called mass points.

Mass points can occur at any location, the more carefully selected, the more accurate the model of the surface. Well-placed mass points occur where there is a major change in the shape of the surface, for example, at the peak of a mountain, the floor of a valley, or at the edge (top and bottom) of cliffs.

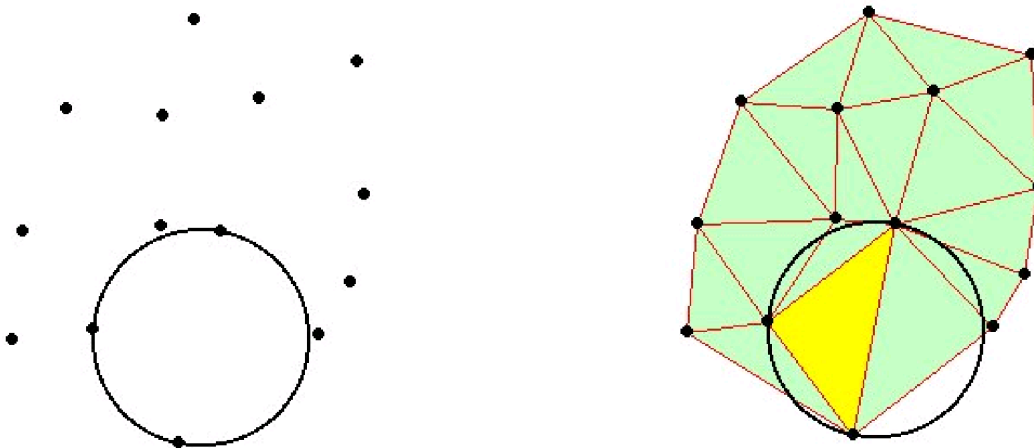
The TIN model is attractive because of its simplicity and economy and is a significant alternative to the regular raster of the GRID model.

Quick comparison:

	TIN	GRID
Advantages	<ul style="list-style-type: none">● ability to describe the surface at different level of resolution● efficiency in storing data	<ul style="list-style-type: none">● easy to store and manipulate● easy integration with raster databases● smoother, more natural appearance of derived terrain features
Disadvantages	<ul style="list-style-type: none">● in many cases require visual inspection and manual control of the network	<ul style="list-style-type: none">● inability to use various grid sizes to reflect areas of different complexity of relief.

The Delaunay Triangulation

Delaunay triangulation is a proximal method that satisfies the requirement that a circle drawn through the three nodes of a triangle will contain no other node



Delaunay triangulation has several advantages over other triangulation methods:

- The triangles are as equi-angular as possible, thus reducing potential numerical precision problems created by long skinny triangles
- Ensures that any point on the surface is as close as possible to a node
- The triangulation is independent of the order the points are processed

TINs from contours

Contours are a common source of digital elevation data. In general all the vertices of the contour lines are used as mass points for triangulation. In many cases this will cause the presence of flat triangles in the surface.

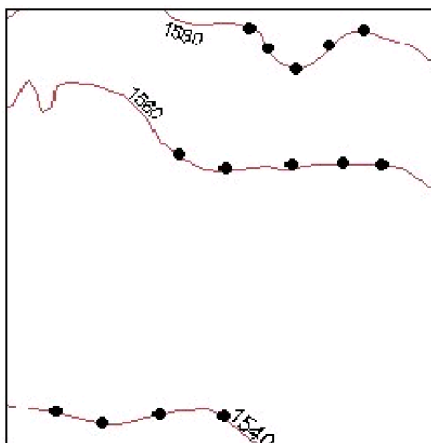
Flat triangles are created whenever a triangle is formed from three nodes with the same elevation value

Flat triangles are frequently generated along contours when the sample points occur along the contour at a distance that is less than the distance between contours. When these "excess" vertices are not removed, the Delaunay triangulation discovers that the closest sample points are those along the same contour, causing the generation of flat triangles.

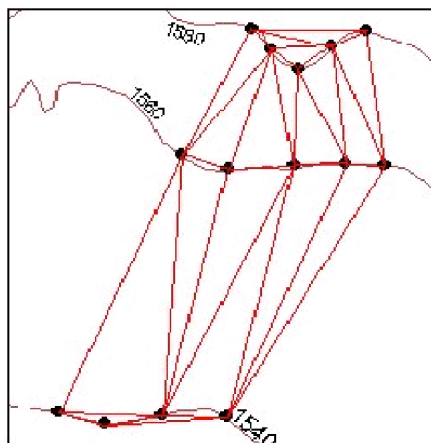
The flat triangles have a slope of 0 and do not have defined aspect. They might cause problems when the surface is used for modeling.

Example:

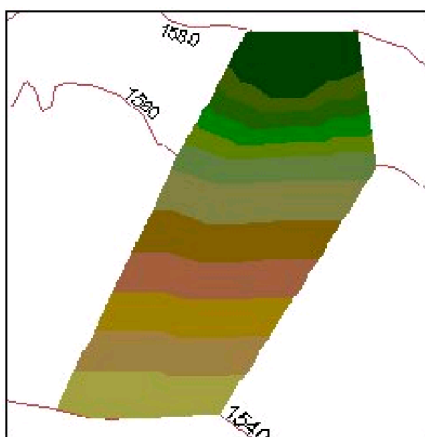
The contours



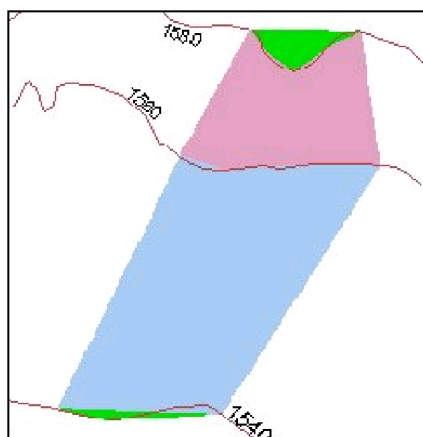
The triangulation - We can see several flat triangles here



The elevation



The slope- The green areas indicate Slope = 0 (flat triangles)



How can we avoid the flat triangles?

- By adding more mass points
- Generalizing the contours
- By adding break lines

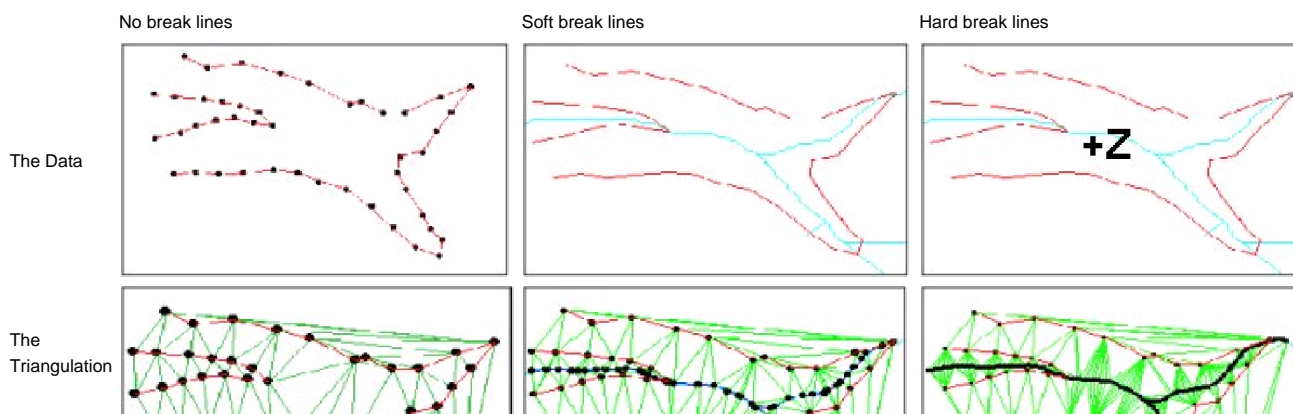
Break lines

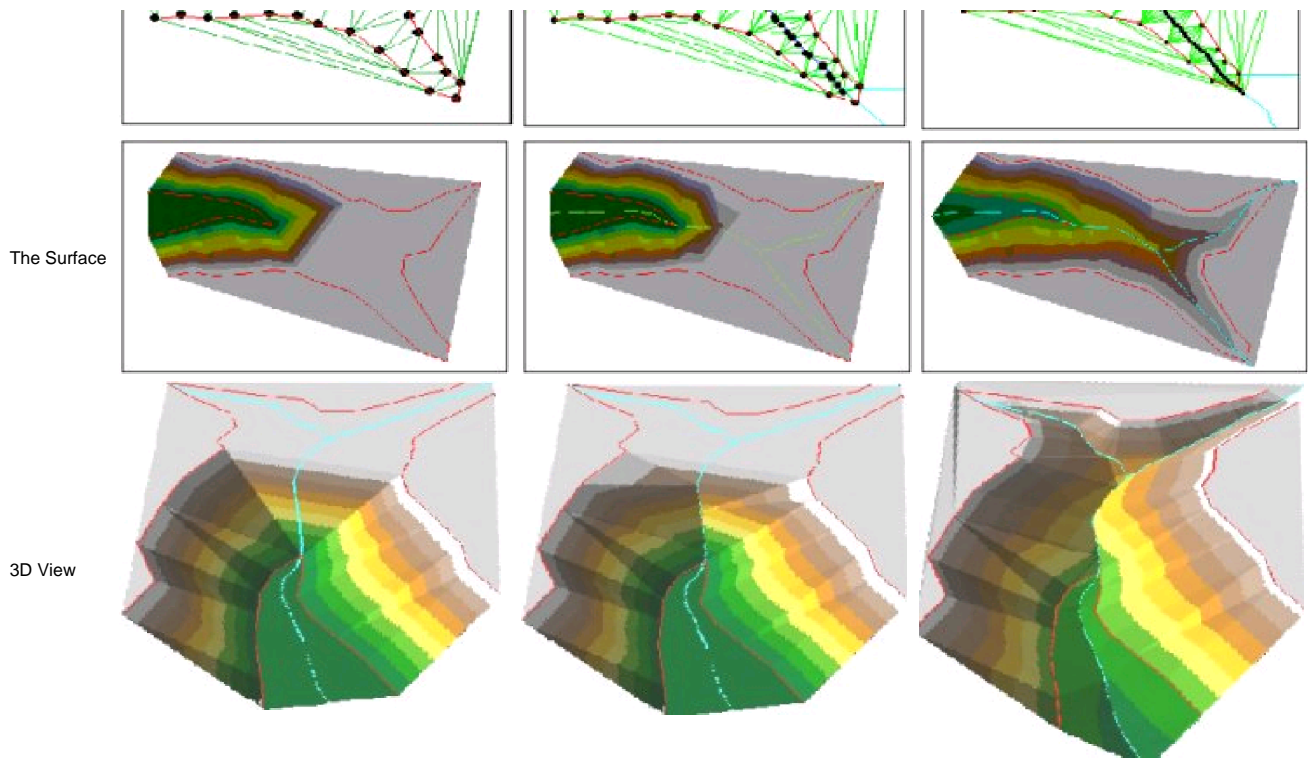
Linear features which define and control surface behavior in terms of smoothness and continuity are called break lines.

Types break lines:

- Soft break lines are used to ensure that linear features and polygon edges are maintained in the tin surface model by enforcing the break line as tin edges. However, they do not define interruptions in surface smoothness – break lines with no Z value
- Hard break lines define interruptions in surface smoothness – break lines with Z value

Example:





Storing TINs

There are basically two ways of storing triangulated networks:

- Triangle by triangle
- Points and their neighbors

The first method is better for storing attributes (slope, aspect ..) for each triangle, but uses more storage space. The second one is better for generating contours and uses less storage space, but slope, aspect , etc must be calculated and stored separately.

NOTE: ET GeoWizards 9.6 and above offer two functions - Smooth Polygons and Generalize Polygons that implement the entire process.

Contents:

- [The purpose](#)
- [ET GeoWizards functions available](#)
- [Definitions](#)
- [The importance](#)
- [Smoothing/Generalizing polygons. Why it is so difficult?](#)
- [So, how to smooth polygons?](#)
- [How to restrict the Smoothing and remove the unnecessary vertices after smoothing?](#)
- [What about the attributes?](#)
- [This is getting too complex....](#)
- [A diagram of the process:](#)

The purpose:

The purpose of this document is to discuss the processes of simplification (generalization) and smoothing features in ArcGIS (ArcView license). While the generalization and smoothing of polylines is comparatively simple process, when applied to polygons there are certain complications connected to the topological relationships between the adjacent polygons.

ET GeoWizards functions available

ArcGIS has standard Generalize and Smooth functions, but they are available only for users with ArcEditor or ArcInfo licenses

ET GeoWizards offers the following tools to ArcGIS users with any license :

- [Generalize](#) - Generalizes (reduces the number of vertices required to represent a polyline) the features of a polyline dataset using the Douglas-Poiker algorithm
- [Densify](#) - Densifies (adds vertices to polyline at a user-specified tolerance) the features of a polyline dataset.
- [Smooth](#) - Smooths the features of a polyline dataset using three different smoothing algorithms
 - Bezier curve
 - B - Spline
 - T - Spline (Tension Spline)

Note that all the functions above are available only for polylines. In this document we'll describe a procedure that will make use of these tools to Generalize/Smooth polygons.

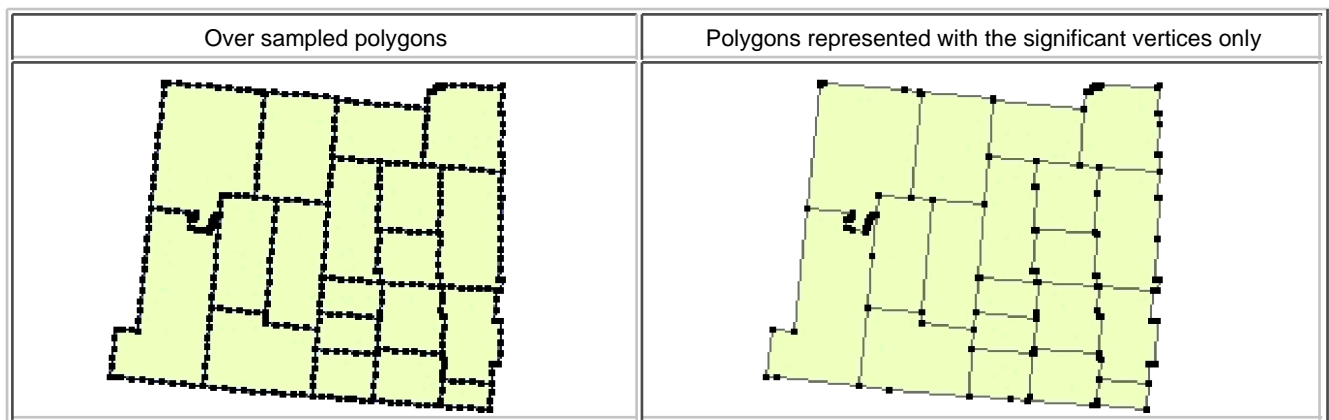
All the links in this document are to the functions available via the interface of ET GeoWizards. If the procedure is to be performed in the Model Builder - refer to the corresponding tools available in the ET GeoWizards ToolBox. If performed via a VBA script - refer to ET GeoWizards Scripting

Definitions:

- Generalization - the process of removing some vertices from a polyline or polygon boundary without destroying its essential shape.
- Smoothing - the process of introducing new vertices in a polyline or polygon boundary in order to achieve shapes with no sharp corners.

The importance:

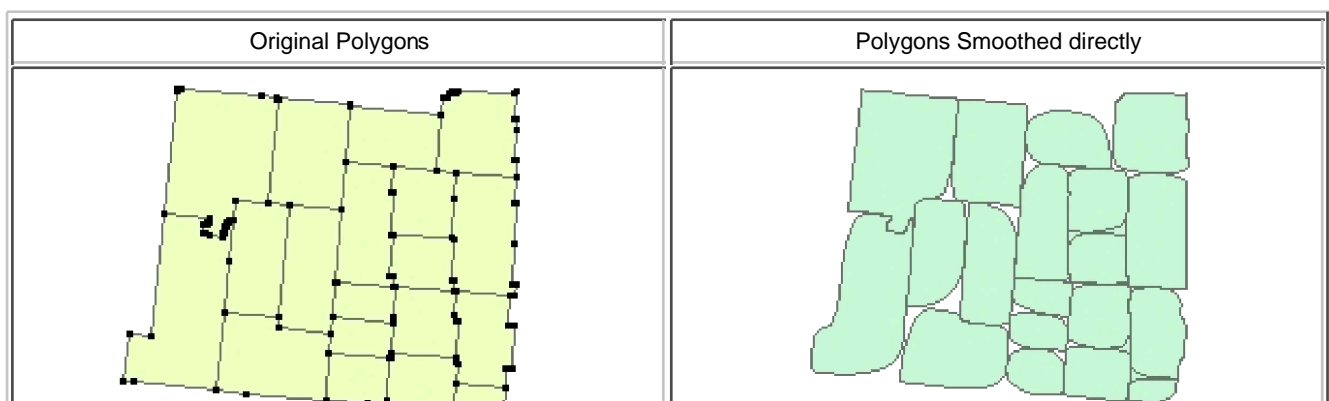
In many cases the data that we receive is over sampled. Nowadays a big part of the GIS data is captured via GPS devices that get locations automatically every few seconds. In this fashion a straight line sometimes is described by many vertices when actually only two vertices are needed. This not only increases the size of the data, but makes certain geoprocessing operations on the data to need extra computer memory to complete. Note that the memory needed is dependant not only on the number of features, but also on the complexity of these features, measured in the total number of vertices.



Smoothing/Generalizing polygons. Why it is so difficult?

In the Shapefiles and GeoDatabases (not in the coverages) each polygon has its own outline. This means that the common boundary between two adjacent polygons is represented by two coincident polylines. If we smooth/generalize these adjacent polygons each of the outlines will be smoothed/generalized separately and DIFFERENTLY. As a result we will get gaps and/or overlaps on the boundary. With other words we will end up with destroyed topology.

Further in this document all the examples and explanations will be based on the Smoothing, because the effect is more pronounced, but everything is applicable for the generalization process as well.



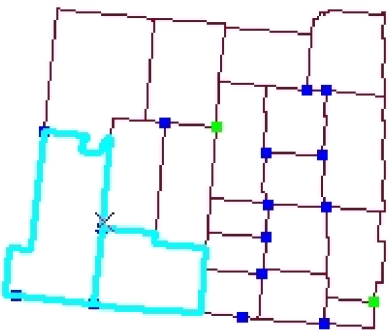
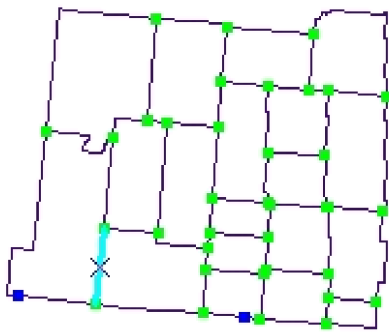
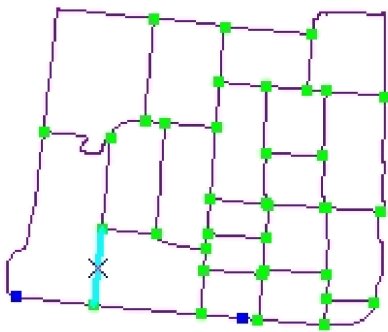
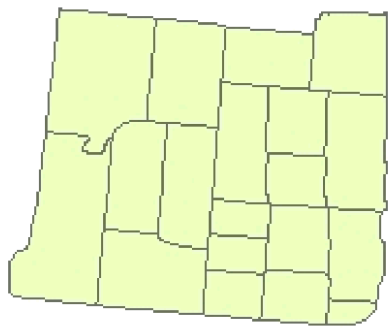
The example above visualizes the gaps resulting from Smoothing the polygons directly.

So, how to smooth polygons?

Lets think a bit in the good old Coverage way, where all the geometries were stored as arcs and the polygons were build from these arcs (no duplicate geometry on the boundary of two adjacent polygons). If something was changed in the geometry of the Arcs it reflected in the polygons built from them. If we smooth/generalize an Arc, the two polygons in which this Arc participates will be smoothed/generalized EQUALLY.



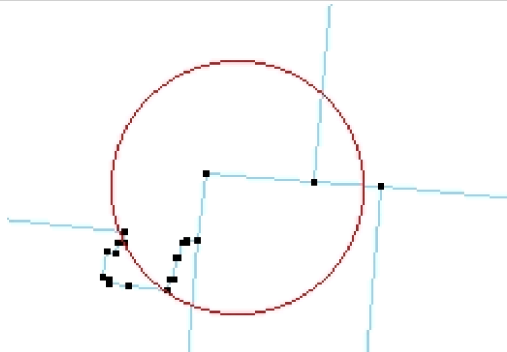
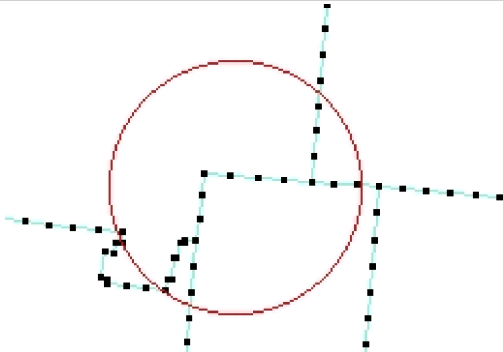
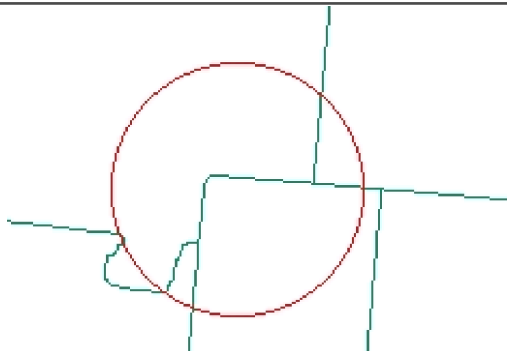
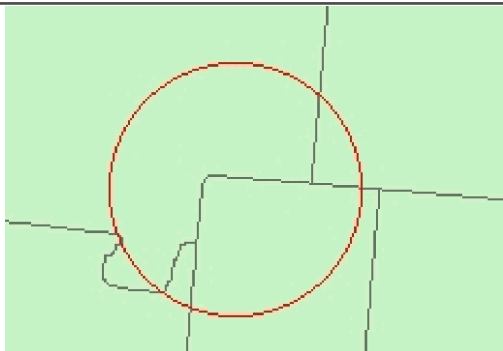
Here is the ArcGIS procedure that emulates this behavior. All the functions are available in ET GeoWizards.

- Convert the polygons to polylines (this just creates a polyline shapefile from the polygon boundaries). [Polygon To Polyline](#) function
- Clean the polylines (this will create all intersections and remove the duplicate polylines on the boundaries between two adjacent polygons). [Clean Polyline](#) function
- Smooth the polylines (there is no duplicate polylines, so no topological problems will be introduced). [Smooth Polyline](#) function
- Build new polygons from smoothed polylines (this will give us smoothed topologically correct polygons). [Build Polygons](#) function

1. Polygon To Polyline	2. Clean Polylines
	
3. Smooth Polylines (B-Spline method used)	4. Build Polygons
	

How to restrict the Smoothing and remove the unnecessary vertices after smoothing?

The degree of smoothing can be controlled by the parameters used in the Smooth function. In some cases however due to the lack of enough vertices the parameters used can not restrict the smoothing too much. In such cases, we can introduce new vertices to the cleaned polylines (**Densify** function) before proceeding with smoothing. After smoothing we can remove the unnecessary vertices before building the polygons.

Polygons smoothed with the procedure above	The degree of smoothing to high
	
Original vertices	Vertices after Densify
	
Smoothed densified polylines	Resulting smoothed polygons
	

The procedure gets just a bit more complex

- Convert the polygons to polylines (this just creates a polyline shapefile from the polygon boundaries). **Polygon To Polyline** function
- Clean the polylines (this will create all intersections and remove the duplicate polylines on the boundaries between two adjacent polygons). **Clean Polyline** function
- Insert new vertices - [Densify Polylines](#) function
- Smooth the polylines (there is no duplicate polylines, so no topological problems will be introduced). **Smooth Polyline** function
- Remove the excess vertices - [Generalize Polylines](#) function
- Build new polygons from smoothed polylines (this will give us smoothed topologically correct polygons). **Build Polygons** function

What about the attributes?

A reasonable question. In the process of converting the polygons to polylines and cleaning the polylines we've lost the polygon attributes. We can rectify this by adding a three simple steps to the procedure:

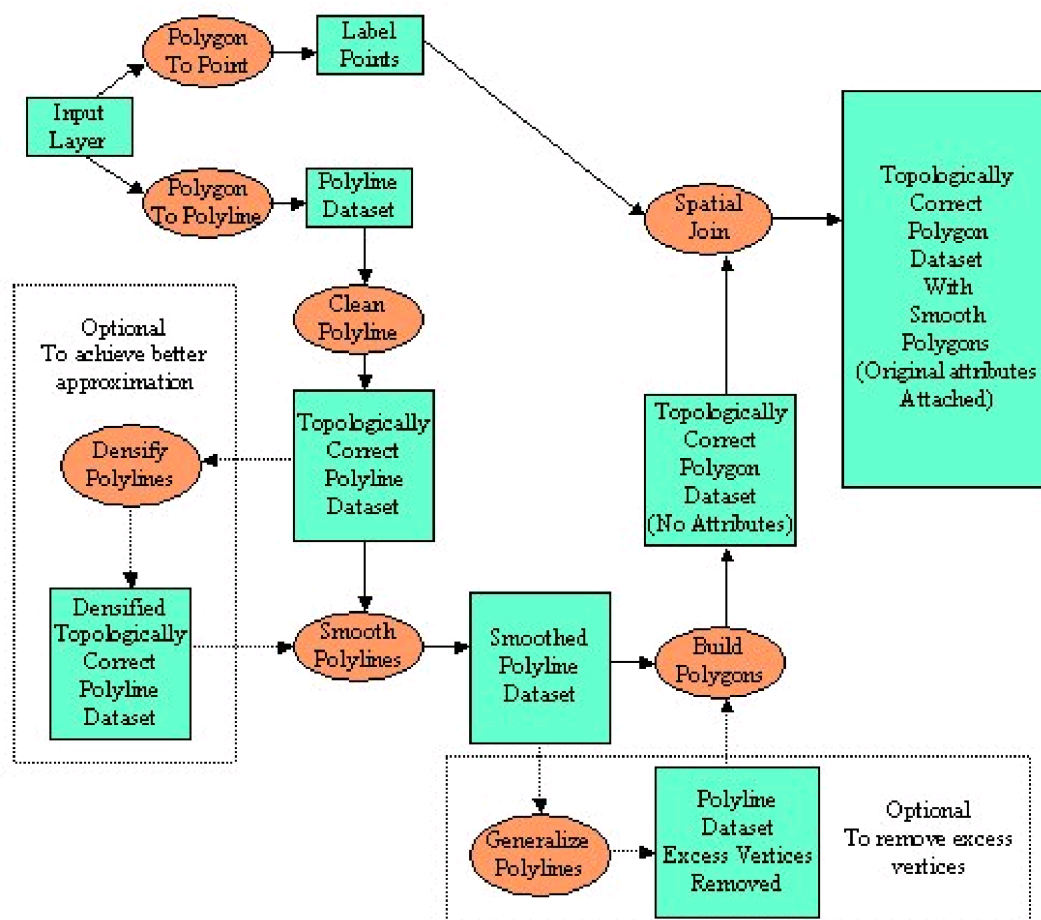
- Get the label points of the smoothed polygons [Polygon To Point](#) function with the Label Point option
- Get the attributes of the original polygons to the label points created above - The standard ArcGIS Spatial Join function (Polygons to Points). If used in the Model Builder - the [Spatial Join](#) tool of ET GeoWizards
- Get the attributes from the label points to the smoothed polygons - The standard ArcGIS Spatial Join function (Points To Polygons). If used in the Model Builder - the [Spatial Join](#) tool of ET GeoWizards

This is getting too complex....

ET GeoWizards offers three ways of achieving the above procedure

- Via the User Interface - just perform the steps one by one using the Wizards available
- With a simple VBA script - many of the functions of ET GeoWizards are available for use within VBA scripts or custom applications written in any COM language. See an example with working code [here](#)
- Create a Geoprocessing Model in the Model Builder (ArcGIS 9.0 and above only) using the [ET GeoWizards geoprocessing tools](#) made available in version 9.2. A model performing the task is included in the download of ET GeoWizards for ArcGIS 9.x

A diagram of the process:





logo_h1.jpg (4952 bytes)

ET GeoWizards HOME
ToolBox User Guide
Scripting User Guide

[User Guide Start Page](#)

[Installation Instructions](#)

[How to use ET GeoWizards](#)

[ET GeoWizards and projections](#)

[ET GeoWizards and Geodatabase](#)

[How to register](#)

[ET GeoWizards toolbar](#)

[Main Dialog](#)

Spatial Relations & Allocation

- [Allocate](#)
- [Build Thiessen](#)
- [Convex Hull](#)
- [Concave Hull](#)
- [Cluster Polygons](#)
- [Spider Diagram](#)
- [Spider Diagram Link](#)

Import/Export

- [Google Earth general](#)
- [Map To Google Earth](#)
- [Import from Google Earth](#)
- [Generate](#)
- [Ungenerate](#)

Point Wizards

- [Clean Point](#)
- [Point Grid](#)
- [Point Distance](#)
- [Point Intersection](#)
- [Snap Point Layer](#)
- [Point Angle and Position](#)
- [Reverse Geocoding](#)

ET GeoWizards is a set of powerful functions that will help the ArcGIS users to manipulate data with easy. It offers a lot of functionality not available as standard in ArcGIS. It also enables the ArcGIS users with ArcView (ArcGIS Basic) licenses to perform some data processing functions currently available only in ArcEditor (ArcGIS Standard) and ArcInfo (ArcGIS Advanced).

The main target of the software are the ArcView license holders, but it will be an asset for everyone using ArcEditor and even ArcInfo

The functionality of ET GeoWizards is available in two different ways

- Via the user friendly wizard type interface
- Via a set of tools for Arc Toolbox (ArcGIS 9.0 or above) which can be used in the Model Builder, Command Line or in Python scripts.

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
- See [How to Register ET GeoWizards](#) for registration information



logo_h1.jpg (4952 bytes)

[ET GeoWizards HOME](#)

[ToolBox User Guide](#)

[Scripting User Guide](#)

[User Guide Start Page](#)

[Installation Instructions](#)

[How to use ET GeoWizards](#)

[ET GeoWizards and projections](#)

[ET GeoWizards and Geodatabase](#)

[How to register](#)

[ET GeoWizards toolbar](#)

[Main Dialog](#)

Spatial Relations & Allocation

- [Allocate](#)
- [Build Thiessen](#)
- [Convex Hull](#)
- [Concave Hull](#)
- [Cluster Polygons](#)
- [Spider Diagram](#)
- [Spider Diagram Link](#)

Import/Export

- [Google Earth general](#)
- [Map To Google Earth](#)
- [Import from Google Earth](#)
- [Generate](#)
- [Ungenerate](#)

Point Wizards

- [Clean Point](#)
- [Point Grid](#)
- [Point Distance](#)
- [Point Intersection](#)
- [Snap Point Layer](#)
- [Point Angle and Position](#)
- [Reverse Geocoding](#)
- [Measure Points](#)
- [Station Points](#)
- [Thin Points](#)
- [Points To Rectangles](#)
- [Connect Points](#)
- [Perpendiculars To Polylines](#)

ET GeoWizards is a set of powerful functions that will help the ArcGIS users to manipulate data with easy. It offers a lot of functionality not available as standard in ArcGIS. It also enables the ArcGIS users with ArcView (ArcGIS Basic) licenses to perform some data processing functions currently available only in ArcEditor (ArcGIS Standard) and ArcInfo (ArcGIS Advanced).

The main target of the software are the ArcView license holders, but it will be an asset for everyone using ArcEditor and even ArcInfo

The functionality of ET GeoWizards is available in two different ways

- Via the user friendly wizard type interface
- Via a set of tools for Arc Toolbox (ArcGIS 9.0 or above) which can be used in the Model Builder, Command Line or in Python scripts.

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
- See [How to Register ET GeoWizards](#) for registration information

- [Measure Points](#)
- [Station Points](#)
- [Thin Points](#)
- [Points To Rectangles](#)
- [Connect Points](#)
- [Perpendiculars To Polylines](#)
- [Disperse Points](#)
- [Random Points On Polylines](#)
- [Random Points In Polygons](#)

Polyline Wizards

- [Clean Polyline](#)
- [Clean Dangling Nodes](#)
- [Clean Pseudo Nodes](#)
- [Split Polyline With Layer](#)
- [Split Polyline](#)
- [Export Nodes](#)
- [Generalize](#)
- [Densify](#)
- [Smooth](#)
- [Snap Polyline Layer](#)
- [Renode Polylines](#)
- [Flip Polylines](#)
- [PolylineZ characteristics](#)
- [Clean Contour Gaps](#)

Polygon Wizards

- [Clean Polygon](#)
- [Eliminate](#)
- [Dissolve Polygons](#)
- [Clean Gaps](#)
- [Advanced Merge](#)
- [Build Polygons](#)
- [Snap Polygon Layer](#)
- [Get Adjacent Polygons](#)
- [Generalize Polygons](#)
- [Smooth Polygons](#)
- [Aggregate Polygons](#)
- [Create Centerlines](#)
- [Partition Polygons](#)
- [Polygon Characteristics](#)
- [Polygon To Polyline Advanced](#)
- [Fill Polygon Holes](#)

Conversion Wizards

- [Polygon To Polyline](#)
- [Polygon To Point](#)
- [Polyline To Point](#)
- [Polyline To Polygon](#)
- [Point To Polyline](#)
- [Point To Polygon](#)
- [Multipoint To Point](#)
- [Shape Z \(M\) To Shape](#)
- [Polygon Z \(M\) To Point](#)
- [Polyline Z \(M\) To Point](#)
- [Point Z \(M\) To Point](#)
- [Point To Polygon Z \(M\)](#)
- [Point To Polyline Z \(M\)](#)
- [Point To Point Z \(M\)](#)
- [Shape To ShapeZ](#)
- [Point To Multipoint](#)

Surface Wizards

- [Build TIN](#)
- [Analyze TIN](#)
- [Interpolate Contours](#)
- [Triangulate Polygons](#)
- [Calculate Surface Area](#)
- [Interpolate Point Elevation](#)
- [Features to 3D](#)
- [ESRI TIN To PolygonZ](#)

Geoprocessing Wizards

- [Clip layer](#)
- [Batch Clip](#)
- [Erase Layer](#)
- [Batch Erase](#)
- [Merge Layers](#)
- [Split By Location](#)
- [Split By Attributes](#)
- [Transfer Attributes](#)
- [Remove Exact Duplicates](#)
- [Symmetrical Difference](#)
- [Closest Feature Distance](#)

Basic Wizards

- [Create New Shapefile](#)
- [Delete Multiple Fields](#)
- [Sort Shapes](#)
- [Move Shapes](#)
- [Rotate Shapes](#)
- [Scale shapes](#)
- [Explode](#)
- [Vector Grid](#)
- [Closest Feature Distance](#)
- [Order Fields](#)
- [Redefine Fields](#)
- [Copy Fields from layer](#)

Miscellaneous Wizards

- [Cogo Inverse](#)
- [Features To Envelopes](#)
- [Features To Convex Polygons](#)
- [Features To Circles](#)
- [Features To Rectangles](#)
- [Station Lines](#)
- [Lines From Points, Direction and Distance](#)
- [Points Along Polylines](#)
- [Points To Pie Segments](#)

[Linear Referencing](#)

- [Create Routes](#)
- [Calibrate Routes](#)
- [Locate Points](#)
- [Locate Polygons](#)
- [Dissolve Events](#)
- [Concatenate Events](#)
- [Union Events](#)
- [Intersect Events](#)

Point Z (M) To Point

[Go to ToolBox Implementation](#)

[Go to .NET Implementation](#)

Converts a point Z (M) data set to a point feature class

Inputs:

- A point Z (M) feature layer

Outputs:

- New point feature class
 - New fields added to the point attribute table
 - [ET_ID] - the FID of original points. The values can be used to link the points back to the original points Z(M).
 - [ET_Z] - is added and populated with Z values if the points are Z aware
 - [ET_M] - is added and populated with M values if the points are M aware

Notes :

- The points can be converted back to Points Z(M) using Point To Point Z(M) Wizard

ToolBox implementation

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

Command line syntax

ET_GPPointZMToPoint <input_dataset> <out_feature class>

Parameters

Expression	Explanation
<input_dataset>	A PointZ(M) feature class or feature layer
<out_feature class>	A String - the full name of the output feature class (A feature class with the same full name should not exist)

Scripting syntax

ET_GPPointZMToPoint (input_dataset, out_feature class)

See the explanations above:

<> - required parameter

{ } - optional parameter

.NET implementation

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

AggregatePolygons(pInFC As IFeatureClass, sOutFName As String, dAggregateTol As Double, Optional dAreaTol As Double = 0) As IFeatureClass

[Open Table of Contents](#)

NOTE: ET GeoWizards 9.6 and above offer a single function that implements the entire process.

Contents:

- [The purpose](#)
- [ET GeoWizards functions to be used](#)
- [The Task](#)
- [The Solution Proposed](#)
- [How to get it done ?](#)
- [A diagram of the process](#)

The purpose:

Many questions similar to the one below have been asked on the ArcGIS forums:

"I am trying to divide county polygons into pieces, and I want to divide it using a road. (i.e. make 2 polygons out of each county, 1 which is north of I-80, and one which is south of I-80, which goes through all of the counties in my dataset). Seems like a relatively simple operation, but I can't figure out how to do it with ArcMap Editor. I thought I'd be able to do this with the GeoProcessing wizard, but I can't, since one is a line shapefile, and one is a polygon shapefile."

Some of the answers (the only ones that gave some sort of a solution) were as follows:

*"- select the polygon to be split
- change the task to Cut Polygon Features
- use the sketch tool
- right-click on the road which will split the polygon
- choose Replace Sketch
- use F2 or Finish Sketch"*

The answer above might do the job, but ONLY if a road goes through the whole county without having an intersection with another road. In other words you need a single polyline that has its ends out of the polygon to be split in order to use the above mentioned approach.

How do you feel about using the above procedure for splitting all the counties in a state with each road of the road network - ONE BY ONE ? Another question is how successful this will be - not that many roads cross a county without intersecting another road.

This document describes a procedure that will help you to split a polygon dataset with a polyline dataset using an ArcView license and the functions available in ET GeoWizards.

ET GeoWizards functions to be used

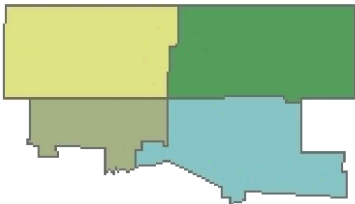
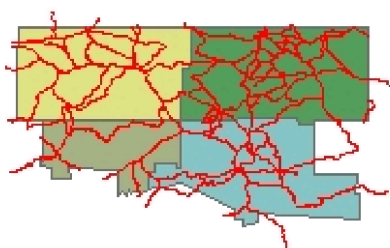
- [Polygons To Polylines](#)
- [Merge Layers](#)
- [Clean Polylines](#)

- [Build Polygons](#)
- **Spatial Join - Standard ArcGIS function**

Note that the links above are to the functions available via the interface of ET GeoWizards. If the procedure is to be performed in the Model Builder - refer to the corresponding tools available in the ET GeoWizards ToolBox. If performed via a VBA script - refer to ET GeoWizards Scripting

The task

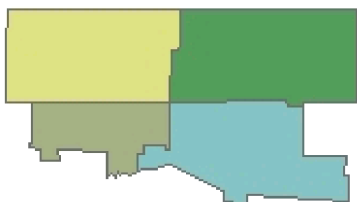
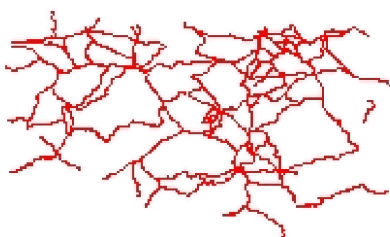
We have four US counties and the highway network in the same area. The goal is to split (divide) the county polygons based on the highway network.

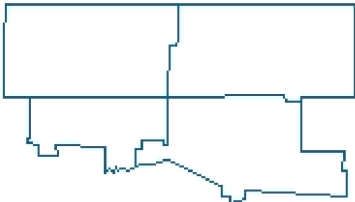
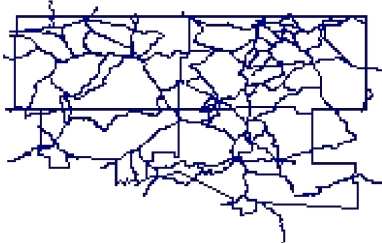

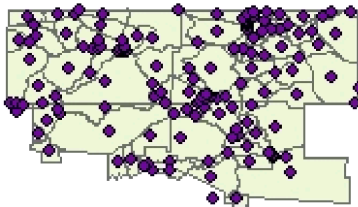
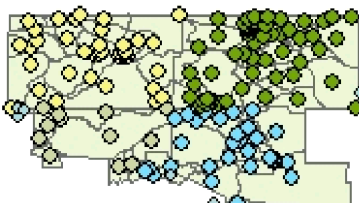
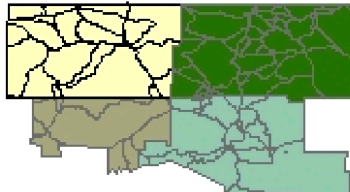
Original polygons (few US counties)	Split Polylines (some US highways)
	

The Solution proposed:

We have the original polygons (counties) and the polylines to be used for splitting (highways). What if instead of trying to split the county polygons with the highway polylines we go a step back - get the counties boundaries (Polylines), merge them with the highways (Polylines) and use the merged dataset to build a brand new polygon dataset. Then we can get the original attributes from the County polygons to the resulting dataset with a simple Spatial Join. The Process step by step:

1. Convert the original polygons to polylines -Polygons To Polylines function
2. Merge the polylines from the previous step with the split polylines - Merge Layers function
3. Build polygons from the merged dataset (the Clean option must be used) - Build Polygons function
4. Create label points for the newly built polygons - Polygon To Point function (label option)
5. Use Spatial Join (polygons to points) to get the attributes of the original polygons to the label points
6. Use Spatial Join (points to polygons) to get the attributes from the label points to the polygons (created in point 3 above)

Original Polygons	Highway Polylines
	

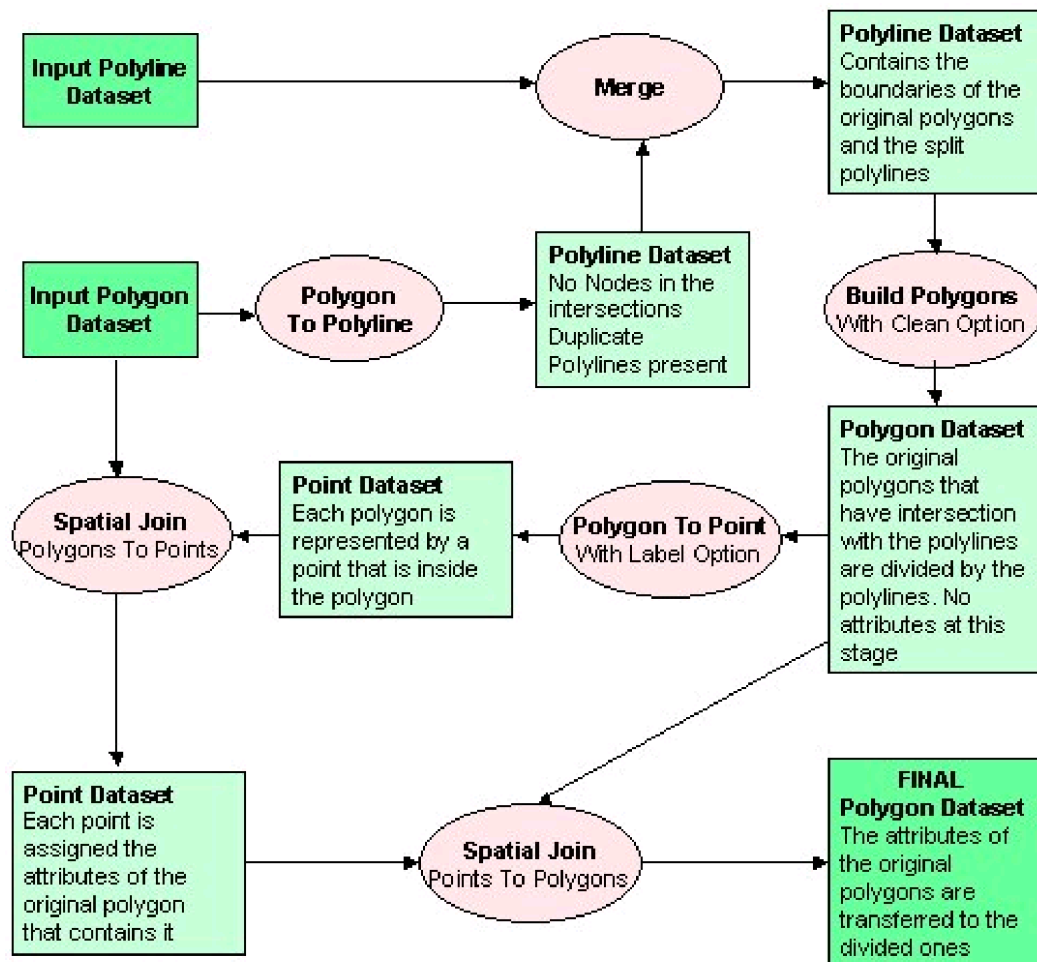
Polygons To Polylines (Get the boundaries of the original polygons as polylines)	Merge (County polylines with Highways)
	
Build Polygons (with clean option) No attributes at this stage	Polygon To Points (with label option) - Get the label points of the polygons created.
	
Get Attributes from original polygons (Spatial Join) and attach them to the label points created.	Get Attributes from the Label Points (Spatial Join) and attach them to the split polygons.
	

How to get it done?

ET GeoWizards offers three ways of achieving the above procedure

- Via the User Interface - just perform the steps one by one using the Wizards available
- With a simple VBA script - many of the functions of ET GeoWizards are available for use within VBA scripts or custom applications written in any COM language. See an example with working code [here](#)
- Create a Geoprocessing Model in the Model Builder (ArcGIS 9.0 and above only) using the ET GeoWizards geoprocessing tools made available in version 9.2. A model performing the task is included in the download of ET GeoWizards for ArcGIS 9.x

A diagram of the process:



See [ET GeoWizards UserGuide](#) for more information

For any comments and enquiries contact: webmaster@ian-ko.com

All ESRI products mentioned are trademarks of Environmental Systems Research Institute, Inc.

Copyright: Ianko Tchoukanski