Model Builder Application

GeoSpatial Services

Ryan Kiefer

Purpose

- Create a Model Builder application which replicates a custom application developed in ArcObjects for pipeline integrity management
  - Creates a Linear Route Event necessary for displaying pipe volume

Introduction

- Geoprocessing
  - Geoprocessing is the processing of geographic information, one of the basic functions of a GIS. It provides a way to create new information by applying an operation to existing data. Any alteration or information extraction you want to perform on your data involves a geoprocessing task. It can be a simple task, such as converting geographic data to a different format, or it can involve multiple tasks performed in sequence, such as those that clip, select, and then intersect datasets.

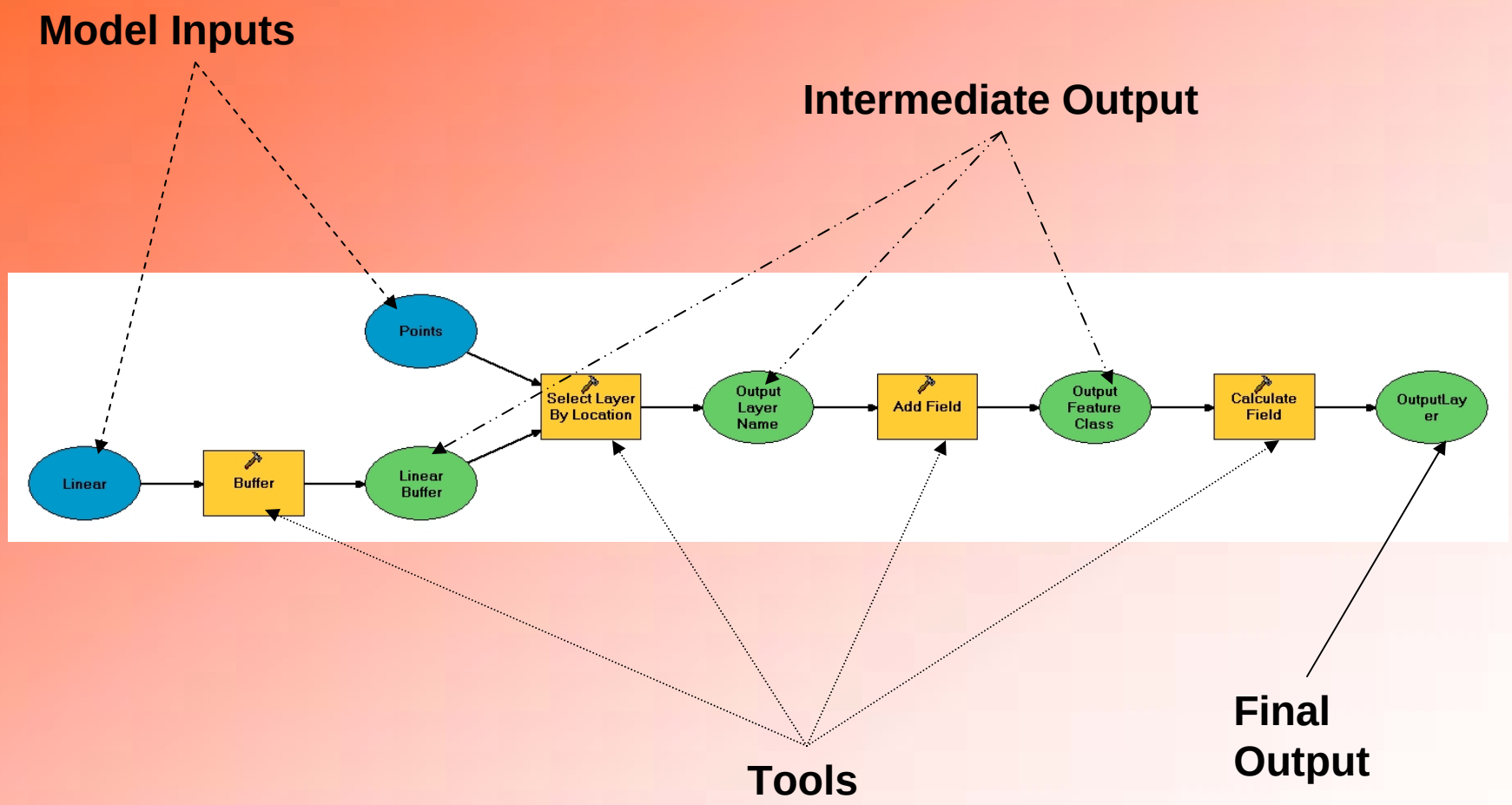
(*ESRI Desktop Help)*

Geoprocessing (Continued)

- Within ArcGIS, you can perform geoprocessing task in a number of ways:
  - Build and run a model that runs a sequence of geoprocessing tools in your work flow. Alter parameter values then rerun the model with a single click.
  - Create and run a script that runs geoprocessing tools. Use system batch processing scripts for repetitive tasks, such as those that run the same tool on multiple inputs, or create your own scripts that run geoprocessing tools.

(*ESRI Desktop Help)*

Model Builder



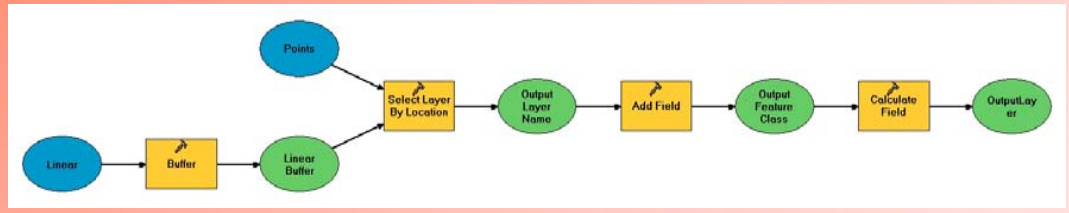
Python

- Python is an Object Oriented Scripting language supported by ESRI for use in Model Builder Scripting

- Why Python?
  - Productivity
    - Reduce code by 1/3 to 1/5
  - Integration
    - “Python code can invoke C and C++ libraries, can be called from C and C++ libraries, can integrate with Java components, can communicate over COM, CORBA, and .NET, and can interact over networks with interfaces like SOAP and XMP-RPC”

      Learning Python, O’Reily Press
  - Open Source
    - Free

Python Example




```
# ---------------------------------------------------------------------------
#  BufferAnalysis.py
#   (generated by ArcGIS/ModelBuilder)
# ---------------------------------------------------------------------------

# Import system modules
import sys, string, os, win32com.client

# Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")

# Set the necessary product code
gp.SetProduct("ArcInfo")

# Load required toolboxes...
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx")


# Local variables...
Linear = "Linear"
Linear_Buffer = "C:/temp/Linear_Buffer.shp"
Points = "Point_Input"
Output_Layer_Name = "Point_Input"
Output_Feature_Class = "Point_Input"
OutputLayer = "Point_Input"

# Process: Buffer...
gp.Buffer_analysis(Linear, Linear_Buffer, "500.000000 Meters", "FULL", "ROUND", "ALL", "")

# Process: Select Layer By Location...
gp.SelectLayerByLocation_management(Points, "INTERSECT", Linear_Buffer, "", "NEW_SELECTION", )

# Process: Add Field...
gp.AddField_management(Output_Layer_Name, "INTERSECT", "TEXT", "", "", "", "", "NON_NULLABLE", "NON_REQUIRED", "", )

# Process: Calculate Field...
gp.CalculateField_management(Output_Feature_Class, "INTERSECT", ""TRUE"", )
```

Model Builder vs. ArcObjects VBA

Cons

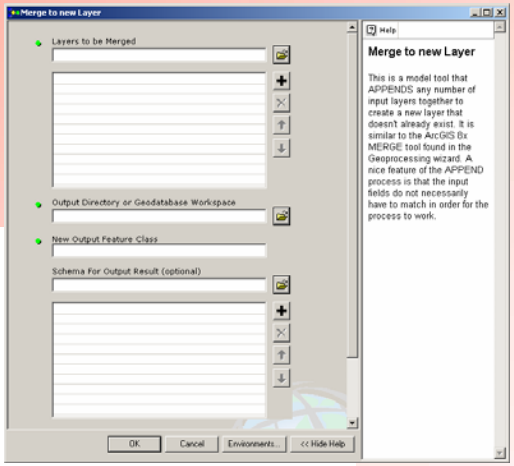
- Different Language
  - Language similar to Perl and Java
    - Differences in syntax
    - Case sensitive
- Limitations with Objects
  - Limited to Model Builder Object Model Diagram

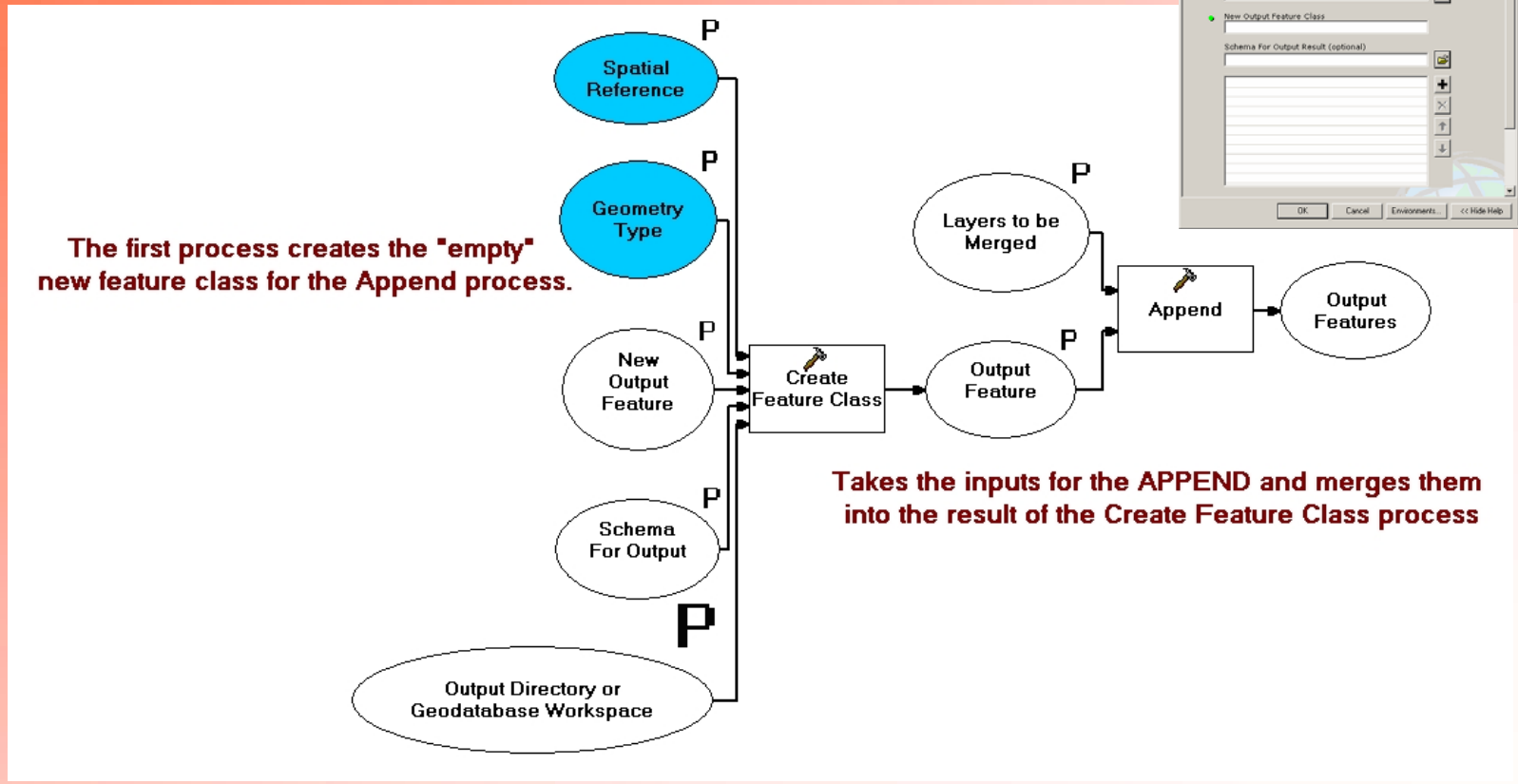
Cons (Continued)

- Limited with Objects (Cont…)
  - Have to create workarounds using only the tools and Objects that are accessible in the Geoprocessor
    - In ArcObjects you create your own path
    - In Model Builder you choose from the given paths
  - Can create custom Model Builder tools in VB though programmatically more difficult
- User forum
  - Less information found in the ESRI User Forum

Cons (Continued)

- NO “Merge”





Model Created by ESRI

Bugs

- Extract Values to Point (Tool)

Pros

- Code Reduction
  - Application created using ArcObjects was over 25 pages of code
  - Application in Model Builder used the Geoprocessing tools and two scripts which had less than 250 lines of code each
    - Locate Features Along Routes
- Time
  - Less code = less time

Pros (Continued)

- Visual Reference
  - Model Builder allows a visual flow through processes of an application
    - Easier to grasp flow of program
    - Easier to change the model using tools visually
    - Easier to share workflow
  - Great for beginning application developers

Pros (Continued)

- Greater intrinsic functionality to Python
  - Arrays
  - Strings

Demo….

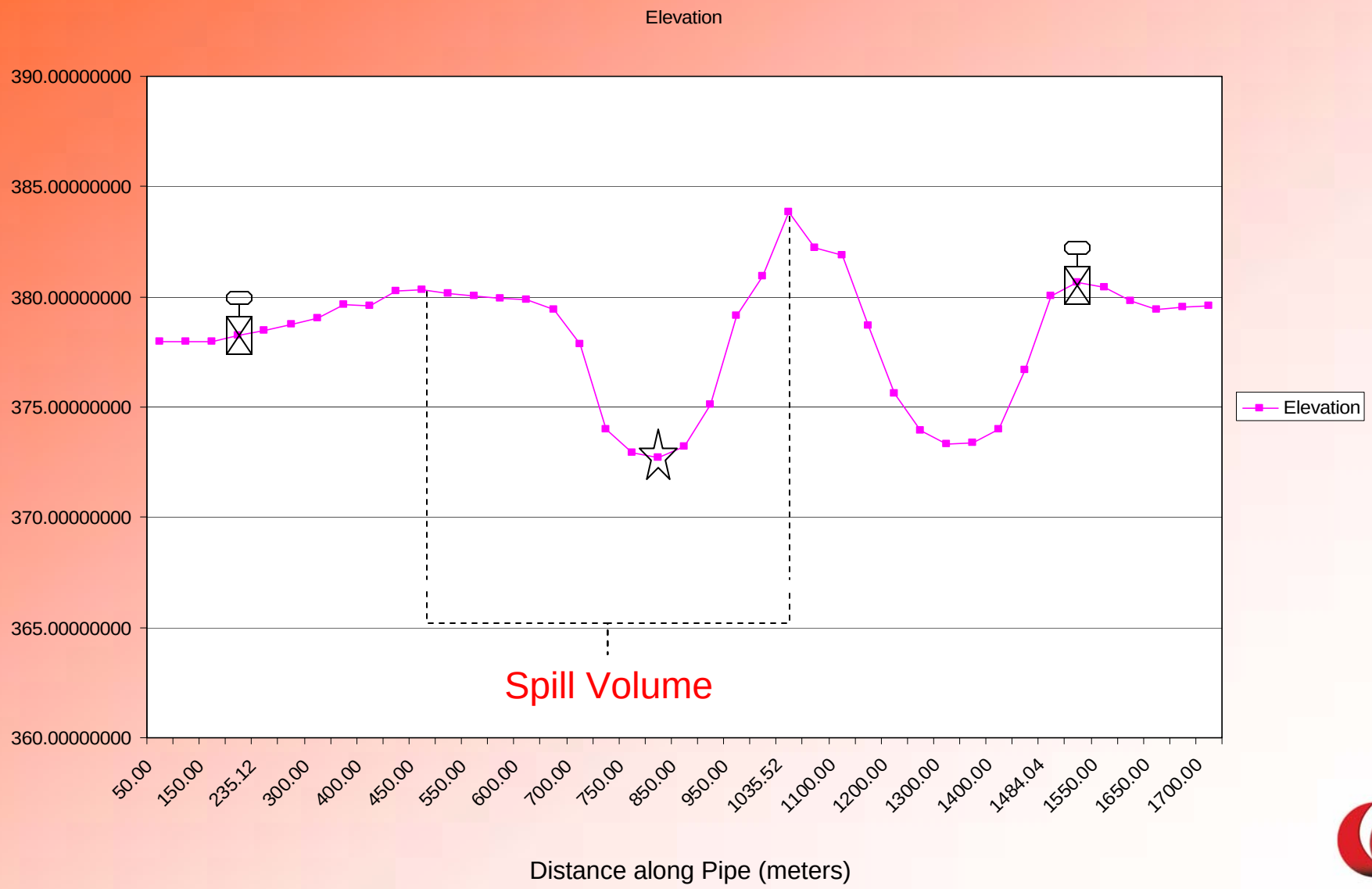
- Input Route Layer
- Input User Specified Interval
- Input Benchmarks
- Input Raster Layer (DEM)
- Input Pipe Diameter (Inches)

- Final Output: Linear Route Event Layer

Acknowledgments

- Dave Hajoglou
- Elizabeth McCord
- Chad Richtman
- Patrick Thorsel
- Lane Urtel

Future Analysis



Questions?

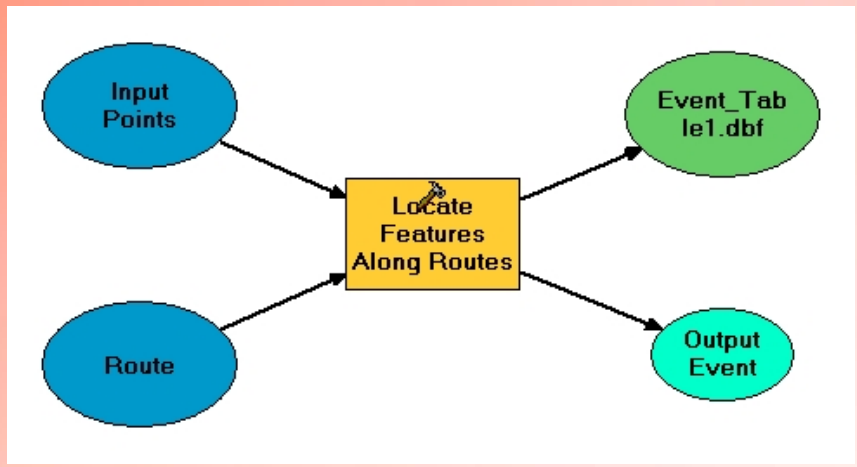
Locate Points Along Routes

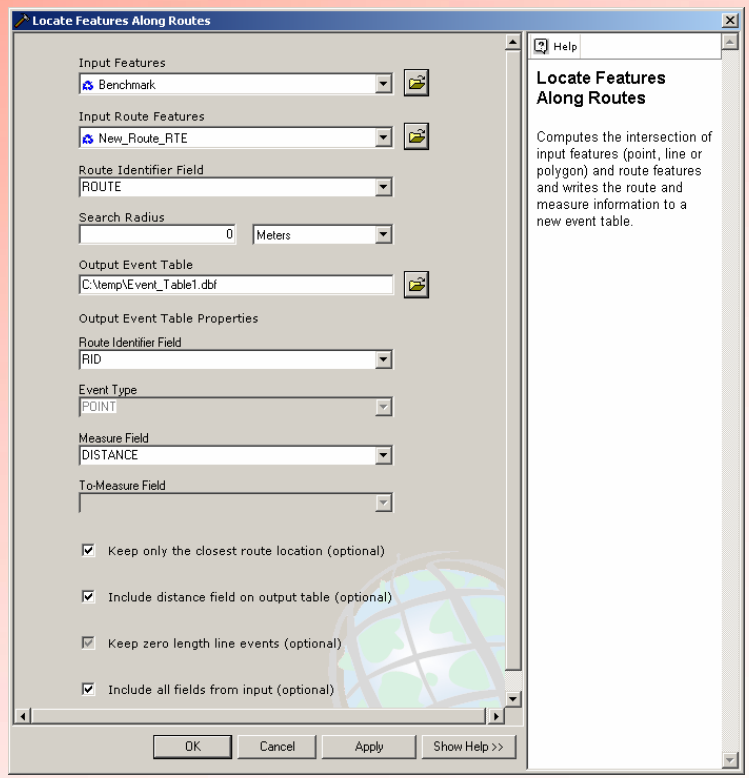
```
Public Sub PointsOnRoute(pFClass As IFeatureClass)

  Dim newWSP As IWorkspace
  Dim newFWS As IFeatureWorkspace
  Dim newWSF As IWorkspaceFactory

  Set newWSF = New ShapefileWorkspaceFactory
  Set newWSP = newWSF.OpenFromFile(savePath, 0)
  Set newFWS = newWSP

  Dim pTempDS As IDataset
  Set pTempDS = newWSP

  Dim pOutDSN As IDatasetName
  Set pOutDSN = New tableName

  Dim pOutWSN As IWorkspaceName
  Set pOutWSN = pTempDS.FullName
  Set pOutDSN.WorkspaceName = pOutWSN
  pOutDSN.Name = "Event_1"


  Dim pDS As IDataset
  Dim pNameRt As IName
  Dim pRMLName As IRouteLocatorName
  Set pDS = pFClass1
  Set pNameRt = pDS.FullName
  Set pRMLName = New RouteMeasureLocatorName
  With pRMLName
    Set .RouteFeatureClassName = pNameRt
    .RouteIDFieldName = "RID"
    .RouteIDIsUnique = True
    .RouteMeasureUnit = esriFeet
    '.RouteWhereClause = "" '+++ used to limit the number of routes
  End With

  Dim pRtProp As IRouteEventProperties2
  Dim pRMPtProp As IRouteMeasurePointProperties2
  Set pRtProp = New RouteMeasurePointProperties

  With pRtProp
    .EventMeasureUnit = esriUnknownUnits
    .EventRouteIDFieldName = "RID"
    '.LateralOffsetFieldName = "offset"
    .AddErrorField = True          'add field for locating errors
    .ErrorFieldName = "LOC_ERROR" 'specify name for the locating errors field
  End With

  '+++ IRouteMeasurePointProperties2 is used to include an angle field to the
        route event source.
    '+++ The angle field can be used to cartagrophical rotate point event
        symbology.
    Set pRMPtProp = pRtProp
    With pRMPtProp
      .MeasureFieldName = "DISTANCE"
      .AddAngleField = True
      .AngleFieldName = "LOC_ANGLE"
      .AsPointFeature = True       'point events shape will be of type Point.
                                                          Multipoint if False
      .NormalAngle = True          'the angle normal to the digitize direction.  Will
                                                          be tangent if False
      .ComplementAngle = False
    End With

    Dim pTempName As IName
    Set pTempName = pRMLName

    Dim pRouteLocator As IRouteLocator
    Set pRouteLocator = pTempName.Open


    Set pFeatLayerValves = pLayerValves
    Dim pTempFClass1 As IFeatureClass
    Set pTempFClass1 = pFeatLayerValves.FeatureClass
    Dim pRouteLocatorOps As IRouteLocatorOperations
    Set pRouteLocatorOps = New RouteLocatorOperations

      With pRouteLocatorOps
        Set .InputFeatureClass = pTempFClass1
        Set .RouteLocator = pRouteLocator
      End With

    Set pTable = pRouteLocatorOps.LocatePointFeatures(0.01, False, pRtProp,
          False, pOutDSN, "", Nothing)

    'add table to TOC

    Call addfields(pTable)
    Call AddToMap(pTable)

End Sub
```

Developed by ESRI

Locate Points Along Routes

Entire sequence of ArcObjects code now in a convenient Geoprocessing Tool





Back

Syntax

- **Python**

```
#Import standard library modules
import win32com.client, sys, os
#Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")

inFc = sys.argv[1]   #Pass in argument for FeatureLayer input

All_Fields = gp.ListFields(inFC)
oneField = All_Fields.Next()

  #Loop through all fields
while oneField:
    print oneField.Name
    oneField = All_Fields.Next()
```

- **ArcObjects (from Layer in TOC)**

```
Public Sub GetFields()
  Dim pMxdoc As IMxDocument
  Set pMxdoc = Application.Document

  Dim pMap As IMap
  Set pMap = pMxdoc.FocusMap

  Dim pFlayer As IFeatureLayer
  Set pFlayer = pMap.Layer(0)

  Dim pFclass As IFeatureClass
  Set pFclass = pFlayer.FeatureClass

  Dim pFields As IFields
  Set pFields = pFclass.Fields

  Dim pField As IField
  Dim i As Integer

    'Loop through all fields
  For i = 0 To (pFields.FieldCount – 1)
      Set pField = pFields.Field(i)
      MsgBox pField.Name
  Next

End Sub
```

Back