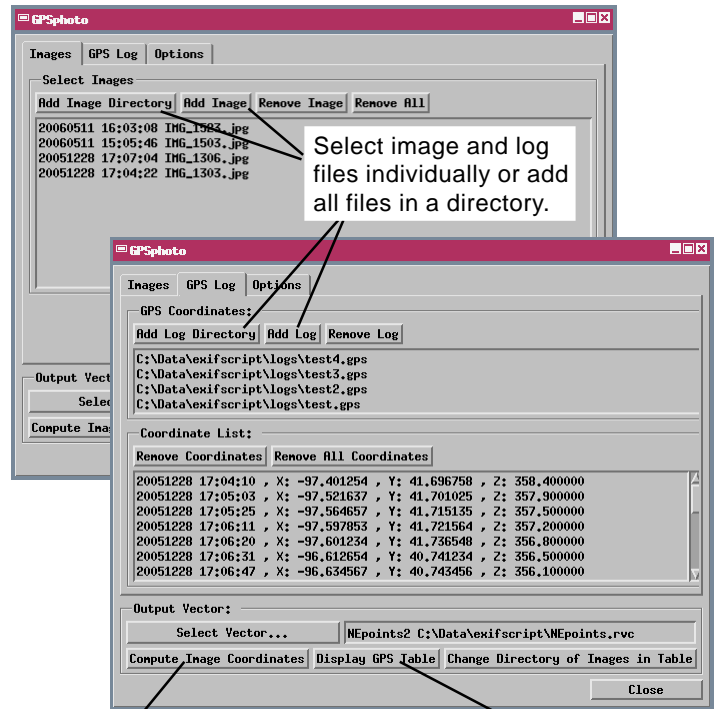


# Sample Geospatial Script

## Map Digital Photo Locations from GPS Logs

Digital photos acquired during a GPS survey can provide a visual record of important features along the survey route. But for this record to be meaningful, each photo file should be linked to the geographic coordinates where it was acquired. At this time a few high-end digital cameras incorporate a GPS card or can be connected to an external GPS to directly record location coordinates with each image file. However, digital cameras do record the date and time each photo was taken in the Exchangeable Image File (EXIF) header of the photo's JPEG file. By comparing a photo's EXIF date and time tags with the times recorded in a GPS track log, the location where the photo was taken can be determined.

MicroImages has prepared a sample geospatial script (GPSphoto.sml; excerpted on the reverse side of this plate) that automatically determines locations for any number of digital photo files using one or more GPS track logs. (As an example, this script was written to process GPS logs in the text format recorded by the TNT products, but it can be adapted to accommodate any track log text format that includes date, time, and geographic coordinates for each reading.) The script creates an output vector object with a point element for each photo location. Records in an attached database table store each photo's date and time, its geographic coordinates, and the directory path and file name of the photo file on your hard drive. (The database field containing the photo filename and directory path could be accessed by a display control script to pop in a display of the photo when the mouse cursor pauses over its point in the displayed vector object. The SpyglassView sample script described in the color plate entitled *Sample GraphTip Script: Spyglass View* could be modified



After selecting your images and GPS log(s) and setting processing options, press the Compute Image Coordinates button to create a location point for each photo file in the selected output vector object.

Press the Display GPS Table button to open a tabular view of the vector point database table that records each photo's geographic location and location in your directory structure.

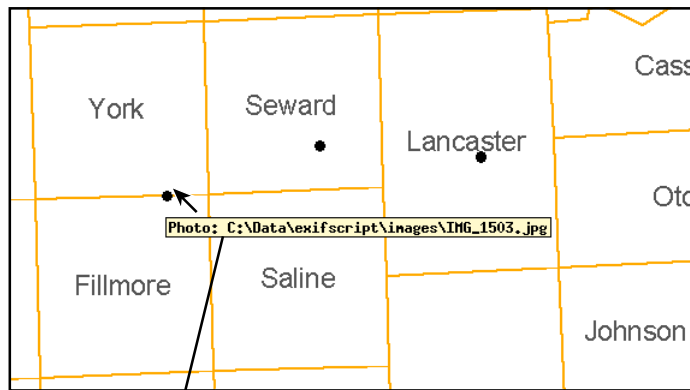
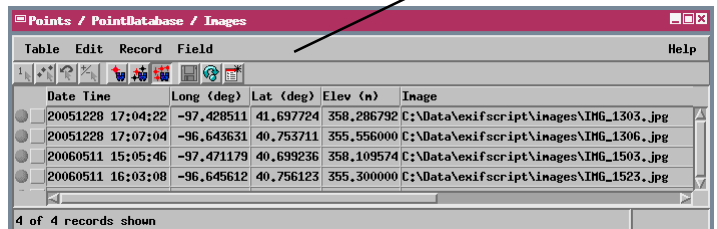


Photo-location points created by the GPSphoto script (solid black circles) overlaid on a map of counties. Point DataTip has been set to show the filename and directory path stored in the point database; the corresponding photo for the displayed DataTip is shown at left.

to provide this capability.) The GPSphoto script also provides the option to write the computed location information to each photo's EXIF header.

The Options panel on the script's control window allows you to set various processing options. You can choose to interpolate photo coordinates using the pair of GPS records with the closest times to the photo, or simply assign coordinates from the closer of the two GPS records. You can set maximum allowed differences in distance and time between GPS points and photos and optionally reject photos falling outside these constraints. You can also specify a time offset to compensate for any difference between the camera's clock time and the GPS times. (To establish this offset, photograph the time display on the GPS unit while taking the survey and later compare it with the image time.)

Many sample scripts have been prepared to illustrate how you might use the features of the TNT products' scripting language for scripts and queries. These scripts can be downloaded from [www.microimages.com/downloads/scripts.htm](http://www.microimages.com/downloads/scripts.htm).

## Script Excerpts for Mapping Photo Locations (GPSphoto.sml)

Function to read EXIF header from image "filename\$". Adds string containing date and time (YYYYMMDD HH:MM:SS) to imageList-Time string list. Returns 0 if it cannot find a header, 1 if successful.

```
func GetEXIF(string filename$) {
  class STRINGLIST keys;
  exifhandle.Open(filename$);
  keys = exifhandle.GetKeyList(); EXIF class function to return stringlist of exif keys
  pass the name of a Key to GetDatumStr and returns the value as a string
  string exifDateTime$ = exifhandle.GetDatumStr("Exif.Image.DateTime");

  key "Exif.Image.DateTime" contains date/time value (YYYY:MM:DD HH:MM:SS);
  parse string and add to EXIFDateTime$ string as YYYYMMDD HH:MM:SS

  local string EXIFDateTime$=sprintf("%s%s %s",exifDateTime$.substr(0,4),
  exifDateTime$.substr(5,2),exifDateTime$.substr(8,2), exifDateTime$.substr(11,8));

  if no value for key "Exif.Image.DateTime" then check key "Exif.Photo,DateTimeOriginal"

  if (GetToken(EXIFDateTime$, ",", 1) == "" || GetToken(EXIFDateTime$, ",", 1) == "") {
    exifDateTime$ = exifhandle.GetDatumStr("Exif.Photo.DateTimeOriginal");
    EXIFDateTime$=sprintf("%s%s %s",exifDateTime$.substr(0,4),
    exifDateTime$.substr(5,2),exifDateTime$.substr(8,2),exifDateTime$.substr(11,8));
  }

  if no value for either key, return 0 as unsuccessful

  if (GetToken(EXIFDateTime$, ",", 1) == "" ||
  GetToken(EXIFDateTime$, ",", 1) == "")
    return 0;

  print("Added: ",EXIFDateTime$, filename$);
  imageListTime.AddToEnd(EXIFDateTime$); add time to string list
  return 1; return 1 as successful
}
end func GetEXIF
```

Procedure to open log file "logname\$" and parse the results. Adds string containing Time(YYYYMMDD HH:MM:SS), X, Y, Z to stringlist. Format of standard MI log file: date (YYYYMMDD), time (HHMMSS), XPos (deg), YPos(deg), Elev(m), XVel(m/s), YVel(m/s), ZVel(m/s), Head(deg), Speed(m/s), DataSrc, NumSat.

```
proc ParseLog(class STRINGLIST slist, string logname$) {
  local string line$, date$, timeToken$, time$;
  local numeric x,y,z=0;
  class FILE logfile;
  logfile=fopen(logname$); open logfile

  while (!feof(logfile)) while not reached end of file
  {
    line$=fgetline$(logfile); get next line in log file
    date$=GetToken(line$,"",1); parse first entry of line (date)
    if (StrToNum(date$) > 1) {
      timeToken$=GetToken(line$,"",2); parse time entry
      change HHMMSS to HH:MM:SS
      time$=sprintf("%s:%s:%s", timeToken$.substr(0,2),
      timeToken$.substr(2,2), timeToken$.substr(4,2));

      x=StrToNum(GetToken(line$,"",3)); parse XPos, YPos, and Elev entries of line
      y=StrToNum(GetToken(line$,"",4));
      z=StrToNum(GetToken(line$,"",5));
      line$ = sprintf("%s %s , X: %.6f , Y: %.6f , Z: %.6f", date$, time$, x, y, z);
      slist.AddToEnd(line$); add date, time, x, y, z to string list
    }
  }
  fclose(logfile); close logfile
}
end proc ParseLog
```

Procedure to create new GPS table.

```
proc CreateTable() {
  if (ObjectExists(filename$, obj$, "Vector") == 0) check if vector exists
  { create vector
    CreateVector(GPSVector, filename$, obj$, desc$, "3DVector");
    crs.Assign("Geographic2D_WGS84_Deg"); set crs for vector
    CreateImpliedGeoref(GPSVector, crs);
  }
  check if vector exists
  if (ObjectExists(filename$, obj$, "Vector") == 1)
    OpenVector(GPSVector, filename$, obj$); open vector

  dbase= OpenVectorPointDatabase(GPSVector); open point database

  if ( TableExists(dbase,IMAGETablename$) == 0) table does not exist
  { create table
    table=TableCreate(dbase, IMAGETablename$, tabledesc$);
    TableAddFieldString(table, "Date Time", 17, 17);
    TableAddFieldFloat(table, "Long (deg)", 9,6);
    TableAddFieldFloat(table, "Lat (deg)", 9,6);
    TableAddFieldFloat(table, "Elev (m)", 9,6);
    TableAddFieldString(table, "Image", 200,100);
  }
  add time, X, Y, Z, image fields
}
end proc CreateTable()
```

Procedure to attach table records to points in vector; call each time new record added.

```
proc AttachPoints() {
  local array numeric recordarray[100]; record array
  local array numeric elementarray[1]; element array
  local array numeric writearray[1]; records to write array
  local numeric numberOfRecords, numberOfElements,
  elementnum, recordnum,x,y,z,i;

  dbase= OpenVectorPointDatabase(GPSVector); get dbase
  table=DatabaseGetTableInfo(dbase, IMAGETablename$); get table class

  elementnum=1; set to first element
  numberOfElements=NumVectorPoints(GPS Vector);
  numberOfRecords=table.NumRecords;

  clear all current attachments
  for i=1 to numberOfRecords add all records to record array
  recordarray[i]=i;
  for i=1 to numberOfRecords { for every element in vector
    elementarray[1]=i;
    TableRemoveAttachment(table, elementarray[1], remove all records from each element
    recordarray, numberOfRecords);
  }

  for recordnum=1 to numberOfRecords { for all records
    x=TableReadFieldNum(table, "Long (deg)", recordnum);
    y=TableReadFieldNum(table, "Lat (deg)", recordnum);
    z=TableReadFieldNum(table, "Elev (m)", recordnum); read in coordinates
    change vector point to coordinates of record
    VectorChangePoint(GPSVector, elementnum, x, y, z);
    add current record to array of records to write; make attachment
    writearray[1]=recordnum;
    TableWriteAttachment(table, elementnum, writearray, 1, "point");

    if(TableReadFieldStr(table, "Date Time",recordnum) !=
    TableReadFieldStr(table, "Date Time", recordnum+1))
      elementnum++; if not duplicate records, move to next point
  }
}
end proc AttachPoints()
```