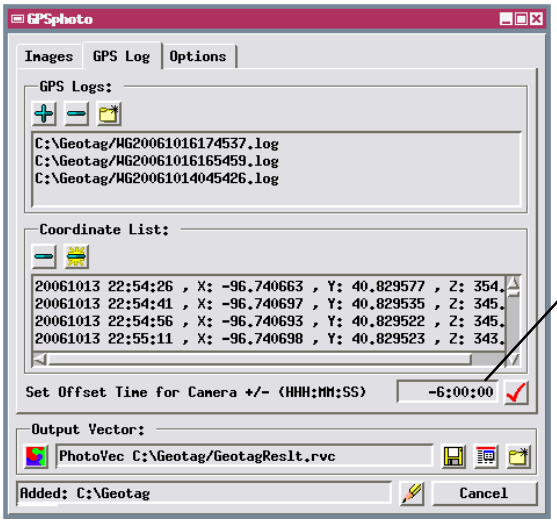
Sample Geospatial Script

Add Geotags to Digital Photos

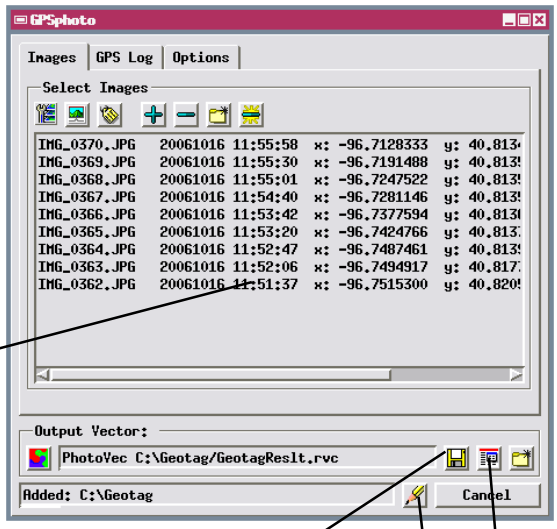
MicroImages has prepared a sample geospatial script that automatically determines locations for any number of digital photo files (*.jpg) using the date and time of acquisition recorded in each photo's Exchangeable Image File (EXIF) header and one or more GPS track logs. The sample script (GPSphoto2.sml, excerpted on the reverse side of this plate) also creates a vector object with a point element for each geotagged photo's location. Records in an attached database table store each photo's date and time, geographic coordinates, and the directory path and file name of the photo file.

The dialog window created by the script allows easy selection of images, GPS logs, and processing options. You can choose to interpolate photo coordinates using the pair of GPS records in the log file with times closest to that of the photo or simply assign coordinates from the GPS record with the closest time. If GPS logs are not available, you can manually enter coordinates or obtain them visually using a point tool in a view of a georeferenced raster opened by the script. This sample script could be modified to provide other options for interpolating GPS logs, to accommodate other photo file formats (e.g. *.tif), or to produce geotag output in different forms (such as records in a standalone database table).

The photo geotagging script uses a GPSDBASE class in SML to carry out many core processing tasks. This class has a method to read and parse GPS logs in NMEA or MicroImages format, maintains internal lists of the selected logs and log records, and has a method for computing coordinates from the log records for a given date and time (in this case read from a photo's EXIF header). Other methods are provided to allow manual parsing and record reading from non-standard log formats, providing further opportunities for customization and extension of this sample script.

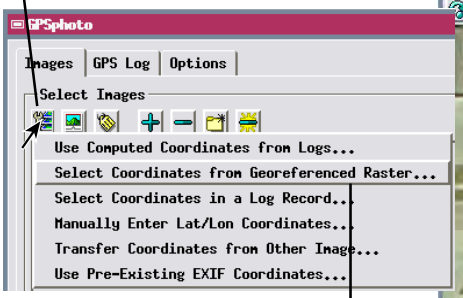


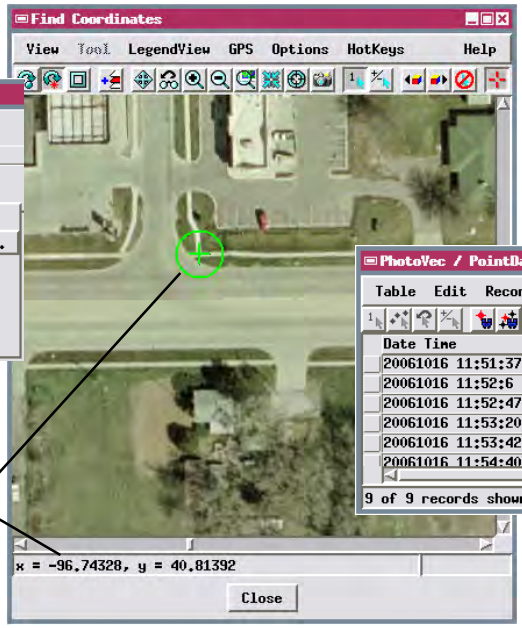
(1) You can set a time offset to adjust for any time reference difference between the camera and GPS, then press the Apply Changes button to update the coordinates in the image list.



(2) Image geotag coordinates are computed automatically and shown in the image list whenever you add GPS logs or images to the lists.

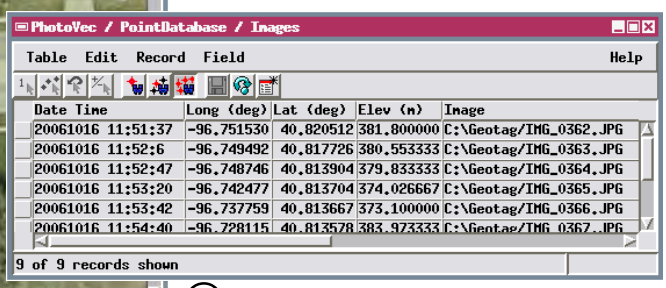
(3) The menu opened by the Image Options icon button on the Image panel allows you to choose alternative sources for coordinates for individual images, including existing EXIF tags in the image file and manual entry. The *Select Coordinates from Georeferenced Raster* option on this menu prompts you to choose a georeferenced raster object and automatically opens a view window displaying the raster. This view has a point tool that you can place and right-click to set the map coordinates for the photo currently highlighted in the Select Images list.





(4) Press the Save Vector icon button to save the current photo positions as vector points with attached database table.

(5) Press the Write EXIF button to write the current image coordinates to the appropriate EXIF tags in each image file's EXIF header.



(6) The Display GPS Table button opens a view of the database table with a record for each geotagged photo attached to the vector point marking its location.

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.

Script Excerpts for Adding Geotags to Digital Photos (GPSphoto2.sml)

Procedure to compute coordinates for images

```
proc ComputeCoordinates(numeric imageNum, numeric computedflag,
      numeric errorreportflag) {

   class POINT3D leftXYZ, rightXYZ, newXYZ;

   local string report$;
   local numeric i, listcount=0;

   local string method$;
   local numeric cameraoffset, timeoffset=0;        options

   local string inputname$;
   class DATETIME inputtime;                        image time and name
   class GPSDATA inputdata;

   if (imageNum == -1) {
      imageListNameAttached.Clear();
      }

   GetOptions(cameraoffset, method$, timeoffset);
   timeoffset=max allowed difference in time


   local numeric startcount, endcount;
   if(imageNum==-1) {
      endcount = imageListName.GetNumItems()-1;
      startcount=0;
      }
   else
      {
      endcount = imageNum;
      startcount=imageNum;
      }
                                                    for all images in
                                                    image list box
   for listcount=startcount to endcount
      {
         get input name, date, and time
      inputname$ = imageListName.GetString(listcount);
      inputtime = imageListTime[inputname$];
      inputdata = imageListData[inputname$];
         check for existing coords
      local string imageName$=inputname$;
      class POINT3D coord;
      class STRINGLIST keys;
      local numeric keyindex = 0;
      local numeric valid=0;

      exifhandle.Open(imageName$);
      keys = exifhandle.GetKeyList();

      for keyindex=0 to keys.GetNumItems()-1{
         if (keys[keyindex]=="Exif.GPSInfo.GPSLongitude")
            valid++;
         if (keys[keyindex]=="Exif.GPSInfo.GPSLongitudeRef")
            valid++;
         if (keys[keyindex]=="Exif.GPSInfo.GPSLatitude")
            valid++;
         if (keys[keyindex]=="Exif.GPSInfo.GPSLatitudeRef")
            valid++;
         if (keys[keyindex]=="Exif.GPSInfo.GPSAltitude")
            valid++;
         if (keys[keyindex]=="Exif.GPSInfo.GPSAltitudeRef")
            valid++;
         }
```

[code for parsing coordinate information from the photo EXIF header is omitted here for brevity.]

```
   else if ( (Assigned[inputname$]!=1 || IsNull(Assigned[inputname$])) || computedflag==1)
      {
      Assigned[inputname$]=0;
      imageListData[inputname$].Position.x = 0;
      imageListData[inputname$].Position.y = 0;
      imageListData[inputname$].Position.z = 0;
         compute coordinates using GPSDBASE class method
      class GPSDATA data;
      gpsdbase.Compute(inputtime, data, method$, timeoffset);
      imageListData[inputname$] = data;

      if (!IsNull(data.Position.x)  && !IsNull(data.Position.y)  && !IsNull(data.Position.z))
         {
         imageListNameAttached.AddToEnd(inputname$);
         imageListData[inputname$] = data;
         }

      else if (errorreportflag == 1)        image coordinates not computed
         report$=report$ + sprintf("Failed to compute coordinates for image '%s' \n",
                  inputname$);

      }     end if Assigned

      else
         {
         imageListNameAttached.AddToEnd(inputname$);
         imageListData[inputname$] = inputdata;
         }
      }     end for listcount

      if (report$ != "")     report$ contains images with no coordinates
         {
         report$=report$ + sprintf("These Images fall outside log range.");
         PopupMessage(report$);
         }
   }           end proc ComputeCoordinates


proc CreateTable() {        Procedure to create new GPS table
   DatabaseCreate
   if (ObjectExists(filename$, obj$, "Vector") == 0)    check if vector exists
      {
      CreateVector(GPSVector, filename$, obj$, desc$, "3DVector");      create vector
      CreateImpliedGeoref(GPSVector, crs);
      }

   if (ObjectExists(filename$, obj$, "Vector") == 1)    check if vector exists
      OpenVector(GPSVector, filename$, obj$);           open vector

   dbase= OpenVectorPointDatabase(GPSVector);           open point dbase

   if ( TableExists(dbase,IMAGEtablename$) == 0)        table does not exist
      {
      table=TableCreate(dbase, IMAGEtablename$, 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);
      }                                                 create table,
                                                        add time, x, y,
                                                        z, image fields
   }           end proc CreateTable()
```