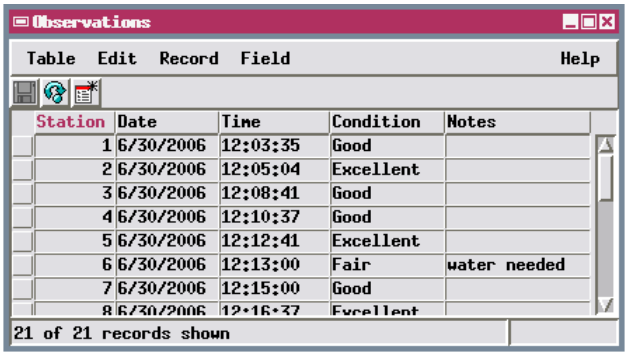
Sample Geospatial Script

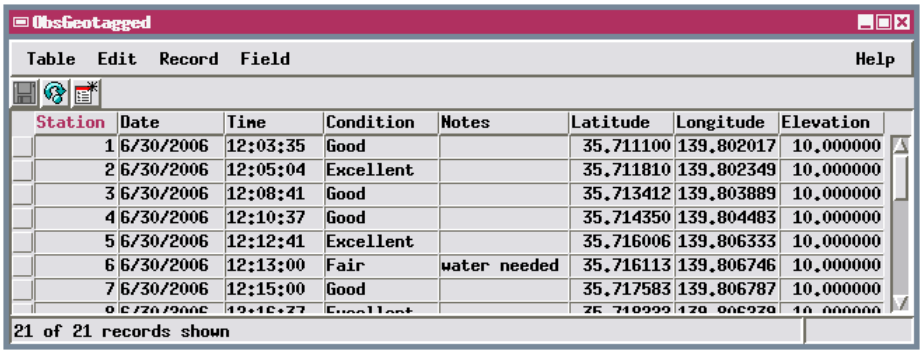
Add Geotags to Database Records

Location-specific field observations are important in many disciplines and can be recorded in varied database formats. If the observation records include date and time information, and a concurrent GPS track log is available, the date/time information in the track log can be used to assign map coordinates for each record.

MicroImages has created a sample geospatial script (excerpted on the reverse side of this plate) that performs this geotagging operation on a database table using a concurrent GPS track log file. The database table must be selected from a standalone database object in a MicroImages Project File, but the table can be either in the internal MicroImages format or be a link to a table in an external database such as Access or PostgreSQL. The Geotag Database script performs the geotagging on a copy of the source table; the result table is placed in the same Project File database object as the source table. The script adds fields for Latitude, Longitude, and Elevation to the result table, computes values for those fields, and writes them to the table records.



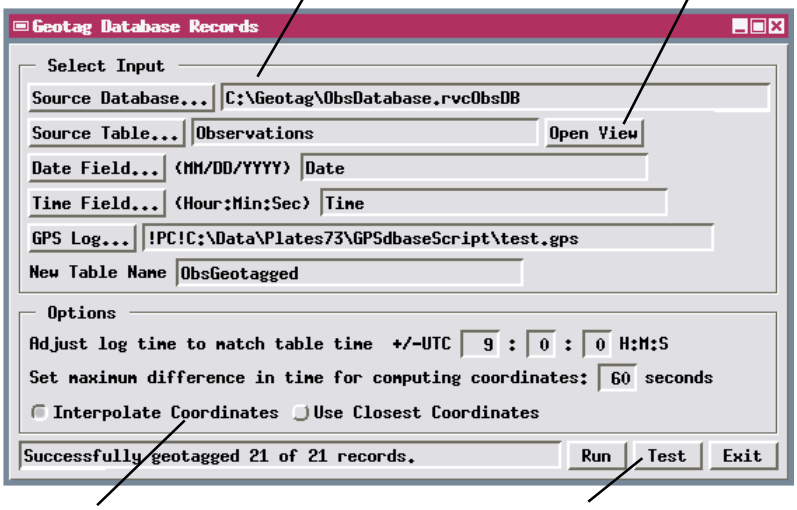
A sample database table with date and time fields (left). This table was processed by the Geotag Database script to produce an output table (below) with position information for each record computed from a GPS track log.



A dialog window is created and opened by the Geotag Database script to simplify selection of the source table, its date and time fields, and the GPS log file. The Open View button to the right of the Source Table text field opens a tabular view of the source table for reference in choosing the date and time fields. An Options section of the dialog provides controls for setting the difference in time reference between the database records and GPS log and the maximum allowed difference in time between database and track log records. You can choose to interpolate coordinates from the nearest (in time) pair of GPS log records or use the coordinates from the closest matching time. A tabular view of the results table is opened automatically when the geotagging operation is completed.

The dialog created by the Geotag Database script simplifies selection of the source table, the fields in that table with the date and time information, and the GPS track log to be used to compute the position coordinates for each record in the table.

The Open View button opens a tabular view of the source table for reference.



Controls on the Options panel set the parameters for processing the date, time, and position information in the input table and the GPS track log.

The Test button reviews the database records and GPS log and reports how many of the records would be geotagged with the current settings.

The Geotag Database script assumes that date and time information is stored in separate string fields in the source table in the date format "month/day/year" and the time format "hour:minute:second". User-defined "getDate" and "getTime" functions in the script read and parse these string values from the source table to provide the required numerical values needed in other parts of the script. These functions can be modified easily to accommodate date and time information in other formats.

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/download/scripts.htm.

Excerpts from Geotag Database Script (GeotagDbase.sml)

Function to get date string from a field in the source database and parse it to get year, month, and day. Returns the Julian date (numeric value). Modify this function to fit the date format in your source table.

```
func getDate(numeric recnum, var class DBTABLEINFO tblSource, string
      srcdatefld$) {
   local class DATETIME dateDT;
   local string date$;
   local numeric year, month, day;

   date$ = TableReadFieldStr(tblSource, srcdatefld$, recnum);
   year = StrToNum( GetToken(date$, "/", 3) );
   month = StrToNum( GetToken(date$, "/", 1) );
   day = StrToNum( GetToken(date$, "/", 2) );

   dateDT.SetDate(year, month, day);
   return dateDT.GetDateJulian();
   }
```

structure to store and convert date/time information

get string from designated date field in source table

parse date string to get year, month, and day

set date in DATETIME class and return value

Function to get time string from a field in the source database and parse it to get hour, min, and sec. Returns the time as seconds since midnight (numeric valus). Modify this function to fit the time format in your source table.

```
func getTime(numeric recnum, var class DBTABLEINFO tblSource, string
      srctimefld$) {
   local class DATETIME timeDT;
   local string time$;
   local numeric hour, min, tsec;

   time$ = TableReadFieldStr(tblSource, srctimefld$, recnum);
   hour = StrToNum( GetToken(time$, ":", 1) );
   min = StrToNum( GetToken(time$, ":", 2) );
   tsec = StrToNum( GetToken(time$, ":", 3) );

   timeDT.SetTime(hour, min, tsec);
   return timeDT.GetTimeSecondsSinceMidnight(1);
   }
```

structure to store and convert date/time information

get string from designated time field in source table

parse time string to get hour, min, and seconds

set time in DATETIME class and return value

Procedure to get processing options from the dialog. Called by Run() and Test() procedures

```
proc GetOptions (var numeric gpsTimeOffset, var numeric maxTimeDiff, var string
      method$) {
   local numeric adjHour = gpsdialog.GetCtrlByID("adjHour").GetValueNum();
   local numeric adjMin = gpsdialog.GetCtrlByID("adjMin").GetValueNum();
   local numeric adjSec = gpsdialog.GetCtrlByID("adjSec").GetValueNum();

   if (adjHour < 0) {
      adjMin = adjMin * -1;
      adjSec = adjSec * -1;
      }

   local numeric gpsTimeOffset = adjHour * 3600 + adjMin * 60 + adjSec;

   local numeric maxTimeDiff =
      gpsdialog.GetCtrlByID("maxDiff").GetValueNum();

   local string method$ = gpsdialog.GetCtrlByID("method").GetValueStr();
   }
```

get gps offset time from dialog

check if hour is negative value and change sign for min and sec

convert gps offset time in H:M:S to seconds for GPSDBASE class

get maximum time difference; used as parameter to GPSDBASE.Compute()

get method (Interpolate or Closest); used as parameter to GPSDBASE.Compute()

end proc GetOptions

Procedure called when Run button is pressed. Copies source table to destination, adds position fields, and loops through records to compute position and write values to the new fields.

```
proc Run () {
   local numeric gpsTimeOffset, maxTimeDiff;
   local string method$;
   GetOptions(gpsTimeOffset, maxTimeDiff, method$);
   gpsdbase.SetOffset(gpsTimeOffset);

   TableCopy(dbSource, tblSource, dbSource);

   local string ctablename$ = srctablename$ + "1";

   tblDest = DatabaseGetTableInfo(dbSource, ctablename$);
   tblDest.Name = desttablename$;

   local class DBFIELDINFO elevInfo;
   TableAddFieldFloat(tblDest, "Latitude", 10, 6);
   TableAddFieldFloat(tblDest, "Longitude", 10, 6);
   elevInfo = TableAddFieldFloat(tblDest, "Elevation", 10, 6);
   elevInfo.UnitType = "length";
   elevInfo.Units = "meters";

   local numeric j;
   local numeric numTagged = 0;

   local class GPSDATA gpsdata;

   local class DATETIME datetime;

   local class POINT3D position;

   for j = 1 to tblDest.NumRecords
      {

      datetime.SetDateJulian( getDate(j, tblDest, srcdatefld$) );
      datetime.SetTimeSecondsSinceMidnight( getTime(j, tblDest, srctimefld$) );

      if (gpsdbase.Compute(datetime, gpsdata, method$, maxTimeDiff) == 0)
         {
         position = gpsdata.position;

         TableWriteField(tblDest, j, "Latitude", position.y);
         TableWriteField(tblDest, j, "Longitude", position.x);
         TableWriteField(tblDest, j, "Elevation", position.z);

         ++ numTagged;
         }
      }

   statusText.SetValueStr( sprintf("Successfully geotagged %d of %d records.",
      numTagged, tblDest.NumRecords) );

   dbDestView = DBEditorCreate(dbSource);
   DBEditorOpenTabularView(dbDestView, desttablename$);
   destTblOpen = 1;
   }
```

get GPS log processing options from dialog using user-defined function

set GPS time offset

copy source table and rename with name previously chosen

copy table; name gets incremented by adding "1"

make string for name of copied table

get table info for copied table

rename table

add fields for Latitude, Longitude, and Elevation to the copied table

loop through records in destination table to compute/add coordinates

loop counter

count of number of records succesfully geotagged

current GPS location to be passed from GPSDBASE

date time info to be passed to GPSDBASE

point location returned by GPSDATA

set date and time in DATETIME class to be passed to GPSDBASE by calling user-defined functions that get them from the destination database and return Julian date and time in seconds since midnight

compute coordinates; method returns 0 if time match found and -1 if not

if time match found

get position for record from GPSDATA as 3D point

write coordinates to table

increment success counter

end for

update status prompt in dialog

open tabular view of result table

end proc Run